

EMC[®] Documentum[®] Content Server

Version 6.5

**Administration Guide
P/N 300-007-198-A01**

EMC Corporation
Corporate Headquarters:
Hopkinton, MA 01748-9103
1-508-435-1000
www.EMC.com

Copyright © 1994 - 2008 EMC Corporation. All rights reserved.

Published July 2008

EMC believes the information in this publication is accurate as of its publication date. The information is subject to change without notice.

THE INFORMATION IN THIS PUBLICATION IS PROVIDED AS IS. EMC CORPORATION MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND WITH RESPECT TO THE INFORMATION IN THIS PUBLICATION, AND SPECIFICALLY DISCLAIMS IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Use, copying, and distribution of any EMC software described in this publication requires an applicable software license.

For the most up-to-date listing of EMC product names, see EMC Corporation Trademarks on EMC.com.

All other trademarks used herein are the property of their respective owners.

Table of Contents

Preface	25
Chapter 1	Introduction	27
	Essential concepts.....	27
	Installation components.....	28
	Configuration choices	28
	Configuration objects.....	29
	Administration tasks	29
	User privilege requirements for administration tasks	31
	Administration interfaces.....	32
	Starting Documentum Administrator.....	32
	Using the Content Server Manager on Windows.....	32
	Using DQL for administrative tasks	33
	Documentum tool suite.....	33
	Administration methods	33
	The dm_error utility	34
	Viewing connected users.....	34
	Where to look for more information.....	34
Chapter 2	Content Repositories	37
	Essential concepts.....	37
	Repository and server connections.....	38
	Repository configuration.....	38
	Adding additional repositories	39
	Adding a repository.....	40
	Configuring the new repository for use with Media Transformation Services	40
	Contents of new repositories	41
	Managing cabinets and folders	42
	Public and private cabinets.....	43
	Home cabinets.....	43
	Creating folders and cabinets	43
	Changing and deleting folders and cabinets	43
	Setting the dd_locales property	44
	Manipulating type indexes.....	44
	Alternate locations for object-type tables on Oracle and DB2	45
	Configuring storage and handling of date values	45
	Enabling a repository as a global registry	46
	Dumping and loading a repository	48
	Code page compatibility issues.....	48
	Supporting object types.....	48
	Execution methods	49

Dumping a repository	50
Dumping objects under retention	50
Aspects and dump operations	51
Dumping an entire repository	51
Dumping specific objects	51
Setting the type property	52
Setting the predicate properties	53
Content files and dumping	54
Dumping without content	54
Including content	55
Compressing content	55
Setting the cache size	55
Using non-restartable dump	56
Using a script to create a dump file	56
Sample script for a full repository dump with content included	57
Sample script for a partial repository dump	57
If the server crashes during a dump operation	59
Moving the dump file	59
Loading a repository	59
Refreshing repository objects from a dump file	60
Loading job objects	60
Loading registered tables	61
Turning off save event generation during load operations	61
Loading a new repository	61
The preLoad utility	62
Load procedure for new repositories	62
DocApps	64
Generating dump and load trace messages	64
Creating location and mount point objects	64
Location objects	65
Mount point objects	66
Platform aliases	66
Format objects	67
The DOS extension property	68
The format_class property	68
Listing current format objects	68
Adding format objects	69
Using DQL	69
Rich media formats	69
Modifying formats	70
Using DQL	70
Deleting formats	71
Using DQL	71
Alias sets	71
Creating an alias set	72
Modifying or deleting an alias set	72
Working with object types	72
Creating a user-defined type	72
Using DQL	73
Modifying an object type	73
Deleting properties	74
Using DQL	74
Changing the default permissions or default storage area	74
Adding a property	74
Deleting a property	75
Lengthening a string property	75

Deleting a type	76
Cleaning up repositories	76
Maintaining query performance	79
Using Update Statistics	79
Setting the DM_GROUP_LIST_LIMIT environment variable	79
Configuring repository-level package name control	80
Chapter 3 Servers	83
Overview of servers.....	84
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
JBoss application server	86
Internationalization	87
The dm_start_repositoryname script (UNIX).....	88
The server.ini file.....	88
SERVER_STARTUP section	89
DOCBROKER_PROJECTION_TARGET sections	93
FUNCTION_SPECIFIC_STORAGE and TYPE_SPECIFIC_	
STORAGE sections	94
FUNCTION_EXTENT_SIZE and TYPE_EXTENT_SIZE sections	95
Keys set during installation	96
docbase_id	96
docbase_name.....	96
database_owner	96
database_conn.....	96
Oracle database_conn value	96
Sybase database_conn value	97
MS SQLServer database_conn value	97
DB2 database_conn value.....	97
database_name.....	97
Sybase database_name value	97
MS SQL Server database_name value.....	97
service	98
host	98
Optional keys.....	98
acl_update_threshold.....	98
check_user_interval	99
commit_read_operations.....	99
data_store and index_store	99
gethostbyaddr	100
history_sessions and history_cutoff	100
max_ftacl_cache_size	100
max_nqa_string.....	101
max_sessions_heap_size	101
owner_xpermit_default.....	101
saveasnew_retain_source_group	101
umask (UNIX only)	101
upd_last_chg_time_from_db	102
use_group_address.....	102
validate_database_user	102
Default keys	103

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

Configuring the workflow agent.....	123
Changing the number of worker sessions	124
Changing the sleep interval	124
Disabling the workflow agent.....	124
Tracing the workflow agent.....	124
Recovering automatic activity work items on Content Server failure	125
Managing the JBoss application server	125
Location of binaries	126
Starting and stopping the JBoss application server	126
Chapter 4 Methods and Jobs	129
Introducing methods	129
Execution agents	130
The dmbasic method server.....	130
Java method server	131
Content Server	131
Choosing the execution agent	132
Performance considerations.....	132
Security considerations	133
Tracing options for methods.....	133
Defining the java.ini file (UNIX only).....	134
java_library_path.....	134
java_version	134
java_classpath	135
java_alias_file	135
java_disabled	135
Enabling the dmbasic method server	135
Configuring the worker threads in the dmbasic method server	136
Implementing a method.....	136
Creating a method to be executed by Content Server or the dmbasic method server	136
Limitation on argument length for Docbasic.....	137
General guideline for the script or program.....	137
Script or program return values for workflow methods.....	137
Calling Java or DFC in a Docbasic script.....	137
User account (UNIX only)	137
dmbasic executables (UNIX only)	138
Locating the java.ini file (UNIX only)	138
Sample code.....	138
Recording the output	140
Setting method object properties for Content Server execution.....	140
Setting method object properties for dmbasic method server execution	140
Creating a method to be executed by the Java method server.....	141
General guideline for the script or program.....	141
Guidelines for a Java method to be deployed as a BOF module	141
Script or program return values for workflow methods.....	141
Recording the output	142
Storing Java methods	143
Setting method object properties for Java method server execution	143
Creating a method object.....	143
Using DQL.....	144
Defining success return codes and success status	144
Executing a method on demand	145

Creating jobs and job sequences.....	145
Introducing jobs	146
Introducing job sequences	146
Repository implementation	147
Job sequence execution	147
Determining success for invoked jobs	148
The repository connection file.....	148
The agent exec process.....	148
Creating a job	149
Using DQL to create a job.....	149
Creating a job sequence.....	150
Scheduling jobs	150
Defining a job schedule	151
Passing arguments.....	151
The run_now property.....	152
Managing jobs.....	153
Activating or inactivating a job	153
Disabling all jobs	153
Modifying agent exec behavior	154
Setting the polling interval	154
Setting the number of jobs in a polling cycle	154
Turning on tracing for the agent exec process	155
Creating and maintaining a repository connection file for job sequences.....	155
Specifying the server connect string	155
Commas and backslashes in the entries	156
The dcf_edit utility	156
Recovering from a job sequence failure	158
Interpreting a job sequence status report	158
Executing dm_run_dependent_jobs independently.....	159
Chapter 5	
Managing Repository Sessions	161
Terminology.....	162
The dfc.properties file	162
Key format.....	162
Setting dfc.properties entries	163
Defining connection brokers for connection requests.....	163
Specifying connection brokers	163
Failover and load balancing.....	164
Requesting a specific server connection.....	165
Requesting a server by name	165
Requesting a server on a specific host	165
Requesting a server by name on a specific host	165
Turning off trusted login	166
Defining the secure connection default for connection requests.....	166
Configuring the number of connection attempts and the retry interval	167
Specifying the maximum number of sessions	167
Limiting which clients can access a repository	168
Configuring privileged DFC use	168
Creating client rights objects.....	168
Configuring a repository to accept only authenticated DFC instances	169
Enabling privileged DFC.....	169

Disabling privileged DFC.....	170
Modifying the Java security policy	170
Managing the keystore file	171
Changing the location and name of the keystore file	171
Changing the passwords used with keytool.....	171
Configuring a shared keystore file	172
Configuring connection pooling	172
Enabling connection pooling	172
Configuring login tickets.....	173
Setting ticket validity period.....	173
Setting the ticket cache size for Content Server	173
Configuring login tickets for backwards compatibility	173
Changing a session's configuration	174
Changing the assigned default operating system permissions (UNIX only)	175
Defining short date formats.....	175
Changing the client local area directory location	176
Setting disk space limits for the client local area	176
Removing content from client local areas	176
Manual clean up.....	177
Managing persistent client caches	177
Enabling and disabling persistent client caching	178
For a repository	178
For client sessions.....	178
Creating cache config objects.....	179
Defining the cached data set.....	179
Defining the server check interval.....	180
Defining the client check interval.....	181
Manually forcing refreshes	181
Flushing a persistent cache.....	181
Setting the client_pcaching_change property	182
Automating cache config data validation	182
Overriding consistency checking rules	183
Defining the persistent cache write interval	183
Troubleshooting persistent caching.....	184
Chapter 6 Connection Brokers	187
An overview of connection brokers.....	187
How many connection brokers are there?.....	188
What information does a connection broker have?	188
How does a connection broker get information?	188
Locating connection brokers.....	189
Connection broker configuration options	189
Servers and connection brokers	189
Clients and connection brokers.....	190
Failover for connection brokers	191
Load balancing for connection brokers.....	191
Configuring a connection broker	191
Connection broker initialization file	192
Invoking the initialization file	193
Configuring shutdown security (UNIX only).....	193
Restricting server access.....	193
Translating IP addresses.....	194

Restarting a connection broker	195
Windows platforms	195
UNIX platforms.....	195
Starting additional connection brokers	196
Shutting down a connection broker	197
Windows platforms	197
UNIX platforms.....	198
Multiple connection brokers on UNIX.....	199
Obtaining information from a connection broker	199
The getServerMap method	200
The getDocbaseMap method	200
Obtaining information about connection brokers	200
Using the getDocbrokerMap method	201
Querying the client config object.....	201
Deleting server information.....	201
Chapter 7 Content Management	203
Storage area options	204
File store storage areas	205
Public and private file store areas	205
Content encryption.....	206
Content compression	206
Content compression characterization.....	207
Content duplication checking and prevention.....	209
How checking and prevention work	209
Supporting properties.....	209
content_hash_mode and content_dupl_pref	210
r_content_hash	210
Tracing duplication checking and prevention	211
Digital shredding	211
EMC Centera storage	212
Configuration options.....	212
Default retention values	213
Compression.....	213
Whether to link or embed the content in the C-clip	214
The C-clip buffer size	214
Write retries	214
Override of the Centera storage strategy configuration	215
Maximum number of socket connections used by the Centera SDK	215
Use of the memory map interface for writes to the storage area	215
NetApp SnapLock storage	215
Configuration options.....	216
Default retention values	216
Compression.....	217
Blob store storage areas.....	217
Turbo storage	218
Distributed storage areas.....	218
External storage areas	219
Use constraints	219
Types of external storage areas	219
Plug-in objects for external storage	220
Linked store storage areas	221
Summary of storage area configuration options.....	222
Content and full-text indexes.....	223

How objects, contents, and storage are connected	223
Allocating content to storage areas	224
Using content assignment policies	225
What content assignment policies are.....	225
Creating content assignment policies	225
Enforcement of assignment policies	226
Behavior on encountering a rule error	227
DFC assignment policy information cache.....	227
Architectural implementation of assignment policies	227
Algorithm used by the DFC policy engine	228
Assignment policy administration	229
Using the default storage algorithm	229
Primary content algorithm	229
Rendition algorithm.....	231
Content Retention.....	231
System-defined storage areas	232
File paths and URLs for content files in storage.....	233
Path specifications for content in file stores.....	233
URL specifications for content files	234
Setting up storage.....	235
Distributed storage setup	236
Setting up blob storage	236
Using DQL to set up blob storage	236
Setting up file store storage areas.....	237
File extensions and the use_extensions property	237
Setting the base URL.....	238
Defining file store storage areas as public (Windows only)	238
Using DQL to set up file storage	238
Linked store setup	239
Using DQL to set up linked storage	240
Setting up external storage	242
An example of importing documents stored on a CD-ROM.....	242
Using the Mount method	243
External URL storage setup	244
Setting up external free storage	244
Configuring for optimal performance on retrieval.....	245
Setting up EMC Centera storage areas.....	245
Setup procedure	246
Defining storage area retention requirements	247
Defining the connection string	248
Required privileges for connection strings.....	249
Defining a connection string supporting EMC Centera clusters	249
Example of use	250
Configuring embedded blob use.....	251
Setting the C-clip buffer size.....	252
Configuring write attempts in EMC Centera storage areas	253
Overriding the Centera single-instancing configuration	253
Resetting the maximum socket connections allowed	254
Configuring use of a memory map for write operations	254
Setting clocks and time zones for Centera hosts and Content Server hosts.....	255
Setting up NetApp SnapLock storage areas	255
Setup procedure	256
Enabling content compression	256
Defining SnapLock storage area retention requirements.....	256
Setting up turbo storage	257

Providing automatic file extensions	258
Moving content files	259
MIGRATE_CONTENT administration method	259
Content migration policies	260
Configurable arguments	261
Generated log files.....	262
Records migration jobs.....	262
Auditing content movement.....	263
Maintenance operations for storage areas.....	263
Changing the state of a storage area	263
Determining the state of a storage area.....	264
Moving file store storage areas	264
Enabling forced deletion in EMC Centera storage areas.....	265
Removing orphaned content objects and files	266
Using dmclean	267
Including content in retention type storage areas	268
Running dmclean using an EXECUTE statement	268
Running dmclean from the operating system prompt	269
Executing the dmclean script.....	269
Using dmfilesan.....	270
dmfilesan arguments	270
Identifying the subdirectories of the scanned storage areas.....	271
Using the -no_index_creation argument.....	272
Using the -force_delete argument	272
Running dmfilesan using an EXECUTE statement.....	273
Running dmfilesan from the operating system prompt.....	273
The generated script	273
Executing the dmfilesan script	274
Replacing a full distributed storage component	274
Resolving a compromised file store key	275
Administering content assignment policies	275
Logging policy use	275
Enabling and disabling assignment policies.....	276
Turning off the policy engine.....	276
Configuring behavior on encountering a rule error	276
Configuring the update interval for the policy information cache.....	277
Archiving and restoring documents.....	277
How the process works	277
The archive and restore methods	280
The Archive tool.....	280
Archiving.....	280
Restoring	281
Moving dump files on and off line	282
Archiving content used in multiple documents.....	283
Options for archiving	283
Scheduling archiving	283
Types of requests.....	284
The repository operator	284
Moving the dump file in and out of the archive directory	284
Choosing an archive directory	285
Implementing archiving.....	285
Starting the Archive tool	286
Restoring documents	286
Archiving restored documents	286
Custom restoration	287
Chapter 8 Users and Groups	289

User names	290
User privilege levels	291
Basic user privileges	291
Extended user privileges	293
Privileged groups	294
Adding users	295
Setting user properties	296
User privileges	297
Defining the default ACL	297
Setting user_db_name	298
Setting default_folder	298
Setting accessible folders	298
Setting client capability	299
Creating a new user with DQL	299
Adding multiple users in a single operation	299
LDIF file contents	300
Setting up the file	300
Extended characters in the file	301
Granting and revoking user privileges	301
Superuser privileges and the admingroup group	302
Granting and revoking privileges using DQL	302
Granting and revoking privileges using DFC	303
Modifying users	303
Using DQL	304
Renaming users	304
Deleting users	304
Using DQL	305
Deactivating, locking, and reactivating users	305
Deactivating or locking users	305
Reactivating users	306
Using DQL to change a user's login state	306
Adding groups	307
Using DQL	308
Modifying groups	309
Using DQL	309
Deleting groups	309
Using DQL	310
Changing the membership setting of a dynamic group	310
Querying groups	311
Obtaining a list of members in a group	311
Obtaining a list of groups with a count of the members in each	311
Chapter 9 Managing User Authentication	313
Authentication options	313
Default mechanism	314
Custom password checking program	314
LDAP directory server	314
Authentication plug-in	314
In-line password	315
The assume user and change password programs	315
Assume user	315
Change Password	316
Using the default authentication mechanism	316

UNIX platforms.....	316
Windows platforms	317
Authenticating in domains	317
No-domain required mode.....	318
Domain-required mode.....	318
Determining the repository's authentication mode.....	318
Converting to domain-required mode.....	319
Using a custom external password checking program	319
Basic steps.....	319
Writing the program	320
Using Windows domain authentication for UNIX users	320
Modifying the dm_check_password program.....	321
Setting auth_protocol.....	322
Setting up the domain controller map	322
Setting the user_source property	323
Using an LDAP directory server	323
Benefits	324
Constraints	324
Integrating an LDAP directory server with a repository	324
Using multiple LDAP directory servers.....	325
User and group synchronization.....	325
Synchronization and federations	325
dm_LDAPSynchronization job	325
How the synchronization job determines which LDAP servers to use	327
Attributes set by the dm_LDAPSynchronization job	327
On-demand user synchronization.....	328
User authentication	328
Authentication and federations	328
How Content Server determines which server to use for authentication	329
LDAP authentication options	330
Connection retry attempts.....	330
Authentication failover	330
Failover for extra directory LDAP servers	331
Tracing failover	332
Implementing an LDAP directory server.....	332
Defining the set-up values.....	333
Distinguished name and bind type	334
Search bases and filters	334
The secure connection properties.....	335
Mapping LDAP attributes to repository user or group properties	335
Required mappings	336
Defining mapping rules	336
Mapping guidelines.....	337
Repository storage of mappings	337
Mapping examples	338
Downloading certutil and the certificate authorities.....	340
Installing the certificate database and CA certificates	340
Activating the dm_LDAPSynchronization job	341
Building and installing an LDAP-enabled password checking program (UNIX only)	341
Note on using Active Directory	342
Note on using LDAP directory servers with multiple Content Servers.....	342
Deleting an LDAP directory server from a repository.....	342
Setting the retry_interval property for user authentication.....	343

How the environment variables are used	343
How to set the environment variables	344
Enabling first-time synchronization rules	344
Binding LDAP users to a different directory server	344
Listing certificates in the certificate database	345
Troubleshooting the synchronization job	345
Using authentication plug-ins.....	346
Plug-in scope	346
Identifying a plug-in for use	347
Defining a plug-in identifier	347
Using the RSA plug-in	348
Using the CA SiteMinder plug-in.....	348
Implementing a custom authentication plug-in.....	349
Writing the authentication plug-in	350
Internationalization	350
Tracing authentication plug-in operations	351
Using an in-line password.....	351
Trusted logins	351
Unified logins	351
Managing encrypted passwords	352
Using encryptPassword	353
If you do not want to use encrypted passwords	353
Changing an encrypted password.....	354
Limiting authentication attempts	356
Chapter 10 Protecting Repository Objects	359
Overview of repository security.....	359
ACLs	360
Additional security options	360
Application-level control of SysObjects	361
Dynamic groups.....	361
Folder security	362
User privileges	362
Table permits	362
Turning repository security on and off	362
Turning folder security on and off	363
Changing folder security.....	364
Setting the default permission level for application-level control of SysObjects	364
Object-level permissions	364
Basic permissions	365
Note on the Relate permit level.....	366
Extended permissions	366
Viewing extended permissions	367
Managing ACLs	368
The ACL object type	369
Access control entries.....	369
AccessPermission and ExtendedPermission entries.....	370
AccessRestriction and ExtendedRestriction entries.....	370
AccessRestriction entries	370
ExtendedRestriction entries	371
Storage in the ACL	371
ApplicationPermission entries	371
ApplicationRestriction entries	372

RequiredGroup entries	372
RequiredGroupSet entries	373
How ACL entries are evaluated	374
Evaluation for non-owners and non-superusers	374
How access is evaluated for object owners and Superusers	374
Access evaluation for an object's owner	375
Evaluating a Superuser's permissions	376
Resolving multiple entries for a user	376
Disabling ACL restrictive entries	377
External and internal ACLs	378
ACL names	379
System, public, and private ACLs	379
Template ACLs.....	379
Creating ACLs	380
How ACLs and objects are connected	380
The default ACLs	381
Creating default ACLs	383
Assigning a default ACL to an object	383
Identifying the default ACL for use	383
Modifying an ACL.....	384
Adding entries	384
Removing entries.....	384
Destroying an ACL.....	385
Removing unreferenced external ACLs.....	385
Removing unreferenced internal ACLs	385
Table permits	386
Setting table permits	386
Table permits and object-level permissions.....	387
Table permits and dump and load operations	387
Auditing	387
What events are auditable	388
Audit trails.....	388
Auditing properties	388
If audit_old_values is T	389
If audit_old_values is F	390
Default auditing	391
Turning off default auditing	392
Modifying the default auditing.....	393
Auditing system events.....	393
Auditing application events	393
Signing audit trail entries	394
Conflicting registration resolution	395
Stopping auditing.....	396
Viewing audit trails	396
Querying and retrieving audit trail entries	397
Interpreting audit trails of DFC method, workflow, and lifecycle events	397
Audit trail properties with a common purpose	397
Properties available for varied purposes.....	397
Use in audit trails for events generated by non-workflow or lifecycle methods.....	398
Use in lifecycle audit trails	404
Use in workflow audit trails	405
Interpreting ACL and group audit trails.....	413
Verifying signed audit trail entries	415
Removing audit trail entries	416
Auditing in a distributed environment.....	416
Implementing signature support	417

Customizing electronic signatures	417
Customizing the default functionality	418
Adding or removing properties on the page	420
Changing the property delimiters	421
Configuring the appearance of the page	422
Defining separate templates for different document types	423
Configuring the number of allowed signatures and signature positioning	424
Creating custom signature creation methods and associated signature page templates	425
Creating custom signature-creation methods	425
Creating custom signature page templates	427
Tracing electronic signature operations	427
Supporting digital signatures	427
Customizing simple signoffs	428
Customizing the signature validation program	428
Registering for notification	429
Querying the audit trail for signoffs	429
Managing the encryption keys.....	429
The AEK	430
Sharing the AEK or passphrase	430
The AEK and distributed sites	431
Backing up the AEK.....	431
Repository encryption keys	431
Encryption utilities	432
Using dm_crypto_boot	432
Troubleshooting with dm_crypto_create	433
Changing a passphrase	434
Managing the login ticket key.....	436
Exporting and importing a login ticket key.....	436
Resetting a login ticket key.....	436
Configuring a repository's trusted repositories	437
Configuring login ticket use	437
Configuring the default login ticket timeout	437
Restricting a Superuser's use of global tickets	438
Revoking tickets for a specific repository	438
Troubleshooting a login ticket	438
Configuring application access control token use.....	439
Enabling AAC token use by a server	439
Enabling machine-only AAC tokens	439
Enabling token retrieval by the client library	440
Generating tokens for storage.....	441
Naming the output file.....	443
Storing tokens generated by dmtkgen	443
Troubleshooting an application access control token	444
Chapter 11 Administration Tools	445
Essential tool concepts	446
How tools are implemented	449
Standard arguments	449
The QUEUEPERSON argument.....	450
The window interval.....	450
Reports and trace log files	451
Reports	451
Storage	451
Trace log files	452

Storage	452
Email messages	452
Activating and scheduling administration tools	453
Activation	453
Defining job schedules	453
Running administration jobs on demand.....	454
Archive	454
Audit Management	456
Consistency Checker.....	459
Running the job from a command line	460
Content Replication	465
Content Warning	468
Create Full-Text Events	470
Arguments.....	471
Report sample	474
Data Dictionary Publisher	475
Database Space Warning	476
Dm_LDAPsynchronization	479
Executing dm_LDAPsynchronization manually	482
Explicitly specifying LDAP servers in -source_directory	482
Dmclean	482
Dmfilescan	487
File Report	493
Group Rename	498
Index Agent Startup	498
Log Purge	499
Queue Management	503
Remove expired retention objects	506
Rendition Manager	509
State of the Repository Report	514
Swap Info	518
ToolSetup.....	519
Update Statistics	519
User Chg Home Db	522
User Rename.....	523
Version Management	525
Tool maintenance and troubleshooting.....	529
Changing the default settings	529
Using DQL.....	529
Using the tool trace log files	530
Viewing the tool reports.....	530
Chapter 12 Logging and Tracing Facilities	531
Introduction.....	531
Content Server logging and tracing.....	531
Starting and stopping Content Server tracing operations.....	532
Starting and stopping tracing from the startup command line	532
Using setServerTraceLevel	533

Using SET_OPTIONS	535
Examples of server tracing	535
Determining which tracing options are turned on.....	536
DFC logging.....	536
DFC tracing.....	536
The logger and logging appender	537
Enabling tracing	537
Configuring the logging appender	537
Trace file names.....	539
Defining file creation mode	539
Defining the timestamp format.....	540
Defining a date format	542
Defining what is traced	542
Configuring method tracing	542
Defining maximum stack depth to trace	542
Specifying method entry and exit tracing	543
Identifying which packages, classes, and methods to trace	543
Tracing users by name	544
Tracing threads by name	545
Including the session ID.....	545
Tracing RPC calls	546
Including stack trace for exceptions	546
Setting verbosity	546
Directing categories to the trace file	547
Interactions of tracing specifications	548
Log file entry format.....	548
Trace file examples	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.....	557
Workflow checks	557
Object type checks	558
Data dictionary checks.....	559
Lifecycle checks.....	560
Object type index checks	561
Method object consistency checks	561
Appendix B IDQL and IAPI	563
Using IDQL.....	563
Starting IDQL.....	563
The IDQL commands.....	566
Entering queries	567
Clearing the buffer.....	567
Entering comments.....	567
Stopping IDQL.....	568
Using IAPI	568
Starting IAPI	568
IAPI commands.....	571

	Entering method calls	572
	Entering comments.....	573
	Quitting an IAPI session	573
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 properties	597
	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	606
	The data dictionary publisher job	607
	The publishDataDictionary method	607
	The PUBLISH key phrase	608
Appendix F	System Events	609
	dm_default_set event.....	609
	DFC events	610
	Workflow events	615
	Lifecycle events	618
	Events sent to the fulltext user	619
Appendix G	Plug-in Library Functions for External Stores	621
	General recommendations	621
	dm_close_all	622
	dm_close_content	622
	dm_deinit_content.....	622
	dm_init_content	623
	dm_open_content.....	623
	dm_plugin_version.....	625
	dm_read_content.....	625

List of Figures

Figure 1.	Small file compression percentages.....	207
Figure 2.	Medium file compression percentages	208
Figure 3.	Large file compression percentages.....	208
Figure 4.	Relationship of content assignment policies and object types	228
Figure 5.	Relationship between a file store object and a location.....	237
Figure 6.	The objects that define a linked store storage area	240
Figure 7.	Content Server and EMC Centera cluster configuration in single-repository distributed environment	251
Figure 8.	The archive and restore process	279
Figure 9.	Default signature template	418

List of Tables

Table 1.	Users created during repository configuration.....	41
Table 2.	Groups created during repository configuration.....	41
Table 3.	Permissions required to modify or delete cabinets and folders.....	44
Table 4.	Example of date handling behavior for combinations of property values and release versions	46
Table 5.	security_type property settings for location objects	65
Table 6.	Interaction of properties that determine package control behavior	81
Table 7.	Locale-based configuration settings by host machine locale.....	87
Table 8.	Server.ini SERVER_STARTUP section keys.....	89
Table 9.	Server.ini DOCBROKER_PROJECTION_TARGET keys	94
Table 10.	Server.ini FUNCTION_SPECIFIC_STORAGE keys	94
Table 11.	Server.ini TYPE_SPECIFIC_STORAGE keys	95
Table 12.	Server.ini FUNCTION_EXTENT_SIZE keys	95
Table 13.	Server.ini TYPE_EXTENT_SIZE keys	95
Table 14.	Standard arguments passed to jobs	152
Table 15.	dcf_edit utility arguments	156
Table 16.	Entries in a job sequence statusreport.....	158
Table 17.	dm_run_dependent_jobs arguments	159
Table 18.	Summary of storage area configuration options.....	222
Table 19.	Availability of retention functionality.....	232
Table 20.	Examples of URLs and formats	244
Table 21.	dmclean arguments	267
Table 22.	Arguments to the dmfilesan utility.....	270
Table 23.	Basic user privileges	291
Table 24.	Extended user privileges	293
Table 25.	Privileged groups	294
Table 26.	Security properties for new users.....	297
Table 27.	Default values for new users	300
Table 28.	RSA plug-in modules.....	348
Table 29.	CA SiteMinder plug-in files.....	349
Table 30.	Password files encrypted by default.....	352
Table 31.	Repository security settings.....	362
Table 32.	Object-level permissions	365
Table 33.	Permitted operations for object-level permissions	365
Table 34.	Extended object-level permissions	366
Table 35.	The extended permission computed properties	367
Table 36.	Table permits	386

Table 37.	Behaviors for audit properties combinations	389
Table 38.	Events audited by the dm_default_event.....	392
Table 39.	Usage of generic properties by API events.....	398
Table 40.	Usage of generic properties by lifecycle events	404
Table 41.	Usage of generic properties by workflow events	405
Table 42.	System-defined document properties for the signature page template.....	419
Table 43.	System-defined signature properties for the signature page template	420
Table 44.	Parameters passed by addESignature to the signature-creation method	425
Table 45.	Interaction of dm_crypto_change_passphrase arguments.....	435
Table 46.	dmtkgen utility arguments.....	441
Table 47.	EMC Documentum tool suite	446
Table 48.	Standard arguments passed to jobs	449
Table 49.	Archive arguments	454
Table 50.	Audit Management arguments	457
Table 51.	Content Replication arguments	467
Table 52.	Content Warning arguments	469
Table 53.	Create Full-Text Events arguments.....	471
Table 54.	Data Dictionary Publisher argument	475
Table 55.	Database Space Warning arguments	477
Table 56.	dm_LDAPsynchronization arguments.....	480
Table 57.	Dmclean arguments.....	483
Table 58.	Dmfilescan arguments	488
Table 59.	File Report arguments.....	495
Table 60.	Files deleted by the Log Purge tool	499
Table 61.	Log Purge arguments.....	500
Table 62.	Queue Management arguments.....	504
Table 63.	Remove Expired Retention Objects arguments.....	507
Table 64.	Rendition Management arguments	509
Table 65.	State of the Repository arguments	514
Table 66.	Swap Info arguments.....	518
Table 67.	Update Statistics arguments	521
Table 68.	Version Management arguments	526
Table 69.	Server start-up trace options.....	533
Table 70.	Trace level severity values	534
Table 71.	Valid facilities for setServerTraceLevel	534
Table 72.	Logging appender configuration options	537
Table 73.	Valid values for dfc.tracing.file_creation_mode.....	540
Table 74.	Valid values for dfc.tracing.timing_style.....	541
Table 75.	Log entry components	548
Table 76.	Consistency checks for users and groups.....	552
Table 77.	Consistency checks for ACLs	553
Table 78.	Consistency checks for SysObjects	554
Table 79.	Consistency checks for folders and cabinets	555

Table 80.	Consistency checks for documents.....	556
Table 81.	Consistency checks for content objects	557
Table 82.	Consistency checks for workflows	557
Table 83.	Consistency checks for object types	558
Table 84.	Consistency checks for the data dictionary	559
Table 85.	Consistency checks for lifecycles.....	560
Table 86.	Consistency checks for object type indexes	561
Table 87.	Consistency checks for method objects.....	562
Table 88.	IDQL command line arguments.....	564
Table 89.	IDQL commands	566
Table 90.	IAPI command line arguments	569
Table 91.	IAPI commands.....	571
Table 92.	Monitoring Scripts.....	589
Table 93.	Codepages for the data dictionary files provided with Content Server	596
Table 94.	Settable data dictionary properties using population script	597
Table 95.	Events represented by the dm_default_set event.....	609
Table 96.	Events arising from DFC methods	610
Table 97.	Workflow events	615
Table 98.	Lifecycle events	618
Table 99.	dm_close_all arguments.....	622
Table 100.	dm_close_content arguments	622
Table 101.	dm_init_content arguments.....	623
Table 102.	dm_open_content arguments	623
Table 103.	dm_plugin_verson arguments.....	625
Table 104.	dm_read_content arguments.....	625

Preface

This manual contains information, instructions, and procedures for the normal system administration tasks of a Documentum installation. It provides an overview of the system and guidelines to customize the configuration. 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 should 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.

Revision history

The following changes have been made to this document.

Revision history

Revision	Description
July 2008	Initial publication of Documentum Content Server Administration Guide 6.5

Introduction

This chapter includes the following topics:

- [Essential concepts, page 27](#)
- [Administration tasks, page 29](#)
- [User privilege requirements for administration tasks, page 31](#)
- [Administration interfaces, page 32](#)
- [Documentum tool suite, page 33](#)
- [Administration methods, page 33](#)
- [The dm_error utility, page 34](#)
- [Viewing connected users, page 34](#)
- [Where to look for more information, page 34](#)

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

Essential concepts

An EMC 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 also be modified later.

Keep your initial installation simple to ensure best results. After you are familiar with EMC Documentum, you can modify the installation if necessary.

Installation components

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 EMC Documentum can run on—are found in the *Content Server Release Notes*. This chapter provides some guidelines to decide the type of configuration for your installation.

Configuration choices

You have a number of choices for your installation, which 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 relational database management system (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 System Installation and Administration Guide*, and *Distributed Configuration Guide* contain information on system architecture. [Chapter 3, Servers](#) and [Chapter 2, Content Repositories](#) 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

There are various configuration choices for a repository, including what types of storage areas will you use and how user connection attempts are handled. [Chapter 2, Content Repositories](#) contains information on the choices available to you when configuring a repository.

- Client configuration

From the client side, there are many configuration choices. For example, you can determine how a particular client shares files with the server, how to display dates, and which connection broker to use. [Chapter 5, Managing Repository 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 to put the installation. If you modify the basic installation, then the number and scope of your decisions 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 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 2, Content Repositories](#), [Chapter 3, Servers](#) and [Chapter 5, Managing Repository Sessions](#) contain information on these tasks. The *Distributed Configuration Guide* contains instructions on creating distributed configurations and federations.

- Creating new repositories

Installation of the Content Server creates one repository, one server, and one connection broker. Your organization may want additional repositories. Instructions for creating new repositories are found in the *Content Server Installation Guide*, with some guidelines also in [Chapter 2, Content Repositories](#).

- Maintaining content repositories

Organizations 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

The *Content Server Installation Guide* and [Chapter 3, Servers](#) contain procedures for server administration.

- Changing session configurations

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

- **Maintaining connection brokers**

The connection broker is an EMC | Documentum 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**

The *Content Server Full-Text Indexing System Installation and Administration Guide* describes full-text indexes and includes procedures for setting up and maintaining full-text indexes.
- **Managing users and groups**

[Chapter 8, Users and Groups](#) contains procedures for adding, changing, and removing local users and groups.
- **Managing security**

Documentum security features include object-level permissions (implemented as ACLs), table permits, user privilege levels, and auditing and tracing capabilities.

[Chapter 10, Protecting Repository Objects](#) describes the Documentum implementation of 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 9, Managing User Authentication](#).

User privileges are described in [Chapter 8, Users and Groups](#).

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

User privilege requirements for administration tasks

Most of the system administration tasks in this manual require at least the Sysadmin user privilege. Some tasks, 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 *Documentum Administrator Deployment Guide*.) Documentum Administrator includes an extensive online Help system.

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

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 IDQL utility
- 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 DQL for administrative tasks

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

- Documentum Administrator
- IDQL utility

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. [Using IDQL, page 563](#) provides information about using IDQL.

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 11, Administration Tools](#). The 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 a DFC `apply()` method or the DQL `EXECUTE` statement. For complete information about these methods, refer to the *Content Server DQL Reference Manual*.

The `dm_error` utility

The `dm_error` utility is a utility that returns information about a specified Content Server 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, Managing Repository Sessions](#).

Where to look for more information

This guide 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 and DQL statements, refer to the reference manuals:

- *Documentum System Object Reference*
- *Content Server DQL 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.

The *Content Server Full-Text Indexing System Installation and Administration Guide* describes the possible full-text configurations and how to install and maintain them.

Content Repositories

This chapter contains the following information about repositories:

- [Essential concepts, page 37](#)
- [Adding additional repositories, page 39](#)
- [Managing cabinets and folders, page 42](#)
- [Setting the dd_locales property, page 44](#)
- [Manipulating type indexes, page 44](#)
- [Alternate locations for object-type tables on Oracle and DB2, page 45](#)
- [Configuring storage and handling of date values, page 45](#)
- [Enabling a repository as a global registry, page 46](#)
- [Dumping and loading a repository, page 48](#)
- [Creating location and mount point objects, page 64](#)
- [Format objects, page 67](#)
- [Alias sets, page 71](#)
- [Working with object types, page 72](#)
- [Cleaning up repositories, page 76](#)
- [Maintaining query performance, page 79](#)
- [Configuring repository-level package name control, page 80.](#)

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, in a retention storage system such as EMC Centera or NetApp SnapLock, or on external storage devices.

A full-text index is associated with a particular repository or, in a consolidated deployment, with all repositories indexed by a particular index server installation. Full-text indexes enable rapid searching for designated values or strings in content files and property values.

Users access repositories through Content Servers. The Content Servers receive queries from clients in the form of DFC 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, users cannot access that repository.

Repository and server connections

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 112](#) for instructions. If a repository is very active, serving many users, or its users are widely spread geographically, multiple servers can provide load balancing and enhance performance. Multiple servers also enables you to dedicate one server to a particular application or group of users and have other servers available to everyone.

Repository configuration

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

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

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 following objects define the repository structurally:

- 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. The use of mount point objects alleviates the need to mount multiple directories on clients.
- **Format objects**
Format objects define file formats. A server recognizes a file format only 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 a fulltext index object in the repository. The basic installation is configured for full-text indexing. For information about managing full-text indexes and directories, refer to the *Content Server Fulltext Indexing System Installation and Administration Guide*.
You must have Sysadmin or Superuser user privileges to change a repository's docbase config object.

Adding additional repositories

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 several questions about the new repository, such as its ID and name, and uses the answers to build the Content Server startup 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.

Adding a repository

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 *Content Server Installation Guide*.

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` property 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` property correctly for the new repository.

Note: 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 1, page 41](#), lists the users that were created during repository configuration and the privileges each has by default.

Table 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
<i>global_registry_user</i>	0 (None)	None

A default user name is suggested during installation, but a different name may be chosen.

<i>dm_bpm_inbound_user</i>	0 (None)	None
<i>dm_autorender_win31</i>	8 (Sysadmin)	None
<i>dm_autorender_mac</i>	8 (Sysadmin)	None
<i>dm_mediaserver</i>	8 (Sysadmin)	None
<i>dm_fulltext_index_user</i>	16 (Superuser)	None

Configuring a repository also creates a number of groups. [Table 2, page 41](#), lists the basic groups. In addition, a set of privileged groups are also created. For a listing of the privileged groups, refer to [Chapter 8, Users and Groups](#).

Table 2. Groups created during repository configuration

Group	Members
admingroup	<i>repository owner</i> <i>installation owner</i>

Group	Members
docu	<i>repository owner</i> <i>installation owner</i> dm_autorender_win32 dm_autorender_mac dm_mediaserver
queue_admin	None This is a role group supporting the queue management feature of Documentum Process Builder.
queue_manager	queue_admin group This is a role group supporting the queue management feature of Documentum Process Builder.
queue_processor	queue_manager group This is a role group supporting the queue management feature of Documentum Process Builder.
process_report_admin	queue_admin This is a role group supporting the queue management feature of Documentum Process Builder.

Managing cabinets and folders

The objects in a repository are organized by 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 by using one of the Documentum clients designed for that purpose or a customized client interface. Documentum Administrator also allows you to link objects to a folder or cabinet. Applications use an `IdfSysObject.link` method to perform these operations.

Public and private cabinets

A cabinet's `is_private` property 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 property 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.

You can create folders and cabinets by using Documentum Administrator or another Documentum client, such as Webtop. (If you need instructions, refer to the online help system for the client you use.) You can also use DQL.

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

For a complete list of cabinet and folder properties, refer to the *Documentum System Object Reference Manual*.

Changing and deleting folders and cabinets

You can modify or delete cabinets and folders by using Documentum Administrator, another Documentum client (such as Webtop), or DQL. Using Documentum Administrator allows you to:

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

If you use DQL, use an `UPDATE...OBJECT` or `DELETE...OBJECT` statement.

Table 3. Permissions required to modify or delete cabinets and folders

Action	Permission required
Changing a cabinet or folder's properties	Minimum of Write permission on the object.
Adding or removing links for a folder	Minimum of Write permission on the folder.
Deleting a folder	Delete permission for the folder.
Deleting a cabinet	Superuser, Sysadmin, or Create Cabinet privilege.

Setting the dd_locales property

The dd_locales property in the doabase config object records the data dictionary locales recognized by the Content Server. This property 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 property manually, and you must reinitialize the server after setting the property to make the change visible to the server.

The locale value is stored in the property 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 or you can move the indexes later, using the MOVE_INDEX function. Indexes that you create can be placed in any directory. The *Content Server Installation Guide* provides instructions to create a repository with separate tablespaces or segments.

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 the *Content Server DQL Reference Manual*. Each can be executed

through Documentum Administrator, an 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 can control where repository information is stored. For example, you can store frequently used data on disks different than those used for 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 *Content Server Installation Guide*.

Configuring storage and handling of date values

By default, date values are stored as follows:

- As UTC (Coordinated Universal Time) time in new repositories (Documentum 6 and later).
- As the Content Server's local time in repositories that are upgraded from before version 6.

The `r_normal_tz` property in the doabase config object controls how Content Server stores dates in the repository. If set to 0, all dates are stored in UTC time. If set to an offset value, dates are normalized using the offset value before being stored in the repository. If set to an offset value, the property must be set to a time zone offset from UTC time, expressed as seconds. For example, if the offset represents the Pacific Standard Time zone, the offset value is $-8*60*60$, or -28800 seconds. When the property is set to an offset value, Content Server stores all date values based on the time identified by the time zone offset.

For example, suppose a client application in the Pacific Standard Time zone sends the date 10/14/2006 1:00:00 p.m. to a Content Server in the Eastern Standard time zone. [Table 4, page 46](#), summarizes how Content Server handles the date on reception depending on whether the client is a Version 6 client or an earlier client.

Table 4. Example of date handling behavior for combinations of property values and release versions

Property settings	Version 6 (or later) Content Server in Eastern Standard Time zone receives 10/14/2006 1:00:00 p.m. from:	
	Version 6 (or later) client in Pacific Standard time zone	Version 5.3 or earlier client in Pacific Standard time zone
r_normal_tz= 0	Content Server converts the received date (10/14/2006 1 p.m.) into server local time, which is 10/14/2006 4 p.m. Then, it converts that server local time into UTC time (10/14/2006 9 p.m.) and saves that into the repository.	Because the client application is earlier than version 6, Content Server assumes that the received date (10/14/2006 1 p.m.) represents server local time. Content Server converts 10/14/2006 1 p.m. to UTC time (10/14/2006 6 p.m.) and saves it into the repository.
r_normal_tz = -18000 (-5*60*60)	Content Server converts the received date (10/14/2006 1 p.m.) into server local time, which is 10/14/2006 4 p.m. Content Server saves 10/14/2006 4 p.m. in the repository because the server local time is the time zone identified by the offset.	Because the client application is earlier than version 6, Content Server assumes that the received date (10/14/2006 1 p.m.) represents server local time. Content Server saves 10/14/2006 1 p.m. in the repository because the server local time is the time zone identified by the offset.

In a new Documentum 6 or later repository, r_normal_tz is set to 0. In a repository upgraded from a release earlier than version 6, r_normal_tz is set to the offset that represents Content Server local time and cannot be changed.

Enabling a repository as a global registry

To enable a repository as a global registry after configuration, you must activate the dm_bof_registry user.

To enable a repository as a global registry:

1. Access Documentum Administrator in a browser and connect to the repository.
2. Navigate to **Administration > User Management > Users**.
3. Locate the user named `dm_bof_registry` and select **View > Properties > Info** to access the **User Properties - Info** page.
4. Verify that the username attribute is set to `dm_bof_registry`.
The value `dm_bof_registry` is required.
5. Optionally, change the user login name to a new value.
6. Change the user's password.
7. Set the `dm_bof_registry` user's status to **Active**.
8. Click **OK** to save the user.
9. During DFC installation on client machines, such as the Webtop or Documentum Administrator hosts, provide the user login name and password.
This updates the `dfc.properties` file and enables that DFC installation to contact the global registry as required.
10. To manually modify the `dfc.properties` file to designate a global registry repository and user credentials:
 - a. On the DFC host, navigate to `$DOCUMENTUM/config` (UNIX or Linux) or `%DOCUMENTUM%\config` (Windows).
 - b. From a command prompt, execute the following command to generate the encrypted form of the global registry user's password:


```
java -cp dfc.jar com.documentum.fc.tools.RegistryPasswordUtils
password_of_user
```

 where `password_of_user` is the global registry user's clear-text password. In step d, enter the encrypted form of this password in the `dfc.properties` file.
 - c. Open the `dfc.properties` file in a text editor.
 - d. Modify the following attributes:


```
dfc.globalregistry.repository=global_registry_repository_name
dfc.globalregistry.username=user_login_name
dfc.globalregistry.password=encryped_password_of_user
```

 where `encryped_password_of_user` is the encrypted password you generated in step b.
 - e. Save the `dfc.properties` file.

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: 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.

Dump and load operations are not supported in repositories where Web Publisher is installed.

Code page compatibility issues

Dump files are created by 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 use the same code page as that in use when the dump file was created.

The dump file header does not indicate the session code page in which the dump file was created. If you do not know the session code page in use when a dump file was created, do not load the dump file.

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 a property that contains the name of the file with the dumped information and properties whose values tell Content Server which objects to copy 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 properties are used by Content Server to manage the loading process. It also has two properties 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 information about the properties of these object types, refer to the *Documentum System Object Reference Manual*.

For information about setting up and using the dump operation, refer to [Dumping a repository, page 50](#).

For information about setting up and using the load operations, refer to [Loading a repository, page 59](#).

For information about moving the dump file, refer to [Moving the dump file, page 59](#).

For information about trace messages related to dump and load procedures, refer to [Generating dump and load trace messages, page 64](#).

Execution methods

To perform dump and load operations manually, you can use either IAPI, Docbasic scripts, or DFC. The DFC interfaces are IDfDumpRecord and IDfLoadRecord. This documentation illustrates how to use dump and load by using method statements that would be executed through IAPI. For help with the DFC interfaces, refer to the Javadocs for the interfaces.

Dumping a repository

To avoid dumping unwanted objects, run `dmclean` before dumping a repository.

To dump a repository, create and save a dump record object. The act of saving the object starts the actual dump operation. The properties 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 51](#) or to [Dumping specific objects, page 51](#) 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 54](#). You can set up the dump operation to include the actual content file in the dump file. [Including content, page 55](#) 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 55](#) for information about defining the cache size.

- Whether the operation is restartable

Refer to [Using non-restartable dump, page 56](#) for information about this option.

Dumping objects under retention

The information in this section only applies to dump and loads that are performed by 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` property 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.

Aspects and dump operations

A dump operation does not dump aspects associated with a dumped object. If you have aspects associated with specific instances of an object type, you must create those aspects in the target repository. Similarly, if you have default aspects defined for an object type and you dump instances of that type, the default aspects must be manually created in the target repository. The aspects must be created in the target repository before performing the load operation.

Dumping an entire repository

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

By default, a full repository dump includes content files by reference. To include them directly, refer to [Including content, page 55](#) 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_docbase_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 55](#) for information about the cache.

- The `type`, `predicate`, and `predicate2` properties

Dumping specific objects

To dump only specific objects in a repository, set the `type`, `predicate`, and `predicate2` repeating properties of the dump record object. The `type` property identifies the type of object you want to dump and the `predicate` and `predicate2` properties define a qualification that determines which objects of that type are dumped. The *Documentum System Object Reference Manual* contains a full description of a dump record object's properties.

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 properties of the dump record object.

When dumping a type that 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 property

The type property is a repeating property. 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 property. 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 by 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
l=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, include the following lines in a dump script to 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 properties

You must supply a predicate for each object type you define in the type property. 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 is 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 properties. The qualification is imposed on the object type specified at the corresponding index level in the type property. 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. The *Content Server DQL Reference Manual* contains the description of a valid WHERE clause qualification.

The predicate property accepts a maximum of 255 characters. If the qualification exceeds 255 characters, place the remaining characters in the `predicate2` property 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` property accepts a maximum of 255 characters also.

Important: If you use the `predicate2` property at any index position, you must also set the `predicate2` property at all index positions before the desired position. Content Server does not allow you to skip index positions when setting repeating properties. 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, EMC Centera storage area, or NetApp SnapLock 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 54](#) describes the default behavior and requirements for handling dump files without content. [Including content, page 55](#) 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 file stores, EMC Centera stores, or NetApp SnapLock stores 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 retention 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 retention 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 filepaths 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, set the `include_content` property to T (TRUE) in the dump record object. If the property 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, EMC Centera store, or NetApp SnapLock 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 retention 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.

Compressing content

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` property 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` property 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` property 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 re-creating 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 properties, saves the object, and checks for errors.

If you do not set the `file_name` property to a full path, Content Server assumes the path is relative to the root directory of the server. The filename must be unique within its directory. This means that after a successful load operation that uses the dump file, you must move the dump file to archival storage or destroy it so you can successfully execute the script later.

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

```
iapi source_db -Username -Password < 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.
- `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.
Destruction of 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` property of the dump record object to `full_dochbase_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_dochbase_dump
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 does not 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 a 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
l=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 do not set the `file_name` property to a full path, Content Server assumes the path is relative to the root directory of the server. The filename 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.
- 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:

- Destroy the dump file (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 61.

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 will not 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` property. The `relocate` property is a Boolean property that controls whether the load operation assigns new object IDs to the objects it is loading.

The type and predicate properties 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` property 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 property 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 property 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]
```

- *repository* is the name of the repository into which you are loading the dump file.
- *filename* is the name of the dump file.
- *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.

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, ensure 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` property 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 `engr_store`, which references the `d:\documentum\data\engr (/u04/home/installation_owner/data/engr)` 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.
6. After the load completes successfully, you can destroy the load object:

```
destroy,c,load_object_id
```

Notes:

- Destroying the load object cleans the load object record objects that are generated by the loading process and old state information.
- If you created the dump file by using a script, move the dump file to archival storage or destroy it after you successfully load the file. 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 `setServerTraceLevel` method.

The trace information 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

Creating location and mount point objects

The directories that a Content Server accesses are defined for the server by location objects. 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 an 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 by using Documentum Administrator or a DQL CREATE...OBJECT statement. For help with the procedure using Documentum Administrator, refer to the Documentum Administrator online help. Using DQL allows you to create the object and set its properties in one step.

For example, the following DQL statement creates a location object, sets its properties, 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'
```

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 properties:

- `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` property. If the location object is associated with a storage area, the setting in the property is applied to files stored in the associated storage area. There are three possible values for the `security_type` property. [Table 5, page 65](#), lists the values and the file permissions associated with each.

Table 5. security_type property 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 properties are optional and the server will provide default values if they are not set. Refer to the *Documentum System Object Reference Manual* for a complete listing of the type's properties.

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 by using Documentum Administrator or the DQL `CREATE...OBJECT` statement. 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 properties and save it in one operation.

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

- `object_name`
- `file_system_path`
- `host_name`

Additionally, the three properties that record alias values for the shared (mounted) disk are typically set. Refer to the *Documentum System Object Reference Manual* for a complete list of the mount point object's properties.

Note: A UNIX client can only mount a Windows Content Server disk if the Windows machine has an 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 properties to record the aliases for the directory represented by the mount point object. Each property 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 by 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` by 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 that represents 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 and will appear in the Chooser for that directory.

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 properties 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 68](#) for a sample query.

For a complete list of properties defined for the format object type, refer to the *Documentum System Object Reference Manual*.

The DOS extension property

When a user requests a file without specifying a filename for the working copy of the file, the server names the file for the user. If the `dos_extension` property 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` property 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 258 for more information about DOS extensions.

The `format_class` property

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

- `ftalways`

All renditions in formats whose `format_class` property is set to `ftalways` are indexed. For example, if a document has renditions in Microsoft Word and PDF formats and the `format_class` property 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` property 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` property is set to `ftalways` are also indexed. If a document has more than one rendition whose `format_class` property 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` property 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"
```

To obtain more information about the format definitions in the repository, modify the query to return other or all format property values.

Adding format objects

You can add a format object by using Documentum Administrator or DQL. For instructions on using Documentum Administrator, refer to the Documentum Administrator online help. The *Documentum System Object Reference Manual* lists the properties of format objects.

Using DQL

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

```
CREATE "dm_format" OBJECT
SET property = value {,SET property = 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" = 'txtxt',
SET "mac_type" = 'TEXT',
SET "dos_extension" = 'txt',
SET "description" = 'ASCII text'
```

Rich media formats

If the format you are adding is a rich media format (video, audio, and so forth), there are four properties that you may want to set depending on the format. Set these properties by using DQL or Documentum Administrator. The properties are:

- `richmedia_enabled`
Set to TRUE 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 to enable applications to identify the kind of content represented by the format. Set the property to a string value that identifies the content.
- `default_storage`
Set 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 property to the object ID of the appropriate storage area.
- `filename_modifier`
Set to have Site Caching Services append a modifier to renditions when exporting multiple renditions of the file. Set the property to the modifier you want to append.

Modifying formats

Use Documentum Administrator or DQL 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 property_name = value
set property_name[[index]] = value
append [n]property_name = value
insert property_name[[index]] = value
remove property_name[[index]]
truncate property_name
[un]link 'folder path'
move [to] 'folder path'
```

For example, here is an excerpt of an IDQL session that sets the `dos_extension` property 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 the *Content Server DQL Reference Manual*.

Deleting formats

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

To delete a format, use Documentum Administrator or DQL.

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 the *Content Server DQL Reference Manual*.

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 property in template ACLs
- The acl_name, acl_domain, and owner_name properties in SysObjects
- The performer_name property in dm_activity objects (workflow activity definitions)
- The folder path in IDfSysObject.link and IDfSysObject.unLink methods

For a complete description of how aliases are used, refer to *Documentum Content Server Fundamentals*.

Creating an alias set

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

In DQL, use a CREATE...OBJECT statement. The *Documentum System Object Reference Manual* lists the properties 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

To use Documentum Administrator to change or delete the alias set, 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 from any of the object types listed in CREATE TYPE statement description in the *Content Server DQL Reference Manual*.

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

Using DQL

In DQL, use the CREATE...TYPE statement to create a new object type. The statement's syntax and usage information are found in the *Content Server DQL Reference Manual*.

Be sure to set the `acl_domain` and `acl_name` properties 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` property 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` property 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 Documentum Administrator to modify system-defined object types and user-defined object types.

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 the ALTER TYPE description in 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.

Deleting properties

Properties 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 a property, the corresponding column in the table representing the type may not actually be removed. In such cases, if you later try to add a property to the type that has the same name as the deleted property, 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 a property

You can add properties only to custom object types. You cannot add properties to object types whose names begin with `dm`. To add a property, use the following syntax:

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

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

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

```
property_name datatype [REPEATING]
```

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

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

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

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

The definition can also include data dictionary information for the property. For information about the syntax for defining data dictionary information, refer to CREATE TYPE description the *Content Server DQL Reference Manual*.

Deleting a property

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

```
ALTER TYPE type_name DROP property_name
```

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

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

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

Lengthening a string property

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

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

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

The *property_def* argument identifies both the property to change and its new length. For example, suppose you want to lengthen the `department` property 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 property, the statement looks like this:

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

Deleting a type

You can remove a type from the repository only if:

- The type is a 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.

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 cleanup. Cleaning a repository involves removal of:

- 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 can remove older versions of a document. You can also 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 a repository regularly helps ensure that there is little or no wasted space in your installation.

To clean a repository:

1. Perform a complete backup of the repository.
2. Delete unwanted versions of documents.

Based on your business rules, you can delete only versions created before a certain date or by a certain author. Or, you can 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, 2000:

```
DELETE "dm_document" OBJECTS
WHERE "r_modify_date" < DATE('01/01/2000')
```

Or, the next statement deletes all documents created before January 1, 1999 and have the version label outdated:

```
DELETE "dm_document" OBJECTS
WHERE "r_creation_date" < DATE('01/01/1999')
AND ANY "r_version_label" = 'outdated'
```

- To delete versions from a version tree, use a IDfSysObject.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 the Javadocs 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, 2000. 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('01/01/2000')
```

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 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, 2000:

```
DELETE "dmi_queue_item" OBJECTS
WHERE "dequeued_date" < DATE('01/01/2000')
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 482. Instructions for running the utility manually are found in [Using dmclean](#), page 267.
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

Use the information in this section to help maintain optimal query performance in the repository.

Using Update Statistics

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 519.

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

Setting the `DM_GROUP_LIST_LIMIT` environment variable

When a query is issued, Content Server checks the user's permissions on all returned results to ensure that only those results the user has permissions for are returned to that user. To perform the permission checking, Content Server also checks each group to which the user belongs. If the user belongs to a large number of groups, Content Server may use a subquery to perform the permission checking.

The number of groups that Content Server can check without resorting to a subquery defaults to 250. If the user belongs to more than 250 groups, the server must use a subquery. For example, suppose a user who is a member of 500 groups executes the following query:

```
SELECT r_object_id FROM dm_document WHERE object_name LIKE '%dm%'
```

When this query is translated to SQL, a subquery is needed to perform the ACL checking because the default group list limit for queries is 250. The performance is slower due to the need for the subquery. To avoid the use of the subquery, you can reset the default group list limit by setting the `DM_GROUP_LIST_LIMIT` environment variable.

Note: Setting the variable affects all queries. If you want to change the limit for a single query, or just a particular set of queries, you can use the `GROUP_LIST_LIMIT` DQL hint in the particular query or queries.

To set the variable on Windows, set the variable as a system variable. Use a no-sign integer as the value.

On UNIX platforms, add the following line to the `dm_start_repositoryname` script after the first line of the script:

```
DM_GROUP_LIST_LIMIT=value export DM_GROUP_LIST_LIMIT
```

where *value* is the number to which you wish to reset the limit.

You must restart Content Server after setting the variable.

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 can disable the ability to record the names in the package or wf attachment object by turning on package control.

Two properties 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` property 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 property in the workflow definition determines whether the name is actually recorded in the package object.

The package_control property in the workflow definition (dm_process object) is set to 0 by default. This setting means that package control is not enabled. This allows the server to record package names specified in the addPackage and addAttachment methods in the package, or the wf attachment objects that are generated for the workflow's activities or work items.

[Table 6, page 81](#) illustrates how the property settings interact to determine whether the server can record the names.

Note: 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 6. Interaction of properties that determine 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 r_component_name	Content Server records object names in r_component_name if specified in Addpackage
1 (on)	Content Server records blanks in r_component_name	Content Server records blanks in r_component_name

To enable package control at the repository level, set the wf_package_control_enabled property in the docbase config object to T (TRUE). In doing so, the server cannot record component names in the package object, regardless of the setting in the workflow definition.

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 84
- Internationalization, page 87
- The `dm_start_repositoryname` script (UNIX), page 88
- The `server.ini` file, page 88
- Moving the server executable (UNIX only), page 109
- Changing default operating system permits on directories and files (UNIX only), page 109
- Changing a server's configuration, page 109
- Setting the secure connection mode, page 110
- Restarting a server, page 111
- Starting additional servers, page 112
- Communicating with connection brokers, page 114
- Specifying queue size for incoming connection requests (Windows only), page 118
- Shutting down a server, page 118
- Stopping a session server, page 120
- Server log files, page 121
- Server load balancing and failover, page 121
- Clearing the server common area, page 122
- Adding additional servlets to the Java method server, page 123
- Configuring the workflow agent, page 123
- Recovering automatic activity work items on Content Server failure, page 125
- Managing the JBoss application server, page 125

Overview of servers

Servers are processes that provide client access to the repository. They receive queries from clients in the form of DFC 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 session threads.

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 session servers.

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 properties 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` property tells the server how many concurrent users it can accept.

At startup, a Content Server always reads the CURRENT version of its server config object.

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, Managing Repository 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` property 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.

If there are multiple jobs to execute, the process runs each job, sleeping for 30 seconds between the jobs. By default, the process runs up to three jobs in each polling cycle. You can reset the number of jobs executed in each cycle through an argument on the agent exec command line. 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 these configurable defaults, refer to [Modifying agent exec behavior, page 154](#). That 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*.

JBoss application server

When the first repository is installed and configured in an installation on a host machine, the process also installs a JBoss application server. The application server hosts the Java Method Server and other Java-based processes.

For information about starting and stopping the JBoss application server, refer to [Starting and stopping the JBoss application server, page 126](#).

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
- Spanish
- Italian
- German
- Swedish
- Korean
- Chinese
- Japanese

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 7, page 87](#) lists these parameters and their default settings for each locale.

Table 7. Locale-based configuration settings by host machine locale

Host machine locale	Parameter		
	locale_name	default_client_codepage	server_os_codepage
English	en	ISO-8859-1	ISO-8859-1
French	fr	ISO-8859-1	ISO-8859-1
Italian	it	ISO-8859-1	ISO-8859-1
Spanish	es	ISO-8859-1	ISO-8859-1
German	de	ISO-8859-1	ISO-8859-1
Swedish	sv	ISO-8859-1	ISO-8859-1
Japanese	ja	Shift_JIS	On Windows: Shift_JIS On UNIX: EUC-JP
Korean	ko	EUC-KR	EUC-KR
Chinese	zh	MS936	MS936

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_repositoryname script (UNIX)

Starting a server invokes the dm_start_repositoryname script. The script checks to ensure 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 log 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_repositoryname 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.

Changes to the server.ini file take effect only after the server is stopped and restarted. Refer to [Restarting a server, page 111](#) for instructions on restarting the server.

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

```
documentum -h
```

If you want to add a comment to the file, use a semicolon (;) 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 8, page 89](#), lists the keys in the server startup section. 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 8. Server.ini SERVER_STARTUP section keys

Key	Datatype	Comments
acl_update_threshold	integer	None
check_user_interval	integer	The default is 100.

Key	Datatype	Comments
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
database_password_file	string	None
deferred_update_queue_size	integer	Default value is 1000. Valid range is 256 to 4096. For details, refer to deferred_update_queue_size , page 107.
distinct_query_results	Boolean	None
docbase_id	integer	None
docbase_name	string	None
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.

Key	Datatype	Comments
ip_mode	string	Specifies whether Content Server supports IP addresses in IPv6 format. Valid values are DUALSTACK and V4ONLY. The default value is DUALSTACK, meaning that Content Server accepts both IPv4 and IPv6 addresses.
listener_queue_length	integer	For Windows platforms only. Refer to Specifying queue size for incoming connection requests (Windows only) , page 118, for details.
mail_notification	Boolean	None
max_nqa_string	integer	Defines the maximum length of a non-qualifiable string property, in bytes. The default is 2000. If this is not set, the default is the maximum length allowed by the underlying relational database.
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
max_storage_info_count	integer	Defines the maximum number of storage areas for which Content Server maintains information in shared memory. The default is 100. Valid values are positive integers from 100 to 65535.

Key	Datatype	Comments
method_server_enabled	Boolean	The default is T (TRUE).
method_server_threads	integer	The default is 5.
owner_xpermit_default	string	This has one valid value: acl owner_xpermit_default, page 101 , describes the key's use.
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
server_config_name	string	None
server_login_ticket_version	integer	Controls format of generated login ticket, for backwards compatibility. Refer to Configuring login tickets for backwards compatibility, page 173 , for details.
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 173 for information about this key.

Key	Datatype	Comments
umask	string(4)	This is supported for UNIX platforms only. Refer to Changing default operating system permits on directories and files (UNIX only) , page 109 for information about its use.
update_access_date	Boolean	None
upd_last_chg_time_from_db	Boolean	None
use_estimate_search	Boolean	None
use_group_address	integer	Configures who receives email notifications when an event is queued to a group. For details, refer to use_group_address , page 102.
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 properties in the server config object. Use the server config properties to define additional projection targets, rather than the server.ini. This method enables you to change a target without restarting the server. If the same projection target is defined in both the server config properties and in the server.ini file, the values for the target in the server config properties are used.

The [DOCBROKER_PROJECTION_TARGET] section defines the first projection target. To define additional targets, use [DOCBROKER_PROJECTION_TARGET_n] sections. The *n* can be any integer from 0 to 49. Refer to [Defining connection broker projection targets, page 114](#), for instructions about defining these sections.

[Table 9, page 94](#) lists the keys for these sections.

Table 9. 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 that represents 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 10, page 94](#) lists the keys for a FUNCTION_SPECIFIC_STORAGE section.

Table 10. 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 11, page 95](#) lists the keys for a TYPE_SPECIFIC_STORAGE section.

Table 11. 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 12, page 95](#), lists the keys for the FUNCTION_EXTENT_SIZE section.

Table 12. 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
database_next_ext_small	integer	None
database_next_ext_default	integer	None

[Table 13, page 95](#), lists the keys for the TYPE_EXTENT_SIZE section.

Table 13. 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 set during 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.

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 failover 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 110](#), 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` property 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 do not 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 do not 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 does not 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

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 do not 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.

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. Use the `apply` method or the `EXECUTE` statement to run `LIST_SESSIONS`. The *DQL Reference Manual* contains additional information about the `LIST_SESSIONS` function.

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.

The default for `history_cutoff` is 240 minutes.

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_nqa_string

The `max_nqa_string` key defines the maximum length of a non-qualifiable string property. The value defaults to 2000 if not set. Nonqualifiable properties are stored in the `i_property_bag` property of an object. For more information about nonqualifiable properties and the property bag, refer to *Content Server Fundamentals*.

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. The 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.

owner_xpermit_default

The `owner_xpermit_default` key controls whether object owners have all available default extended permissions for an object or only those explicitly assigned to the owners.

If the key is set to `acl`, object owners have only the extended permissions assigned explicitly. The owners are not granted default extended permissions. If the key is not set, object owners have all extended permissions available to object owners by default.

saveasnew_retain_source_group

The `saveasnew_retain_source_group` key controls which default group is assigned to a new object created by a `IdfSysObject.saveAsNew` method. If the key is set to `T` (TRUE), the new object is assigned to the default group of the original (source) object. If the key is set to `F` (FALSE), the new object is assigned to 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 664. Or, if you set `umask=22`, then the permissions become 755 for directories and 644 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. This is the default setting.
- 1, which directs the server to send email to the group's address (identified in the group's `group_address` property). 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.

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.

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 an `IDfQuery.execute` method with the `readquery` flag set to `FALSE` causes an internal connection request.
- Executing an `apply` method or an `EXECUTE DQL` statement 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. The limit is:

- 1022 for Solaris and AIX platforms
- 2046 for HP-UX
- 1024 for Windows systems

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 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 value is TRUE.

max_storage_info_count

The maximum storage information count key defines the maximum number of storage areas for which Content Server maintains information in shared memory. Valid values are positive integers from 100 to 65535. The value defined by the key does not limit the number of storage areas you can create. For example, if the key is set to 150 and you already have 150 storage areas, you can create additional storage areas, but information for the additional storage areas is not maintained in memory.

In a multiserver environment, ensure that this key is set to the same value for all Content Servers.

The default value is 100.

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 143](#).

method_server_threads

The method_server_threads key defines the maximum number of method server worker processes that 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 property 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 re-create 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 re-create 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.

deferred_update_queue_size

The `deferred_update_queue_size` key controls the size of the deferred object update record queue. The queue is set to 1000 by default when Content Server is installed. The valid range for this parameter is 256 to 4096.

If repository operations generate a large number of deferred object updates, you may want to increase this queue size. If the queue fills up, the deferred updates are performed immediately, which may impact performance of an application.

update_access_date

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

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 username before authenticating the user. Valid settings are:

- upper
- lower
- NULL

The default is NULL, which means 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 username 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 value is T (TRUE).

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_repositoryname` script or modify the command line in the script that references the executable.

The command line references the executable by 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 `664`. Or, if you set `umask=22`, then the permissions become `755` for directories and `644` 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. Navigate to **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.
3. Save the file.
4. Stop and restart the server to make the changes visible.

Modifying the server config object

The server startup 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 to modify the server config object. For instructions on using Documentum Administrator, refer to the Documentum Administrator online help.

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: The server listens on an unsecure port. Connection requests from clients that ask for a secure connection will fail.
- Secure: The server listens only on a secure port. Connection requests from clients that ask for a native connection will fail.
- Native and secure: The 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` property in the server's server config object. You can use Documentum Administrator or DQL to reset the `secure_connect_mode` property. You must restart the server after resetting the `secure_connect_mode` property.

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 you 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. Navigate to **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_repositoryname` script that references the server to start.

Starting additional servers

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 property of the server.ini file invoked for the server.



Caution: 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 the *Distributed Configuration Guide* for those instructions.

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. 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 instructions in this section 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. Content 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 the checkpoints is configurable. Refer to [Setting the checkpoint interval, page 116](#) 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 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.

Definitions in the server config object

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

- projection_targets

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

- projection_ports

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

- projection_proxval

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

- projection_enable

The projection_enable property 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 to the connection broker.

- projection_notes

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

The values at the same index positions across the properties 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 properties 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 properties. 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_1]
...
```

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` property in the server's server config object. You can set the checkpoint interval using Documentum Administrator.

Use Documentum Administrator 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.

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` property 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).

Use Documentum Administrator 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.

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 when requesting a connection.

The proximity values should 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 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.

Specifying queue size for incoming connection requests (Windows only)

Content Server creates a socket listener for incoming connection requests with a maximum backlog set to 200 by default. On Windows platforms, you can reset that maximum if needed. To do so, set the `listener_queue_length` key in the `server.ini` file. Set the key to a positive integer value. Content Server passes the specified value to the Windows Sockets call `listen()`.

Shutting down a server

On Windows platforms, you can use Documentum Content Server Manager, the Windows user interface, or an `IDfSession.shutdown` method to shut down 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 an `IDfSession.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 shutdown 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. Navigate to **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 `IDfSession.shutdown` method provides additional options. The method can:

- Shut down the server immediately, without waiting for currently connected users to finish.
- Shut down the server after all sessions finish their current transaction.
- Direct the connection broker to delete its information about the server.

These options are implemented by two arguments to the shutdown method: `immediate` and `deleteEntry`.

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)`.

When you shut down a server by 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 `deleteEntry` flag to `T (TRUE)`.

Both of these arguments are `FALSE` by default.

Stopping a session server

Use a 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. The *DQL Reference Manual* contains more information about the `LIST_SESSIONS` and `SHOW_SESSIONS` administration methods.

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.

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 property in the server config object. The location object is created and the property 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 in 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 the 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 additional servers are 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 reestablished.
- 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 reused.

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. For a proper failover, ensure that Content Servers project to the appropriate connection brokers and with appropriate proximity values.

Clearing the server common area

Use the procedures in this section to remove files that are in the server common area.

To remove all files from the server common area:

1. Stop Content Server.
2. Do one of the following:
 - On Windows, edit the Content Server service to add `-oclean` to the end of the command line.
 - On UNIX, add the `-oclean` argument in the `dm_start_repositoryname` command line.
3. 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. Use the following procedure to implement such a servlet.

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` property and the servlet's URI to the `app_server_uri` property.

Use Documentum Administrator to modify the server config object. The Documentum Administrator online help provides instructions on how to modify the server config object.

4. Select **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 124](#), 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` property 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` property in the server config object. The value is interpreted in seconds. Use 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` property in the server config object to 0. Use Documentum Administrator to set the property. 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 [Starting and stopping tracing from the startup command line, page 532](#). For instructions on using `SET_OPTIONS`, refer to the *Content Server DQL Reference Manual*.

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` property to the `r_object_id` 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 property. The method resets the `a_wq_name` property 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.

Managing the JBoss application server

This section provides instructions and information for administration of the JBoss application server on a Content Server installation.

Location of binaries

The JBoss binary is bundled with your installation suite. The application server is installed into the `jboss<version>` subdirectory of the Content Server installation directory. The default directory, referred to as the JBoss root directory, is:

- On Windows:
`C:\documentum\jboss<version>`
- On UNIX:
`$DOCUMENTUM_SHARED\jboss<version>`

A JBoss server instance is created in `<JBossRoot>\server\<instance name>`. For example, `C:\documentum\jboss4.2.0\server\DctmServer_MethodServer`.

Starting and stopping the JBoss application server

When you first install the Content Server installation, the JBoss application server is started automatically. However, starting and stopping a Content Server in the installation does not automatically start or stop the JBoss application server.

The application server hosts one or more Java applications or processes. In a Content Server installation, the method server hosts, for example, the Java Method Server, ACS server, and the DmMail application.

To start the application server:

1. On Windows:
 - a. Navigate to `<JBossRoot>\server`.
 - b. Execute `startMethodServer.cmd`.
 - c. Alternatively, start the Windows service for the server instance.
For the Java method server, and the server instance hosting the Java method server, the service name is Documentum Java Method Server. Starting that service is equivalent to executing `startMethodServer.cmd`.
2. On UNIX:
 - a. Navigate to `<JBossRoot>\server`.
 - b. Execute `start MethodServer.sh`.

Use the following procedure to stop the application server. Stopping the server stops all applications running within it.

To stop an instance server:

1. On Windows:
 - a. Navigate to <JBossRoot>\server.
 - b. Execute stopMethodServer.cmd.
 - c. Alternatively, stop the Windows service for the server instance.
For the Java method server, and the server instance hosting the Java method server, the service name is Documentum Java Method Server. Stopping that service is equivalent to executing stopMethodServer.cmd.
2. On UNIX:
 - a. Navigate to <JBossRoot>\server.
 - b. Execute stopMethodServer.sh.

Methods and Jobs

This chapter describes how to create and execute methods and jobs. The chapter includes the following topics:

- [Introducing methods, page 129](#)
- [Execution agents, page 130](#)
- [Choosing the execution agent, page 132](#)
- [Tracing options for methods, page 133](#)
- [Defining the java.ini file \(UNIX only\), page 134](#)
- [Enabling the dmbasic method server, page 135](#)
- [Configuring the worker threads in the dmbasic method server, page 136](#)
- [Implementing a method, page 136](#)
- [Executing a method on demand, page 145](#)
- [Creating jobs and job sequences, page 145](#)
- [Managing jobs, page 153](#)

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 properties that identify the executable and define command line arguments, and the execution parameters.

Methods are executed by issuing a DO_METHOD administration method or by 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 145](#).

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 the executable in %DOCUMENTUM%\dba\java_methods (\$DOCUMENTUM/dba/java_methods) or deploy the executable as a BOF module.

Deploying a Java-based method as a BOF module allows the method and the executable to be packaged and deployed in a DAR archive. It also allows you to take advantage of BOF sandboxing, and to make modifications to the module without requiring you to restart the Java method server.

You can execute a method by using the dmbasic method server, the Java method server, or Documentum Content Server. [Execution agents, page 130](#), describes each of these agents and [Choosing the execution agent, page 132](#) 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
- Java method server
- Documentum Content Server

The dmbasic method server

The dmbasic method server is a separate process that is installed with Documentum 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 135](#).

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 and executes 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 100. 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 136](#).

The method server uses connection pooling. This is true regardless of whether connection pooling is enabled for the Documentum Server.

Java method server

The Java method server is a third-party application server, installed as a component of Content Server installation. Additionally, Documentum provides a servlet to execute methods called by DO_METHOD. During installation, the `app_server_name` and `app_server_uri` properties are set to configure use of the Java method server. The `app_server_name` property in the server config object identifies the application servers recognized by Content Server. The value “do_method” in that property represents the Java method server. The value in the corresponding index position in the `app_server_uri` property contains the host and port number for the server.

Refer to *Documentum Content Server Installation Guide* for more information about how to configure the Java method server and the servlet.

The Java method server resides on the same host as Content Server. The Java Method Server is started automatically by the Content Server installation process. However, if you start or stop Content Server after installation, you must start or stop the Java Method Server manually, if needed. Starting and stopping Content Server does not start or stop the Java Method Server. [Starting and stopping the JBoss application server, page 126](#) provides instructions to start and stop the Java method server.

The programs launched with DO_METHODs that are sent to the Java method server must be Java methods stored in `%DOCUMENTUM%\dba\java_methods` (`$DOCUMENTUM/dba/java_methods`) or deployed as a BOF module. 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 to execute Java methods that are not associated with a method object. To do this, use an HTTP_POST administration method to send the request to the Java method server. For details about the administration method, refer to 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` property to TRUE to direct the execution to the method server or Java method server.

Choosing the execution agent

The execution agent that 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 dmbasic 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 or deployed as BOF modules.

To execute Java or DFC calls that use the dmbasic 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` property to `TRUE` and setting the remaining properties appropriately. Refer to [Creating a method object, page 143](#) for instructions.

Performance considerations

Use of either the method server or the Java method server 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.

Use of 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. Use of connection pooling removes the need to open new repository and RDBMS sessions for subsequent method executions.

Use of 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.
- Logging in without passwords.

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. 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 does not match the IP address of a Content Server host, the request fails.

The second issue occurs because the 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. A process running on the Content Server host as installation owner can log into the repository as the installation owner without 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 a user 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 username and a login ticket to the method in the arguments. The method can use this information 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 property 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 property setting. The trace information, up to 2047 characters, includes the command executed to invoke the method and messages that indicate 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 dmbasic method server or Content Server, set the trace_method_server server flag. Set this flag by 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 timestamp 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>
```

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, 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 dcm.jar file. Paths specified in classpath are separated by semicolons (;). For example:

```
java_classpath=<$dm_home>/classes/com/documentum/dcm.jar:/usr/java/bin/jre
```

java_alias_file

This specifies the full path to the dcm.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. Do one of the following:
 - To enable the method server, set the method_server_enabled key to T.
 - To disable the method server, set the method_server_enabled key to F.
3. Save the server.ini file.
4. Restart Content Server.

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 chosen program language and the execution agent chosen 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 136.

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 141.

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.

Limitation on argument length for Docbasic

The total length of the arguments passed in a Docbasic method cannot exceed 4095 characters.

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 guidelines in this section.

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 for Linux and Solaris
- LIBPATH for AIX
- SHLIB_PATH for HP-UX

Also located in \$DM_HOME/bin are two scripts to help set up user environments:

- dm_set_server_env_sh
- dm_set_server_env.csh

dmbasic executables (UNIX only)

Content Server ships with three versions of the dmbasic executable: dmbasic, dmbasic_shr, and dmbasic_noshr.

The standard interface is dmbasic and uses the shared library version of the DMCL, which dmbasic links to dynamically.

The dmbasic_shr and dmbasic_noshr executables are copies of dmbasic. Although dmbasic_shr is the same as dmbasic, it is provided for backwards compatibility. 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 -eentrypoint -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 properties for Content Server execution

To execute the method using Content Server, the use_method_server property must be set to F (FALSE). The remaining properties are set as needed. For example, if the program is written in Docbasic, then method_type and method_verb are set to dmbasic. The *Documentum System Object Reference Manual* lists the properties 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 143.

Setting method object properties for dmbasic method server execution

To execute a method using the dmbasic method server, set the method properties as follows:

- Set `method_type` property to `dmbasic`.
- Set `use_method_server` property to `T (TRUE)`.
- Set `run_as_server` property to `T (TRUE)`.

For instructions on how to create a method object, refer to [Creating a method object](#), page 143.

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: 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.

Guidelines for a Java method to be deployed as a BOF module

If you intend to the deploy the method as a BOF module, observe these guidelines:

- The method must implement the `IDfMethod` interface.
- The module must be a simple module, one which implements the `IDfModule` interface.
- The `method_verb` property of the `dm_method` object must be set to the module's name, not the class name.

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 application server) 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

The storage location of the executable methods to be executed by the Java method server depends on whether or not the method is deployed as a BOF module.

If the method is not deployed as a BOF module, the method must be stored in %DOCUMENTUM%\dba\java_methods (\$DOCUMENTUM/dba/java_methods).

If the method is deployed as a BOF module, it is stored automatically in the default location for the module.

Setting method object properties for Java method server execution

Methods that you intend to execute on an Java method server must have the following property values:

- The method_type property must be set to Java.
- The use_method_server property must be set to T (TRUE).
- The method_verb property is set to identify the method:
 - For methods stored in the java_methods directory, method_verb must be set to a fully qualified class name of a Java implementation class. For example: com.documentum.services.myAutoMethod
 - For methods deployed as a BOF module, method_verb must be set to the module name.
- The run_as_server property 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 143.

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 144](#) describes how to use DQL to create a method object.

The values you set in many method properties 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 136](#) and [Creating a method to be executed by the Java method server, page 141](#). For a list of the properties of a method object, refer to the *Documentum System 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 144](#) for information about setting the `success_return_codes` and `success_status` properties.

Using DQL

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

```
CREATE dm_method OBJECT
SET property_name[[index]]=value
[,SETFILE filepath CONTENT_FORMAT=format_name]
{,SETFILE filepath PAGE_NO=page_number}
```

Refer to the *Documentum System Object Reference Manual* for information about the properties of a method object.

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. The *Content Server DQL Reference Manual* has complete information about using the SETFILE clause.

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` property or the `success_status` property or both. The `dm_run_dependent_jobs` method, which invokes sequenced jobs, uses one or both of those properties to determine whether an invoked job completed successfully.

The `dm_run_dependent_jobs` method compares the values in the `success_return_codes` property and the `success_status` property 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 property. 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` property. The property is set to the value returned by the job's method. The job's `a_current_status` property 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 to issue a DO_METHOD call to execute a method. The *Content Server DQL Reference Manual* describes how to use DO_METHOD and the EXECUTE statement.

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 146](#)
- [Introducing job sequences, page 146](#)
- [The agent exec process, page 148](#)
- [Creating a job, page 149](#)
- [Creating a job sequence, page 150](#)
- [Scheduling jobs, page 150](#)
- [Passing arguments, page 151](#)
- [The run_now property, page 152](#)

Introducing jobs

Jobs automate method execution. Jobs are stored in the repository as `dm_job` objects. The properties 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 11, Administration Tools](#), for a list and description of the tools.)

Use Documentum Administrator to create jobs. You must have Sysadmin or Superuser privileges to create a job. Refer to [Creating a job, page 149](#), 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 5.3 or later repositories.

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 properties 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` property 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` property 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` property. (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 158](#), contains instructions on handling this situation.

Determining success for invoked jobs

The controlling method, `dm_run_dependent_jobs`, uses the values in the method properties 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 144.

The repository connection file

The repository connection file 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` properties 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 155.

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` property is less than the current date and time or if its `run_now` property 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 154](#).

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 151](#), 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 143](#), 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 144](#), 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.

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` properties. You will probably also want to set the optional properties that set the job's name

and expiration date, pass argument values to the method, and activate the job. (The *Documentum System Object Reference Manual* has a description of job object properties. The scheduling properties are described in detail in [Scheduling jobs, page 150.](#))

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 properties: `success_return_codes` and `success_status` properties. (For information about setting these properties, refer to [Defining success return codes and success status, page 144.](#))

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 148](#), 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 property 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 properties 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 properties:

- `start_date`
- `a_next_invocation`

The starting date is the earliest date at which the job can be executed. The `a_next_invocation` property 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` property 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 properties:

- `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 Repeat (`run_mode`) is set to Weeks and Frequency (`run_interval`) is set to 1, the job repeats every week. If Repeat is set to Weeks and Frequency is set to 3, the job repeats every three weeks.

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` property. The end date for the job is recorded in the `expiration_date` property. When either value is met, the agent exec process ignores the job and no longer schedules it for execution.

Passing arguments

Jobs have a property 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 14, page 152](#), to the method.

Table 14. Standard arguments passed to jobs

Argument	Datatype	Value	Description
-docbase	string(64)	<i>repository_name</i>	Identifies the repository
-user_name	string(64)	<i>user_name</i>	The user executing the job
-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> property of the job object.

If the server passes the standard arguments, it does not pass to the method any arguments defined in the job's `method_arguments` property. The method must use the job ID passed as a standard argument to obtain the argument values defined in the job's `method_arguments` property.

If `pass_standard_arguments` is set to F, the server passes the method the argument values found in the job's `method_arguments` property, but does not pass the standard arguments.

The default for `pass_standard_arguments` is F.

Note: For all the system administration jobs installed with Content Server, `pass_standard_arguments` is set to T. Do not reset that value to F.

The `run_now` property

The `run_now` property 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 property 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 selecting 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 153](#)
- [Disabling all jobs, page 153](#)
- [Modifying agent exec behavior, page 154](#)
- [Creating and maintaining a repository connection file for job sequences, page 155](#)
- [Recovering from a job sequence failure, page 158](#)
- [Interpreting a job sequence status report, page 158](#)
- [Executing `dm_run_dependent_jobs` independently, page 159](#)

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` property. Use Documentum Administrator to change the status.

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` property 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 does not 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 156](#), 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` property 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_property` 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 15, page 156](#), lists the arguments accepted by the method or on the command line.

Table 15. `dcf_edit` utility arguments

Argument	Description
<code>-server_config</code> <code>server_config_name</code>	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> <code>domain_name</code>	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>	<p>Identifies the user as whom to connect.</p> <p>This argument is optional for the Add and Remove operations, but is invalid for Backup operations.</p>
<code>-password <i>user_password</i></code>	<p>The clear-text password for the user identified in <code>-login_user</code>.</p> <p>This argument is required for the Add operation, but is invalid for Remove and Backup operations.</p>
<code>-f <i>repository_filepath</i></code>	<p>Path to the repository connection file.</p> <p>This parameter is a required parameter for all operations.</p>
<code>-operation <i>operation_name</i></code>	<p>Identifies the operation being performed. Include only one operation on each execution of the utility. Valid operation names are:</p> <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 158](#), 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 159](#), 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 16, page 158](#), lists the events recorded in the report and describes their format.

Table 16. Entries in a job sequence statusreport

Event	Entry Format
Start	<i>Date Time start-sequence=job_sequence_name</i>
Start Job	<i>Date Time start job - doctype=repository_name job=job_id attempt=1 2 3</i>

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 17, page 159, lists the arguments accepted by the method. Some arguments are required and some are optional.

Table 17. dm_run_dependent_jobs arguments

Argument	Required?	Description
-docbase <i>repository_name</i>	Yes	Identifies the repository that contains the job sequence.
-user_name <i>user_name</i>	Yes	Identifies the user executing the job sequence.
-job_sequence <i>sequence_name</i>	Yes	Identifies the job sequence to be executed.
-method_trace_level <i>trace_level</i>	No	Defines a trace level for the dm_run_dependent_jobs 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.

Managing Repository Sessions

This chapter contains instructions for common administration tasks for repository sessions. For a detailed description of sessions and how sessions behave, refer to *Documentum Content Server Fundamentals*. The topics in this chapter are:

- Terminology, page 162
- The `dfc.properties` file, page 162
- Defining connection brokers for connection requests, page 163
- Failover and load balancing, page 164
- Requesting a specific server connection, page 165
- Turning off trusted login, page 166
- Defining the secure connection default for connection requests, page 166
- Configuring the number of connection attempts and the retry interval, page 167
- Specifying the maximum number of sessions, page 167
- Limiting which clients can access a repository, page 168
- Configuring privileged DFC use, page 168
- Configuring connection pooling, page 172
- Configuring login tickets, page 173
- Changing a session's configuration, page 174
- Changing the assigned default operating system permissions (UNIX only), page 175
- Defining short date formats, page 175
- Changing the client local area directory location, page 176
- Setting disk space limits for the client local area , page 176
- Removing content from client local areas, page 176
- Managing persistent client caches, page 177

Terminology

This chapter uses the following terms in the manner described:

- 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.

The dfc.properties file

Most of the configuration parameters that configure how repository sessions are handled by DFC are contained in the `dfc.properties` file. For example, the file contains keys that identify which connection brokers are used to obtain a session, specify how many sessions are allowed for one user, and enable persistent client caching. Many keys have default values, but some must be explicitly set by an administrator or application.

Some of the keys in the file are dynamic; that is, changing them affects any open sessions as soon as DFC notes the change. Other keys are not dynamic. Changing non-dynamic keys affects only future sessions. Current sessions are not affected. Dynamic changes are effective the next time DFC checks the file for changes. By default, the file is checked for changes every 30 seconds. This interval is configurable. It is set in the `dfc.config.check_interval` key in the `dfc.properties` file.

The `dfc.properties` keys are described in file called `dfcfull.properties`, which is installed when you install DFC. The information in that file describes each key and its default and valid values, as applicable.

Key format

Each key in the `dfc.properties` file has the following format:

category.name=value

The *category* identifies the functionality, feature, or facility that is configured or controlled by the key. Key categories are in the format:

dfc.category_name

where *category_name* can be one or more words separated by periods. For example, the following are key categories: `dfc.data`, `dfc.content`, `dfc.tracing`, and `dfc.search.ecs`.

The *name* begins after the last period in the key. If the name contains multiple words, they are separated by underscores. For example, the following are key names: `dir`, `max_error_retries`, and `enable`.

When you add a key to the `dfc.properties` file, you must specify both the category and the name, in addition to a value. For example:

```
dfc.data.dir= value
dfc.tracing.enable= value
dfc.search.ecs.broker_count = value
```

Setting `dfc.properties` entries

For EMC Documentum web-based products, the `dfc.properties` file is packaged in the application's WAR file. The file is modified using a JMX-based resource agent through Documentum Administrator.

For desktop applications and Content Server, the file is installed on the file system during product installation. On Windows hosts, it is found in the `C:\Documentum\Config` directory. On UNIX hosts, it is found in `$DOCUMENTUM_SHARED/config`. The file may be edited using any text editor.

The `dfc.properties` file is a standard Java properties file. If the value you are specifying for a key has a special character, you can use a backslash (`\`) to escape the character. In directory path specifications, use a forward slash (`/`) to delimit directories in the path.

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 `dfc.properties` file. There must be at least one connection broker identified in the `dfc.properties` file. If multiple connection brokers are defined in the file, the system will use the additional connection brokers if needed for backup or failover. [Failover and load balancing, page 164](#), for more information.

Specifying connection brokers

The `dfc.properties` file used by a session must specify at least one connection broker. Including additional entries for backup is optional; however, providing additional entries for backup can help ensure that all connection requests are successful.

To specify a connection broker in the `dfc.properties` file, use the following keys:

```
dfc.docbroker.host[n]=host_name|IP_address    #required
dfc.docbroker.port[n]=port_number           #optional
```

The $[n]$ is a numeric index, where n is an integer starting with 1. All keys for a particular connection broker must have the same numeric index. If there are entries for multiple connection brokers, DFC contacts the connection brokers in ascending order based on the numeric index by default.

The *host_name* is the name of the machine on which the connection broker resides. The *IP_address* is the IP address of the machine.

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.

The following example defines two connection brokers for the client:

```
dfc.docbroker.host[1]=bigdog
dfc.docbroker.port[1]=1489

dfc.docbroker.host[2]=littlecat
dfc.docbroker.port[2]=1489
```

In this example, lapdog is defined as the connection broker. Because lapdog is not using the default port number, the port number is also specified in the file:

```
dfc.docbroker.host[1]=lapdog
dfc.docbroker.port[1]=1491
```

Only the host specification is required. The other related keys are optional.

If you add, change, or delete a connection broker's keys, 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 are controlled by settings in the `dfc.properties` file. The `dfc.docbroker.auto_request_forward` key controls failover. The `dfc.docbroker_search_order` key controls load balancing. For information about setting up default failover, refer to [Failover for connection brokers, page 191](#). For information about setting up load balancing, refer to [Load balancing for connection brokers, page 191](#).

Requesting a specific server connection

If you have started multiple servers for a repository, you can be as specific as you like about which server you want to use for your repository connection. You can:

- Let DFC choose the server for you
- 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, include the name of the server in the session request.

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 an `IDfDocbrokerClient.getServerMap` or `getServerMapFromSpecificDocbroker` method. Refer to the Javadocs for information about these methods.

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 repository in the request as `repository@host_name` in the connection request.

If there is no server on the specified host that accesses the specified repository, the connection fails. To ensure that you request a host that has at least one server that accesses the requested repository, use an `IDfDocbrokerClient.getServerMap` or `getServerMapFromSpecificDocbroker` method. Refer to the Javadocs for information about these methods.

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 repository in the connection request as `repository.server_name@host_name`.

If the specified server does not reside on the specified host or if the server does not access the specified repository, the connection fails.

Turning off trusted login

By default, applications running on the Content Server host are allowed to make repository connections as the installation owner without presenting a password. This is called a trusted login. If an application, such as Documentum Administrator, that has an explicit login dialog box is installed on a Content Server host, a user is able to login as the installation owner without a password using a trusted login.

If you do not wish to allow trusted logins through such applications, you can set the following preference in the `dfc.properties` file to F:

```
dfc.session.allow_trusted_login=F
```

When that key is false, a user is required to always provide a password, even when logging in as the installation owner.

Defining the secure connection default for connection requests

All connections between Content Server and a client application are either secure or native (unsecure) connections. Which option is used for a particular connection depends on how the server is configured and what sort of connection is requested by the client application when it attempts to obtain a session. If the request does not specify what type of connection is requested, the connection type specified in the `dfc.properties` file, in the `dfc.session.secure_connect_default` key, is used.

There are four possible settings for this key:

- native
native means that you want only a native connection. If DFC cannot establish a native connection, the connection attempt fails.
- secure
secure means that you want a secure connection only. If DFC cannot establish a secure connection, the connection attempt fails.

- `try_secure_first`
`try_secure_first` means that you prefer a secure connection, but will accept a native (unsecure) connection. DFC attempts to establish a secure connection first. If it cannot, it tries to establish a native connection. If that also fails, the connection attempt fails.
- `try_native_first`
`try_native_first` means that you prefer a native connection, but will accept a secure connection. DFC attempts to establish a native connection first. If it cannot, it tries to establish a secure connection. If that also fails, the connection attempt fails.

The default setting for the key is `try_native_first`.

Specifying a connection type in the application overrides the default setting in the `secure_connect_default` key. For information on how to specify it in an application, refer to the *DFC Development Guide*. The Javadocs for `IDfLoginInfo.setSecurityMode` describe the interaction between the connection type requested by the client and the server default setting. [Setting the secure connection mode, page 110](#), describes how to set the server default.

Configuring the number of connection attempts and the retry interval

Two keys in the `dfc.properties` file control DFC behavior regarding the number of connection attempts and the interval between the attempts when an application issues a request for a session. One key controls how many attempts are made to obtain a session by DFC in response to the request and one controls how long DFC waits between each attempt.

To set the number of attempts, set the `dfc.session.max_connect_retries` key. This key is set to 2 by default.

To set the interval between attempts, set the `dfc.session.connect_retry_interval` key. The key is set to 0 by default. Zero means that the retries are issued immediately, with no waiting time.

Specifying the maximum number of sessions

The `dfc.session.max_count` key in `dfc.properties` defines the maximum number of sessions that can be opened for a particular user or application. By default, the key is set to 10. The format of the key is:

```
dfc.max_count=n
```

where n is an integer value greater than zero.

Note: The maximum number of sessions possible for a client process running on a UNIX platform is also limited by the descriptors set in the UNIX kernel. If a client process requires more sessions than this limit permits, the UNIX system administrator must modify the UNIX kernel.

Limiting which clients can access a repository

You can enforce which EMC Documentum client versions may access a particular repository.

Two properties of the docbase config object control client access: the `oldest_client_version` property and the `check_client_version` property. The `oldest_client_version` property 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 6.0 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`. However, note that the older clients can bypass the restrictions imposed by Retention Policy Services.

Configuring privileged DFC use

Privileged DFC is a standard security feature that enables DFC instances to invoke a privileged BOF module to perform operations not otherwise permitted to the user as whom the application is running. The procedures in this section configure the use of this feature.

For a detailed description of this feature, refer to *Content Server Fundamentals*.

Creating client rights objects

Client rights objects define the roles a particular DFC may use within a repository. After installing a DFC instance, use Documentum Administrator to create a client rights object in each repository in which the DFC will use a privileged role or invoke a privileged module. When you create a client rights object, Content Server also copies the

specified public key certificate from the global registry to the repository to which you are connected. The copy is stored in a public key certificate object.

For instructions on using Documentum Administrator, refer to the online help or to the *Documentum Administrator User Guide*.

Configuring a repository to accept only authenticated DFC instances

An authenticated DFC instance is a DFC instance that has a client rights object and an associated public key certificate object in the repository. By default, a Content Server accepts a connection request sent by any DFC instance. However, you can configure a repository to decline all connection requests except those coming from authenticated DFC clients. To do so, set the `approved_clients_only` property in the `doabase` config object.

Setting this to T directs a repository's Content Server to accept connection requests only from DFC instances that have a client rights and an associated public key certificate in the repository. If the property is F (false), the repository's Content Server may accept connection requests from any DFC instance.

Use Documentum Administrator to set this property.

Enabling privileged DFC

In most installations, no manual operations are required to enable and use privileged DFC. There are two requirements to use privileged DFC, both of which are met automatically in most environments. These requirements are:

- The `dfc.privilege.enable` key in the `dfc.properties` file must be set to T.
This is the default setting for the key.
- The Java security policy file or files must be modified to give the `dfc.jar` file all permissions

In installations in which the JRE environment is owned by the Documentum product, the necessary modifications are done during product installation. For example, the modifications are performed automatically when you install Content Server, Documentum Administrator, or a WDK-based application on the application server.

However, in some environments, the application does not own the JRE environment. In those instances, you must manually modify the file. For example, if you install IAPI or IDQL and a DFC instance on a workstation, you must manually modify the Java security policy file or files. For instructions on modifying the file, refer to [Modifying the Java security policy](#), page 170.

Disabling privileged DFC

To disable a DFC instance's use of privileged DFC, set the `dfc.privilege.enable` key in the `dfc.properties` file to `false`.

To disable use of privileged roles and modules:

1. Add the following key to the `dfc.properties` file used by the DFC instance:

```
dfc.privilege.enable=F
```

2. Reinitialize the DFC instance.

Modifying the Java security policy

The implementation of the privileged DFC feature assumes that the `dfc.jar` file has all possible privileges in the JRE environment. To make that assumption true, the Java security policy files are modified automatically during installation of Documentum products that also install a DFC instance. However, if you are running a DFC instance in an environment in which you do not own the JRE environment, you must make the necessary file modifications manually if you want the DFC to run as a privileged DFC. For example, a workstation on which IAPI is installed may require the manual modifications.

The security policy files are referenced in the `$JREHOME/lib/security/java.security` file, which configures the JRE startup environment for finding the policy files. The entries for the default policy files are:

```
policy.url.1=file:${java.home}/lib/security/java.policy
policy.url.2=file:${user.home}/.java.policy
```

The default policy files are:

```
${java.home}/lib/security/java.policy
${user.home}/.java.policy
```

The file found in `${java.home}/lib/security` is a global file. Modifying that file affects all instances of the JVM in `$JREHOME/` on the host. The file found in `${user.home}` is specific to the user who starts the JVM. The set of permissions given to a program is the union of the permissions contained in the global and user-specific policy files.

These files are text files. To edit the files, use the `policytool` utility or a text editor. For each instance of a `dfc.jar` file that may be used for privileged DFC, the policy file must contain an entry like this:

```
grant codeBase "file:/C:/fully_qualified_path_to_dfc.jar"
{permission com.documentum.fc.client.impl.bof.security.RolePermission
  "*", "propagate";
};
```

where *fully_qualified_path_to_dfc.jar* is the fully qualified path to the *dfc.jar* instance. If needed, consult the JDK documentation for help with the syntax. A syntax guide for the entries in the

Managing the keystore file

The keystore file stores a DFC instance's PKI credentials. By default, this file is named *dfc.keystore* file and it is stored in the same directory as the *dfc.properties* file. You can change the file's name and location.

The credentials in the file are accessed using the *keytool* utility provided with the Java SDK.

Changing the location and name of the keystore file

By default, the PKI credentials for a DFC instance are stored in a file named *dfc.keystore* that is stored in the same directory in which the *dfc.properties* file is stored. You can change its location, and optionally its name, by setting the following key in the *dfc.properties* file:

```
dfc.security.keystore.file
```

The key must be set to a fully qualified file name.

Changing the passwords used with keytool

PKI credentials are composed of two parts: a public key and a private key. When you access a set of credentials using the *keytool* utility, you must provide a password to access either key. There are two different passwords: one for the public key and one for the private key. Default passwords are provided for accessing each key.

The default password for the public key is "dfc". To change that password, set the following key in the *dfc.properties* file:

```
dfc.security.keystore.password=newPassword
```

where *newPassword* is the password you wish to use to access the public key in the credentials.

The default password for the private key is "!!dfc!!". To change that password, set the following key in the *dfc.properties* file:

```
dfc.security.keystore.privatekey.password=newPassword
```

where *newPassword* is the password you wish to use to access the private key in the credentials.

Configuring a shared keystore file

It is possible to configure multiple DFC instances to share the same keystore file. To do so, you must set the name and location of the keystore file to the same name and location for each keystore. You must also configure an alias for each DFC instance to disambiguate the identities of the instances within the file. By default, the alias for each DFC instance is `dfc`. If multiple instances are sharing the same keystore, you must define unique aliases for each.

To define an alias for a DFC instance, set the following key in the `dfc.properties` file used by the DFC instance:

```
dfc.name=alias
```

where *alias* is the alias name you wish to assign the instance. The alias must be unique among those DFC instances sharing the same keystore file.

Configuring connection pooling

Connection pooling allows applications to reuse sessions. When a session is released or disconnected, 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. DFC uses connection pooling by default. However, explicitly enabling connection pooling is recommended because the most efficient resource management is provided when pooling is explicitly enabled. For complete details of how connection pooling is implemented, refer to *Content Server Fundamentals*.

Enabling connection pooling

To enable connection pooling, set the `dfc.session.pool.enable` key in the `dfc.properties` file to `TRUE`.

Setting this key affects all sessions that are opened after the key is set. Sessions that are open when the key is set are not affected.

Configuring login tickets

Login tickets are values that applications can use in place of a user's password when establishing a repository session or authenticating a user. Login tickets are obtained by executing one of several methods in the IDfSession interface. For a complete description of login tickets, their use and implementation, refer to *Content Server Fundamentals*.

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. You can also configure whether Content Server generates login tickets for backwards compatibility.

Setting ticket validity period

To define the length of time for which a login ticket is valid, set the `login_ticket_timeout` property in the server config object. By default, the property is set to 5 minutes.

After you set the property, 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.

Configuring login tickets for backwards compatibility

In environments that have Content Servers at mixed version levels, you may wish to generate login tickets that are compatible with the version level of the server that will receive the login ticket. For example, if you have multiple Content Servers on one repository, there may be a period where one or more of the servers are not yet upgraded to version 6. Or if you are using global login tickets, the Content Servers in the different repositories may be at different version levels.

A Version 6 Content Server can accept a login ticket generated from any Content Server, regardless of the server's version level. However, a pre-5.3 SP4 Content Server cannot automatically accept login tickets generated by a Content Server at version 5.3 SP4 or higher. And, a 5.3 SP4 or 5.3 SP5 Content Server cannot automatically accept login tickets generated by a Version 6 Content Server.

To ensure that login tickets generated by a Content Server are backwards compatible, set the `server_login_ticket_version` key in the `server.ini` file:

- Set the key to 1 to generate login tickets acceptable to a pre-5.3 SP4 Content Server
This setting is only valid on Version 6, 5.3 SP4, and 5.3 SP5 Content Servers.
- Set the key to 2 to generate login tickets acceptable to a 5.3 SP4 or 5.3 SP5 Content Server
This setting is only valid on Version 6 Content Servers.
- Set the key to 3 to generate login tickets acceptable to Version 6 Content Servers.
This setting is only valid on Version 6 Content Servers.

The key default to 3 if not set.

Changing a session's configuration

A session's configuration, or operating behavior, is determined by the values of the properties in its session config and connection config object and in the client config object. The session and connection config objects are constructed when the session is started, using the properties of the client config object and the server's server config object. The values in the client config object are reflections of the values in the client's `dfc.properties` file.

You can change the behavior of a session by changing any of the configuration objects or the `dfc.properties` file. Changing the session config or connection config objects affects only the current session. Changing the `dfc.properties` file or the client config object affects all sessions using those objects.

If you change the server config object, and reinitialize the server, 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 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.

Changing the assigned default operating system permissions (UNIX only)

When a client DFC 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 `dfc.data.umask` key in the `dfc.properties` file or by setting the corresponding property in the client config object.

Setting `dfc.data.umask` in the `dfc.properties` file affects all public directories and files created during sessions established using that file. Setting it in the client config object affects only directories and files created during repository sessions under that session manager.

The `dfc.data.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 664. Or, if you set `umask=22`, then the permissions become 755 for directories and 644 for files.

Defining short date formats

Different parts of the world have different shorthand 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. 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 and Language Options 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.

Changing the client local area directory location

To change the location of the client local area directory, you can:

- Set the `dfc.data.local_dir` key in the `dfc.properties` file.

Changing the key affects the current session and all future sessions using that file.

- Set the `dfc.data.local_dir` property in the client config object.

Changing this property affects the current session and any session established while the client config object persists. The object persists while the DFC remains initialized.

- Set the `dfc.data.local_dir` property in the session config object.

Changing this property affects only the current session.

Setting disk space limits for the client local area

You can set the `dfc.data.local_diskfull_limit` key in the `dfc.properties` file to set a limit on the size of the client local area. By default, this key is set to 0, which means the size of the client local area is allowed unlimited growth.

This is an integer key. If set to a value other than zero, the value is interpreted as megabytes and represents the maximum allowed size of the client local area. 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.

By default, when a client reaches the 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 `dfc.data.local_purge_on_diskfull` key in the `dfc.properties` file. It is a Boolean key and is set to TRUE by default during the installation procedure.

Removing content from client local areas

When users terminate their sessions, the files that they requested during their sessions are automatically removed from the client local area. If a session terminates abnormally,

for example, if the machine crashes, the local area is not cleaned up. However, by default, 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.

The default cleanup when starting a session is controlled by the `dfc.data.local_clean_on_init` key in the `dfc.properties` file. This key is true by default.

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 limit specified in the `dfc.data.local_diskfull_limit` key, you can use the `IDfSession.purgeLocalFiles` method to remove the files in your local area.

The 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 sessions A and B open, executing `purgeLocalFiles` from session A only removes files copied into the area during session A. The files requested by session B are not touched.

Any user can use this method.

Managing persistent client caches

Persistent client caches are caches of objects and query results managed by the client DFC and maintained across sessions. Objects and query results are placed in the caches by specific request when an application or user issues a caching fetch or query. 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 DFC methods that support object and query caching.

Cached data is checked for consistency with the repository based on consistency check rules defined when the fetch or query is executed. 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 persistent client caching and its use, refer to *Content Server Fundamentals*. This section contains instructions for enabling or disabling persistent client caching and for creating cache config objects.

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` property in the `docbase` 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 property 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

Persistent client caching is controlled at the session level by the `dfc.cache.enable_persistence` key in the `dfc.properties` file.

The 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 the key to FALSE, users cannot persistently cache objects or query results.

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. 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 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` properties.
These properties define the set of cached data managed by the cache config object. Refer to [Defining the cached data set, page 179](#), for information and guidelines for setting these properties.
4. Set the `server_check_interval`.
This determines how often Content Server validates the cached data. [Defining the server check interval, page 180](#), contains guidelines for setting this property.
5. Set the `client_check_interval`.
This determines how often a client application refreshes the cached data. [Defining the client check interval, page 181](#), contains information about how this property is used.
6. Save the cache config object.

Defining the cached data set

You must set two properties to identify the cached data managed by a cache config object: `cache_element_queries` and `cache_element_types`. Both are repeating properties. The `cache_element_queries` property contains a list of DQL queries. Each index position in the property 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` property 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 the IDfQuery object 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` property 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 is expressed as seconds and is defined in the `server_check_interval` property.

When a `CHECK_CACHE_CONFIG` administration method is issued, 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` property and the `r_last_checked_date` and `r_last_changed_date` properties 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` property.

When a session first references a cache config object, DFC 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, DFC 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, DFC 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 DFC is checked for consistency and automatically updated when needed, you may want to occasionally force a refresh. To force a refresh, you can:

- Flush the objects you want to refresh from a cache
- Reset the `client_pcaching_change` property in the doctype config object

Flushing a persistent cache

To flush a persistent client cache, use an `IDfSession.flush` method. You can flush cached objects or queries or both. The method 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 the Javadocs.

Setting the `client_pcaching_change` property

The `client_pcaching_change` property 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 property. 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, DFC 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 DFC 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 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` property of the `cache_config_object`.

For general information about creating jobs, refer to [Creating jobs and job sequences](#), page 145.

Overriding consistency checking rules

Overriding the consistency checking rules forces DFC to use the default consistency checking rule. The default is to always check. 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 `dfc.force_coherency_check` key in the `dfc.properties` file or the corresponding property in the client config object or connection config object. If you set `force_coherency_check` to T, the consistency checking rules defined in a fetch method or with a query are ignored by DFC. Instead, DFC uses the default consistency checking rule.

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 DFC. 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, DFC determines whether the client 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 `dfc.cache.write_interval` key in the user's `dfc.properties` file. The default interval is 3600 seconds (60 minutes). 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:

- Turn on RPC tracing and trace the fetch methods and query execution.

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, to test its behavior or results.

Use the `DO_METHOD` administration method to invoke the method. Set the `-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 DFC 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 181](#), 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 to determine if the problem persists. If the problem persists, it is likely that the persistent cache is not source of the problem.

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 187](#)
- [Servers and connection brokers, page 189](#)
- [Clients and connection brokers, page 190](#)
- [Failover for connection brokers, page 191](#)
- [Load balancing for connection brokers, page 191](#)
- [Configuring a connection broker, page 191](#)
- [Restarting a connection broker, page 195](#)
- [Starting additional connection brokers, page 196](#)
- [Shutting down a connection broker, page 197](#)
- [Obtaining information from a connection broker, page 199](#)
- [Obtaining information about connection brokers, page 200](#)
- [Deleting server information, page 201](#)

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 `dfc.properties` file. [Defining connection brokers for connection requests, page 163](#), 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 165](#), 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. [Starting additional connection brokers, page 196](#), 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 description of the repository

There are methods in the DFC IDfDocbrokerClient interface that return information held by connection brokers. For example, the `getDocbaseMap` method returns the information about repositories known to connection brokers. The `getServerMap` method returns the information for servers.

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 189](#),

contains a detailed description of the communications between servers and connection brokers.

Locating connection brokers

You can determine which connection brokers a client can talk to by issuing an `IDfDocbrokerClient.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 DFC 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 200](#), contains the procedure.

Alternatively, you can also examine the client's `dfc.properties` file.

Connection broker configuration options

A standard connection broker works as installed and needs no additional configuration. However, you can 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 to 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 191](#), 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, the checkpoint interval is five minutes; however, you can reset it to a different value ([Setting the checkpoint interval, page 116](#), 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 the interval ([Setting the keep entry interval, page 116](#), contains instructions).

When a system administrator shuts down a server gracefully, 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. For example, a client can be Webtop, an external application, or a user working with Documentum Administrator. In each case, at the start of a new client session, a connection must be established with a repository. To establish the initial connection, the client application first sends 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 also have backup connection brokers.

The default and backup connection brokers for clients are defined in the `dfc.properties` file. Each EMC Documentum web-based client application contains a `dfc.properties` file packaged with the application's WAR file. [Defining connection brokers for connection requests, page 163](#), contains information about defining connection brokers in a `dfc.properties` file.

Failover for connection brokers

Failover for connection information requests from a client is provided by identifying multiple connection brokers in the `dfc.properties` file and ensuring that the `dfc.docbroker.auto_request_forward` key in the `dfc.properties` file is set to `T`, which is the default.

By default, a connection request is handled by the first connection broker listed in the file. However, if that connection broker does not respond within 15 seconds or less or does not know about the requested repository, failover occurs if another connection broker is defined and `auto_request_forward` is `T`. The request is forwarded to the next connection broker in the file. If that connection broker does not respond, the request is forwarded to the next connection broker in the file. The request goes in turn to each connection broker listed in the file until a connection broker responds successfully or until all connection brokers defined in the file have been tried. If there is no successful response, the connection request fails.

Load balancing for connection brokers

To ensure that connection requests do not all fall on the first connection broker identified in a `dfc.properties` file, use the `dfc.docbroker.search_order` key in the `dfc.properties` file. The `search_order` key determines whether the request is sent first to the first connection broker or to a connection broker randomly selected from the list of connection brokers in the file.

Setting this key to “random” causes DFC to randomly select a connection broker from those specified in the `dfc.properties` file for a connection request. Randomly selecting a connection broker from the list of connection brokers in the 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)
Configure a connection broker to require a password whenever the connection broker is shut down.
- Server access restrictions
Configure a connection broker to accept or reject broadcasts from specific servers.

- Specific IP listening addresses

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 196](#), contains instructions on this option.

- IP address translation

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 and 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 password key in the security [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 when 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 startup 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. Enter the 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.

Note: The connection broker service name is Documentum Docbroker Service *connection_broker_name* where *connection_broker_name* is the name of the connection broker. The default service name is Documentum Docbroker Service Docbroker.

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\6.0\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 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 or you can run them on the same machine. If you 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\6.0\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. Modify the projection properties in the server config object 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 114](#), 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 shut down the connection broker.

Windows platforms

A connection broker on a Windows host may have been started as a 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 executing `dm_stop_docbroker` in the interactive mode (without the `-B` flag), the utility tells you which connection broker it intends to stop and prompts for a 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 193, contains information about imposing security on connection brokers.)

If 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 to be stopped when the `dm_stop_docbroker` utility is run. The following 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`. 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 specified 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$bigbark $@
#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 an `IDfDocbrokerClient.getServerMap` method to obtain information about the servers known to a connection broker that access a particular repository.

The `Getservermap` method returns an `IDfTypedObject` object whose type name is `server_locator`. This is a non-persistent object that contains the information known to the connection broker about any server for the repository. The *Documentum System Object Reference Manual* contains a description of the properties in a server locator object.

The information for a single server appears at corresponding index positions in the object's repeating properties. 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 an `IDfDocbrokerClient.getDocbaseMap` method to obtain information about the repositories that servers can access.

The method returns an `IDfTypedObject` object whose type name is `docbase_locator`. A docbase locator object is a non-persistent object that contains information about the repositories associated with the servers known to the connection broker. The *Documentum System Object Reference Manual* contains a description of the properties in a docbase locator object.

The information for a single repository appears at corresponding index positions in the repeating properties. 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

To obtain information about which connection brokers a client can access, do one of the following:

- Use the `getDocbrokerMap` method.
- Query the client config object.

Using the getDocbrokerMap method

The `getDocbrokerMap` method returns an `IDfTypedObject` object whose type name is `docbroker_locator`. This object lists, in repeating properties, all connection brokers that are visible to the client session.

The information in the `docbroker_locator` object's properties is taken from the `dfc.properties` file that the client session is accessing. The information for a single connection broker appears at corresponding index positions in the repeating properties. For example, the values at `network_protocol[2]`, `host_name[2]`, `port_number[2]`, and `time_out[2]` describe one connection broker.

The method 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.

Querying the client config object

Each client session references a non-persistent client config object for some configuration information. The information in this object is taken from the `dfc.properties` file in use by the session. Included in that information are the connection brokers for the session.

The information about the connection brokers in the properties file is contained in the keys with the category `dfc.docbroker`. For example, a `dfc.docbroker.host` key identifies a connection broker host and a `dfc.docbroker.port` key identifies the port in use by a connection broker. The keys include indexes, with the information across any particular index representing the information about one connection broker.

In the client config object, the information in these keys is found in repeating properties of the same name. For example, all the names of connection brokers are found in a repeating property named `dfc.docbroker.host`.

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 occurs when a server is shut down with instructions to broadcast a delete me message. [Using the shutdown method, page 120](#), contains instructions.

- A server fails to broadcast a checkpoint message within the expected `keep_entry` interval.

This occurs when a server is not active as a result of a shutdown without a delete message, 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 (except property 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 204](#)
- [Summary of storage area configuration options, page 222](#)
- [Content and full-text indexes, page 223](#)
- [How objects, contents, and storage are connected, page 223](#)
- [Allocating content to storage areas , page 224](#)
- [Content Retention, page 231](#)
- [System-defined storage areas, page 232](#)
- [File paths and URLs for content files in storage, page 233](#)
- [Setting up storage, page 235](#)
- [Providing automatic file extensions, page 258](#)
- [Moving content files, page 259](#)
- [Maintenance operations for storage areas, page 263](#)
- [Administering content assignment policies, page 275](#)
- [Archiving and restoring documents, page 277](#)

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 Administrator User Guide* or in the Documentum Administrator online help.

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 the types of files you plan to store in the repository, how you intend to use those files, and how you designed the storage strategy in your installation. The storage 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 205](#), describes file store storage areas in detail.

- Retention storage

Retention storage areas store content that you want to retain for a specified time. They are often used for storing massive amounts of unchanging data, such as email archives or check images. There are two types of retention stores:

- EMC Centera store

EMC Centera stores can store metadata values with each piece of content and are described in [EMC Centera storage, page 212](#).

- NetApp SnapLock store

NetApp SnapLock storage systems are described in [NetApp SnapLock storage, page 215](#).

- Blob storage

Content in blob storage is stored directly in the repository, in a special table. [Blob store storage areas, page 217](#), describes blob storage areas.

- Turbo storage

Content in turbo storage is stored in a property in content or subcontent objects. [Turbo storage, page 218](#), 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 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 219](#), 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 221](#), describes the use of a linked store storage area.

Note: Linked store storage areas are deprecated. DFC versions 6 and later do not support linked store storage areas.

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 properties inherited by all the storage subtypes. The description of the type in the *Documentum System Object Reference Manual* lists these common properties.

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`. The *Documentum System Object Reference Manual* lists the properties 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 encryption and digital shredding options require 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` property of the storage area's `dm_filestore` object. The property 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 *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.

Note: The encryption key is 192 bits in length and is used with the Triple DES-EDE-CBC algorithm.

Content compression

This option requires a Content Storage Services license. Content compression is enabled when the storage area is created and cannot be reset later.

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.

A file store storage area may not use compression if that storage area will be a component of a distributed storage area. Distributed storage areas do not support compression.

Note: If your clients are configured to use compression (`dfc.content.use_compression` in the `dfc.properties` file is set to T), Content Server uncompresses the content received from the client and applies the compression algorithm used for the storage area if saving content to a storage area configured for compression.

Content compression characterization

These graphs provide information about how much a file is compressed when compression is enabled for a storage area. The percentages are averages representing the average percentages by which a file was compressed. For example, a small Powerpoint document (PPT) was compressed by approximately 42.41 %, meaning that after compression, the document was approximately 57.59% of its original size.

Figure 1, page 207, shows the compression percentages for small files in various formats.

Figure 1. Small file compression percentages

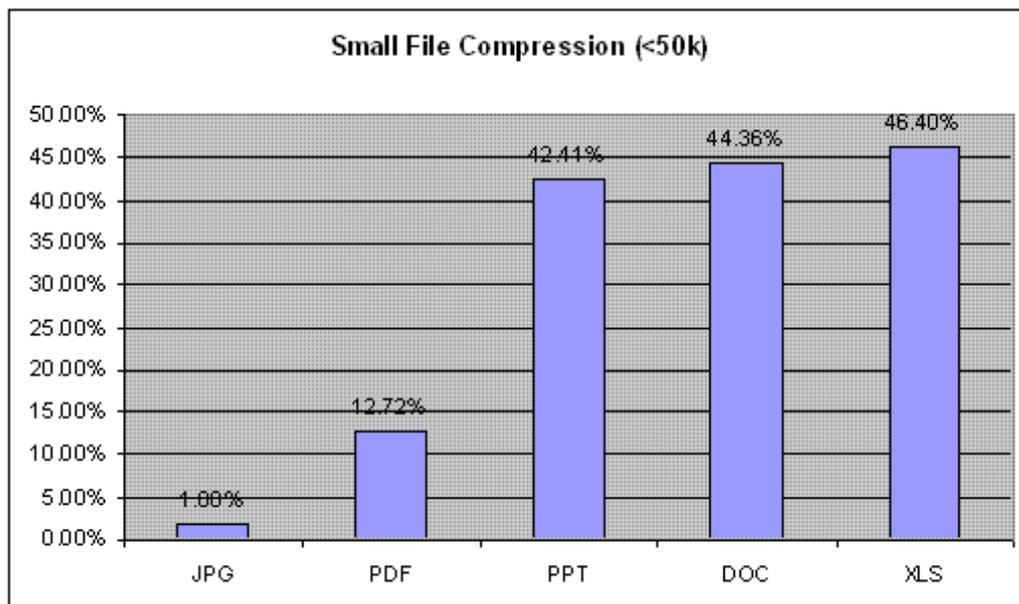


Figure 2, page 208, shows the compression figures for medium-size files in various formats.

Figure 2. Medium file compression percentages

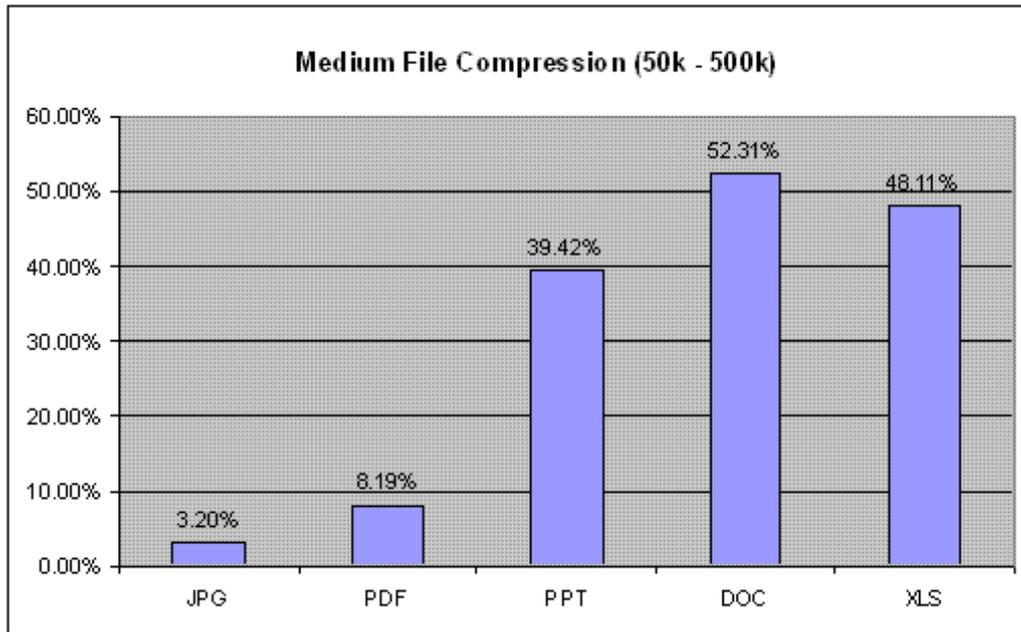
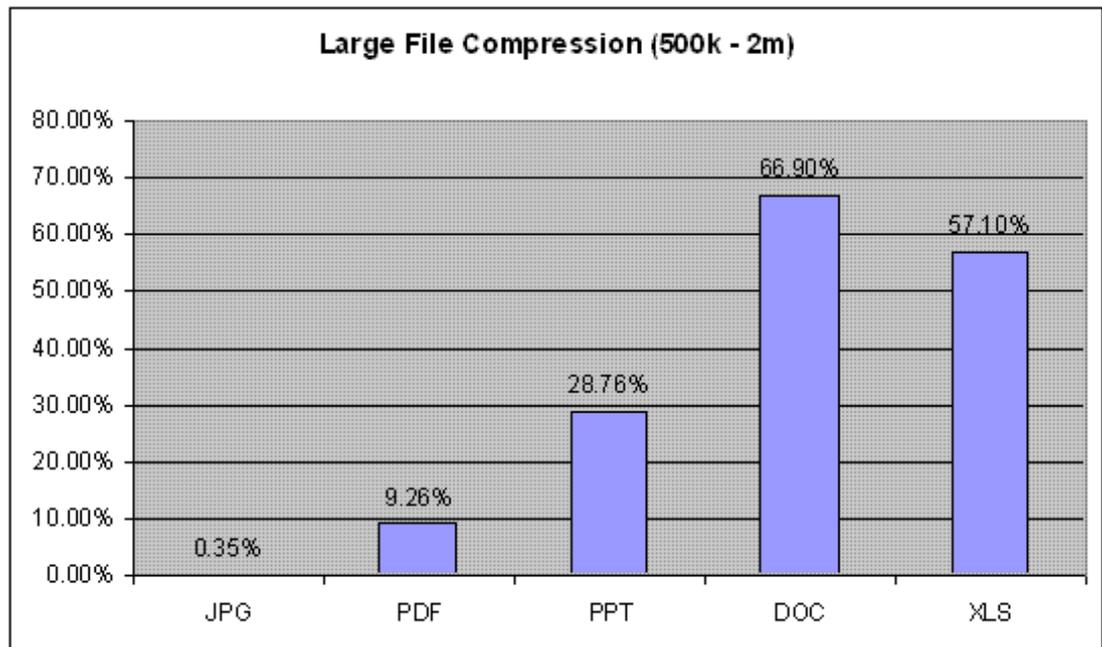


Figure 3, page 208, shows the compression figures for large files in various formats.

Figure 3. Large file compression percentages



Content duplication checking and prevention

This option requires a Content Storage Services license.

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.

A file store storage area may not use content duplication checking and prevention if that storage area will be a component of a distributed storage area. Distributed storage areas do not support content duplication checking and prevention.

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 a property 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 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 properties

Three properties support the content duplication checking and prevention functionality:

- `content_hash_mode`
- `content_dupl_pref`
- `r_content_hash`


```
    digestedData[0], digestedData[1], digestedData[2],
    digestedData[3], digestedData[4], digestedData[5],
    digestedData[6], digestedData[7], digestedData[8],
    digestedData[9], digestedData[10], digestedData[11],
    digestedData[12], digestedData[13], digestedData[14],
    digestedData[15], digestedData[16], digestedData[17],
    digestedData[18], digestedData[19]);

    return hashString;
}
```

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 `setServerTraceLevel` 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. This feature is not supported for file store storage areas that are components of a distributed storage area.

EMC Centera storage

Using EMC Centera storage requires a Content Services for EMC Centera license. If you install Content Server with that license, Content Server supports the use of the EMC Centera Storage System. (Refer to the release notes for the exact version.)

Use EMC Centera 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 Centera 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 Centera storage area.

Content in an EMC Centera 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 first content address generated for a content file is stored in the content object. Additional addresses are stored in the `i_contents` property of an associated `dmi_subcontent` object.

Content Server communicates with the storage system through a plug-in and shared library installed with Content Server.

In the repository, a Centera 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 Centera storage area, refer to [Setting up EMC Centera storage areas, page 245](#). For information about how to save documents to a Centera storage area, refer to *Content Server Fundamentals*.

Configuration options

You can set the following configuration options for an EMC Centera storage area when you create the storage area:

- A default retention value
- Content compression

Content compression is supported only if you have also installed Content Server with the Content Storage Services license.

- Content storage as embedded blobs
- Size of the C-clip buffer
- Number of retries for write attempts

- Override of the Centera single-instance checking configuration
- Maximum number of socket connections used by Centera SDK
- Use of the memory map interface for writes to the storage area in certain circumstances

If configured, the memory map interface is only used when content is moved from an unencrypted, uncompressed file store to the Centera storage area using `MIGRATE_CONTENT` or by setting an object's `a_storage_type`.

Default retention values

EMC Centera storage areas support the definition of a default retention value for content stored in those storage areas. The default is applied to all content saved into the storage area if a retention value is not set explicitly for the content when it is saved.

The default retention value is defined at the storage-area level and is enforced by the Centera host system. Content saved with a retention value, either explicit or defaulted, cannot be removed from the storage system until the retention expires.

You can define the default retention value as a number of days or as a hard date. If you specify a number of days, the content is held for the specified number of days counting from the time the content is saved to the storage area. If you specify a hard date, the content is retained until that date.

Note: The retention value defined for a Centera 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 Centera store retention periods interact, refer to [Defining storage area retention requirements, page 247](#).

Compression

Content stored in an EMC Centera 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 an EMC Centera 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 Centera 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 Centera 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 Centera store object, it applies only to content whose `a_storage_type` is set to that Centera store object. If there are multiple Centera 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 Centera 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 251](#).

This option is changeable after the storage area is created. You can start or stop its use or change the size of embedded blobs.

The C-clip buffer size

The C-clip buffer is a client (SDK) buffer used by Centera to manage C-clips created by an application. The default size for this buffer is 150K. If an application is managing a large number of concurrent sessions or using embedded C-clips or both, the default size may not be large enough. If the buffer is full, Centera pages out the overflow to files on the local disks. This can be a performance issue. You can reset the buffer size.

For instructions on resetting the buffer size, refer to [Setting the C-clip buffer size, page 252](#).

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 attempts to write a content file to the storage area, the interval between retries, and whether a final attempt is made after the maximum number of attempts has been made.

For more information and instructions regarding the retry options, refer to [Configuring write attempts in EMC Centera storage areas](#), page 253.

Override of the Centera storage strategy configuration

Centera systems can be configured to store a single copy of the content when the same content is submitted for storage multiple times. For example, that storage strategy is useful when the content is an attachment to an email sent to multiple recipients. In these cases, you probably want to store only one copy of the attachment, instead of one for each recipient.

In some cases, you may want to override the Centera storage strategy. You can set a storage parameter for a Centera store storage area so that all content saved to that storage area bypasses the storage strategy configured for the Centera host on which the storage area resides. For instructions, refer to [Overriding the Centera single-instancing configuration](#), page 253.

Maximum number of socket connections used by the Centera SDK

By default, the Centera SDK can use up to 99 socket connections with the Centera host. You can change that maximum by setting a storage parameter for the Centera storage area. For instructions, refer to [Resetting the maximum socket connections allowed](#), page 254.

Use of the memory map interface for writes to the storage area

If this option is configured, the memory map interface is only used when content is moved from an unencrypted, uncompressed file store to the EMC Centera storage area using MIGRATE_CONTENT or by setting an object's `a_storage_type`.

Using the memory map interface increases the performance of the write operation. There are two pool options that configure this option. For instructions, refer to [Configuring use of a memory map for write operations](#), page 254.

NetApp SnapLock storage

Network Appliance SnapLock (NetApp SnapLock) is a licensed software that provides storage level retention capability through the creation of Write Once Read Many (WORM) volumes on Network Appliance systems. These WORM volumes enable users to prevent altering or deleting content until a specified retention date.

Use NetApp SnapLock storage systems to store massive amounts of unchanging data, such as email archives or check images. There are two types of NetApp SnapLock stores:

- SnapLock Compliance store handles data retention to meet SEC regulations.
- SnapLock Enterprise store handles data retention to help customers meet their self-regulated date retention requirements.

Refer to the SnapLock documentation provided by Network Appliance for more information about the two types of stores.

Content Server communicates with the storage system through a plug-in and shared library installed with Content Server.

In the repository, a SnapLock storage area is defined using a ca store object. It also defines the default retention period, if any.

For instructions on setting up a NetApp SnapLock storage area, refer to [Setting up NetApp SnapLock storage areas, page 255](#)

Configuration options

You can set the following configuration options for a NetApp SnapLock storage area when you create the storage area:

- A default retention value
- Content compression

Content compression is supported only if you have also installed Content Server with the Content Storage Services license.

Default retention values

NetApp SnapLock provides basic retention enforcement capabilities. SnapLock storage areas support a default retention value for content stored in those storage areas. The default is applied to all content saved into the storage area if a retention value is not set explicitly for the content when it is saved.

The default retention value is defined at the storage-area level and is enforced by the SnapLock host system. Content saved with a retention value, either explicit or defaulted, cannot be removed from the storage system until the retention expires.

You can define the default retention value as a number of days or as a hard date. If you specify a number of days, the content is held for the specified number of days counting from the time the content is saved to the storage area. If you specify a hard date, the content is retained until that date.

Note: The retention value defined for a SnapLock 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 in the SnapLock system, and the retention date furthest in the future is enforced.

Compression

Content stored in a NetApp SnapLock storage area may be stored as compressed content if you have installed Content Server with a Content Storage Services license. Content compression is enabled when the storage area is created and cannot be reset later. 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.

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.

Note: Linked store storage areas are deprecated. DFC versions 6 and later do not support linked store storage areas.

One property, named `ascii`, is defined for the blob store type. It is a Boolean property 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` property 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` property of the `dmr_content` object. You can store up to 2000 bytes in the property 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 property. 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 an `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` property.)

Note: Linked store storage areas are deprecated. DFC versions 6 and later do not support linked store storage areas.

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 Content Server Distributed Configuration Guide* describes how to implement and administer a distributed storage area. The *Documentum System Object Reference Manual* lists the properties 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.

Note: Support of optical devices is deprecated. DFC versions 6 and later do not support optical storage devices.

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` 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 external store type is a subtype of `dm_store` and its properties 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.

Note: Support for optical storage devices is deprecated. DFC versions 6 and later do not support optical storage devices.

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. EMC Documentum clients and Content 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 when the plug-in is executed on the server host 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.

Note: Executing the plug-in on the client host is not supported by DFC version 6 or later.

You can create XML stores using the external free store subtype. Use XML stores to store and query large volumes of XML content. An XML store is a native XML database that is fully optimized for XML content. To create an XML store using Documentum Administrator, follow the basic instructions to create an external free store. There are four attributes in the `dm_extern_store` type that are automatically inherited by `dm_extern_free` to support an XML store:

- `storage_class`: Indicates the purpose the store is used for, such as XML.
- `is_writable`: Indicates if content is pushed to an XML store that Documentum manages.
- `a_storage_param_name`: Indicates the path to the application server that hosts an Xhive database.
- `a_storage_param_value`: Indicates the location of the external XML file store.

The *Documentum XML Store Installation and Administration Guide* provides more information about external XML file stores.

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.

For Documentum 6 and later, the plug-in must be configured to execute on the server host. DFC version 6 or later does not support executing the plug-in on the client host.

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 [Appendix G, Plug-in Library Functions for External Stores](#).

Linked store storage areas

Note: Linked store storage areas are deprecated. DFC versions 6 and later do not support 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. The *Documentum System Object Reference Manual* lists the properties 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.

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 18, page 222, summarizes availability of the configuration options that are not specific to a particular type of storage area by storage area type. There are some configuration options that are not included in this table because they are specific to EMC Centera storage areas. These options are listed in [Configuration options, page 212](#).

Table 18. 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
EMC Centera Store	No	No	Yes	No	No	No
NetApp SnapLock 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/XML Store	No	No	No	No	No	No
Linked Store**	No*	No*	No*	No*	No*	Yes
Distributed 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.

**Linked store storage areas are deprecated . DFC versions 6 and later do not support linked store storage areas.

***Compression and content duplication checking and prevention are not supported for file stores if the file store is a component of a distributed storage area.

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.

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, how indexing works, and how to turn off content indexing, refer to the *Content Server Full-Text Indexing System Installation and Administration Guide*.

How objects, contents, and storage are connected

Content objects are used to associate 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 property `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` property of the content object and in the `a_storage_type` property of the associated document object. The `storage_id` property 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 a property that points, directly or indirectly, to an actual storage area:

- If the storage type is file store, the property is called `location_name`. The `location_name` property 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 property is called `link_location`.
On Windows platforms, the `link_location` property 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 property identifies the directory containing the logical link to the actual storage directory.
Note: Linked store storage areas are deprecated. DFC versions 6 and later do not support linked store storage areas.
- If the storage type is distributed store, the property is called `r_component`. The `r_component` property is a repeating property that contains the object IDs of the storage objects for the component storage areas.
- If the storage type is blob store, the property is the name property 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` property contains the value `dm_turbo_store` instead of the object ID for a storage object.
- If the storage type is an EMC Centera storage, the first index position [0] in the `a_storage_params` property 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.
- If the storage type is NetApp SnapLock storage, the Content Server uses the directory path to pass the value to the mount point.

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` property 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 property at the type or format object level and using the default storage algorithm. [Using content assignment policies, page 225](#), describes the implementation of content assignment policies and the requirements for using them. [Using the default storage algorithm, page 229](#), 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.

Note: Documentum Administrator's online help or user guide provides information about creating rules based on document properties. Contact Documentum Professional Services or Documentum Developer support if you need assistance in creating, implementing, or debugging a custom rule.

Each rule in a policy is identified with a particular storage area. The storage area can be either a file store storage area, EMC Center store storage area, or NetApp SnapLock 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

Define content assignment policies 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 repository 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.

Note: Documentum Administrator's online help or User Guide provides information about creating rules based on keywords or system-defined SysObject properties. Contact Documentum Professional Services or Documentum Developer support if you need assistance in creating, implementing, or debugging a custom rule.

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 folder in the System cabinet. All users in the repository (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 DFC called the policy engine. Any client application built on the DFC enforces content assignment policies automatically if the Content Storage Services license is present and the policy engine in DFC is enabled. The `dfc.storagepolicy.enable` key in `dfc.properties` controls whether the policy engine is enabled or disabled. The key is T by default, meaning the policy engine is enabled.

The policy engine enforces content assignment policies:

- 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 checks in an object.

The `IDfSysObject.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:

- Fetch an object, change its properties, and save the changes.
- An application sets the object's `a_storage_type` property.

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.

Behavior on encountering a rule error

Each assignment policy defines one or more rules for content storage. At runtime, if the policy engine encounters an error in a rule, for example, because of a bad property name, by default, the policy engine returns an error, and the save operation fails. You can configure the policy engine to ignore such an error in a rule and continue evaluating the following rules in a policy. For instructions, refer to [Configuring behavior on encountering a rule error, page 276](#).

DFC assignment policy information cache

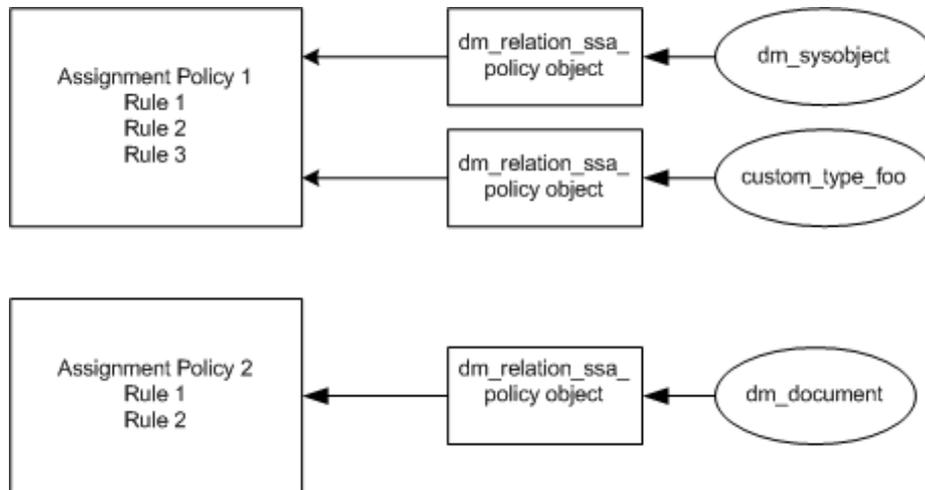
DFC maintains a cache of assignment policy information. By default, the cache is updated every 30 seconds. You can reset the interval. If you increase the interval, then it takes longer for policy changes to be recognized by the policy engine. If you decrease the interval, changes may be available for use sooner but performance may be degraded.

For instructions on resetting the interval, refer to [Configuring the update interval for the policy information cache, page 277](#).

Architectural 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`. A 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 4, page 228](#), illustrates how content assignment policies, as represented by ssa policy objects and object types are related.

Figure 4. 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 also deleted. 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` property 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.
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.
5. If no policy is found in the object's type tree, use the default storage algorithm.

[Using the default storage algorithm, page 229](#), describes 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 275](#).

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 property.

A user or application can specifically set the object's `a_storage_type` property 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` property of the content's associated format object. If that property 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 property is set to the thumbnail storage area. For all other

formats, the property must be manually set if you want to store content in a particular format in a particular storage area.

3. If neither a `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` property 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 properties:

- `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` property 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 property for an object before you save the object for the first time.

The `default_storage` property 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` property. For those formats, the property 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` property 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` property of the `dmi_type_info` object for the object type. The Content Server installation procedure sets the `default_storage` property 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 properties 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 is not defined in the addRendition method, the server checks for a value in the default_storage property of the content file's associated format object.
3. If a default storage area is not 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 three 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 *Content Server Fundamentals* or the *Documentum Administrator Users Guide*. For information about defining a retention policy, refer to Documentum Administrator online help.

- You can store documents in an EMC Centera storage area that has a defined or required retention period.

Documents whose content files are stored in an EMC Centera 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 Centera storage area, refer to [Defining storage area retention requirements, page 247](#). *Content Server Fundamentals* has information about forced deletions. [Enabling forced deletion in EMC Centera storage areas, page 265](#), describes how to enable forced deletions for a Centera storage area.

This option, which can be used in conjunction with Retention Policy Services, requires a Content Services for EMC Centera license.

- You can store documents in a Network Appliance (NetApp) SnapLock storage area that has a defined or required retention period.

NetApp SnapLock is a licensed software that provides storage level retention capability through the creation of Write Once Read Many (WORM) volumes on Network Appliance storage systems. These WORM volumes enable users to prevent altering or deleting content until a specified retention date has expired. This option, which can be used in conjunction with Retention Policy Services, requires a NetApp SnapLock license.

There are two types of NetApp SnapLock stores:

- SnapLock Compliance store handles data retention to meet SEC regulations.
- SnapLock Enterprise store handles data retention to help customers meet their self-regulated date retention requirements.

Refer to the SnapLock documentation provided by Network Appliance for more information about the two types of stores.

Both EMC Centera and NetApp SnapLock systems provide retention capabilities. However, installing Retention Policy Services provides additional functionality as shown in [Table 19, page 232](#).

Table 19. Availability of retention functionality

Functionality	EMC Centera	NetApp SnapLock
Default retention	Yes	Yes
Object-level retention	Requires RPS	Requires RPS
Retention holds	Requires RPS	Not available
Event-based retention	Requires RPS	Not available
Privileged Delete	Yes. Also available with RPS.	Not available

Note: In distributed environments, if a document is saved with retention, retention is enforced regardless of whether the content is written to the repository synchronously or asynchronously.

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` property is set to 0.
- `thumbnail_storage_01`
This is the default storage area for objects with primary content in thumbnail format. Its `media_type` property is set to 1.
- `streaming_storage_01`
This is the default storage area for object with primary content in streaming format. Its `media_type` property 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\<repository_name>` (`$DOCUMENTUM/data/<repository_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\repository_id\8a\xx\yy\mm[.ext]`

On UNIX platforms:

`storage_area_path/repository_id/8a/xx/yy/mm[.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`).
- *repository_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\repository_name` (`$DOCUMENTUM/dba/config/repository_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\repository_id\80\00\02\3b` (`storage_area_path/repository_id/80/00/02/3b`).

The name of the file is `3b`.

If the `use_extensions` property 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` property that points to the content file
- A `store` property that identifies the storage area containing the content file

If the storage area's `require_ticket` property 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=00232803/80/00/01/0b.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 a property 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 236](#), which briefly describes the implementation of creating distributed storage areas
- [Setting up blob storage, page 236](#), which contains information and instructions for creating a blob storage area
- [Setting up file store storage areas, page 237](#), which contains information and instructions for creating a file store storage area.
- [Linked store setup, page 239](#), which contains information and instructions for creating a linked store storage area.

Note: Linked store storage areas are deprecated. DFC versions 6 and later do not support linked store storage areas.

- [Setting up external storage, page 242](#), which contains information and instructions for creating external storage areas
- [Setting up EMC Centera storage areas, page 245](#), which contains information and instructions for setting up a Centera storage area
- [Setting up turbo storage, page 257](#), which contains information and instructions for using turbo storage areas

Distributed storage setup

A distributed storage area has 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 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. The *Documentum System 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` property of the blob object. If you set the content type to ASCII, the property 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 property 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 = 'blobstore1',
3>set ascii = true
4>go
```

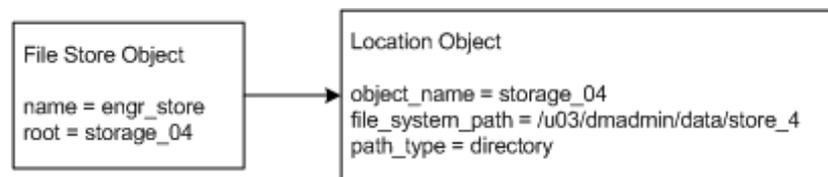
object_name is the name you assign to the blob store object.

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 also created.

The file store object defines the storage properties of the area. The root property 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 5, page 237](#).

Figure 5. 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.

Use Documentum Administrator to create file store storage areas, although you can also use DQL.

File extensions and the use_extensions property

By default, Documentum Administrator checks Use Extensions. 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 put files in a storage area.

The choice is recorded in the `use_extensions` property of the area's storage object. This is a Boolean property set to `TRUE` if the box is selected and `FALSE` if the box is not selected.

When the property 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 258](#), for complete details about implementing the use of file extensions when storing and retrieving files.

Setting the 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` property 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
```

The *Documentum System Object Reference Manual* has detailed description of the properties of location objects.

3. Use the CREATE...OBJECT statement to create the file store object for the area and set its properties.

In addition to the required properties, 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 205](#), for a discussion of these features. The *Documentum System Object Reference Manual* has a detailed description of the properties of file store objects

The following example sets only the basic properties.

```
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 property is the object name of the location object.

Linked store setup

Note: Linked store storage areas are deprecated. DFC versions 6 and later do not support linked store storage areas.

Linked storage areas 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.

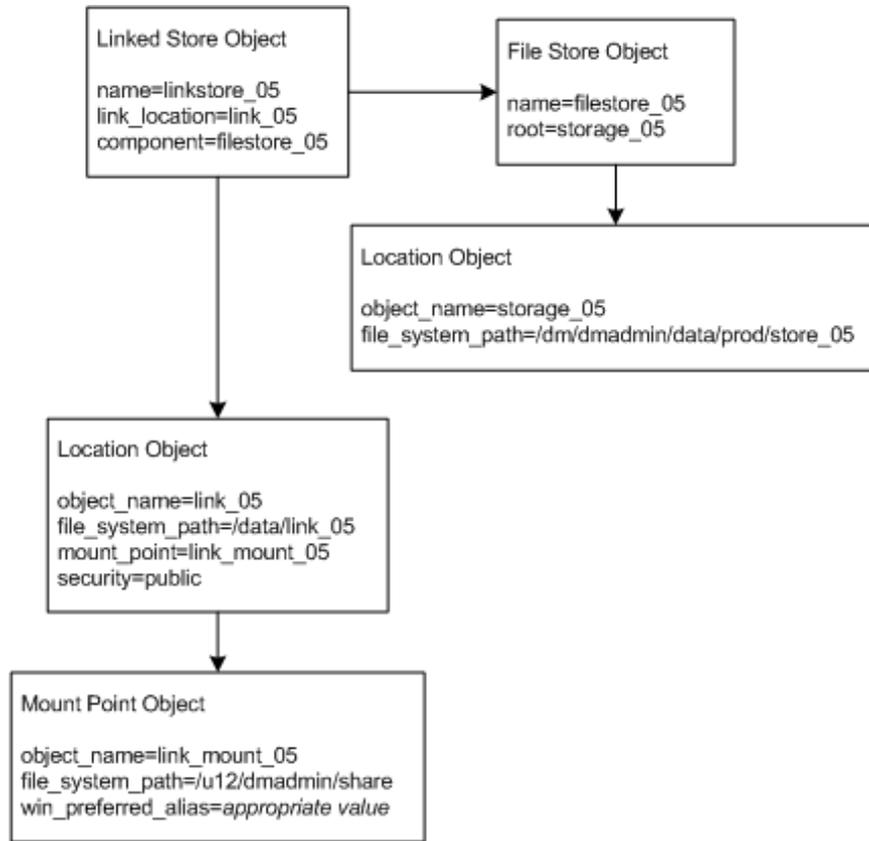
A mount point object has properties that let you specify the alias for the shared or mounted directory when referenced by clients. Set the appropriate property or properties

depending on your site's configuration. Refer to [Chapter 2, Content Repositories](#), for information about creating mount points and setting these alias-related properties.

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 6, page 240](#).

Figure 6. 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 237](#), 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\dmadmin\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 to set up an external storage area. EMC Documentum provides standard technical support for creating a plug-in object or external storage object. However, contact Documentum Professional Services or Documentum Developer Support for assistance in creating, implementing, or debugging the shared library or DLL that is the content of the plug-in object.

To create an external storage area:

1. Create the shared library or DLL.
The shared library or DLL must implement the functions described in [Appendix G, Plug-in Library Functions for External Stores](#). 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.
3. Create a plug-in object.
4. Create the external storage area.
You must configure the plug-in to execute on the server host.
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 `IDfSysObject.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 `fileName` argument in the `Setpath` command to:

```
'public\docs\extstore.doc'.
```

Using the Mount method

Note: The information in this section is only useful if the plug-in is configured to execute on the client host. Executing the plug-in on the client host is not supported by DFC version 6 or later.

The `IDfSysObject.mount` method provides a way to dynamically change the path to content in an external file store. 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 method to correct the path.

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 method.

Then, use `getFile` or `getContent` to retrieve the content from the CD-ROM.

If the `a_exec_mode` property 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` property 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 20, page 244](#) shows some examples of URLs and formats.

Table 20. 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 the `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 properties 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` property in the storage area object for the external storage area or set the `dfc.content.extern_store_content_static` key in the `dfc.properties` file. Both the property 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` property affects all `getFile` methods that access content in that storage area. Setting the `dfc.content.extern_store_content_static` key affects only `getFile` methods issued by sessions established using that `dfc.properties` file.

Setting up EMC Centera storage areas

To use an EMC Centera store storage area, you must have purchased a Content Services for EMC Centera license.

Using EMC Centera storage requires the EMC libraries, a plugin object, and a `dm_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 an EMC Centera 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, Linux, and HP Itanium, the library is `libemcplugin.so`
- On HP-UX PA-RISC, 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 plug-in are recorded in the server log file when the server loads the plug-in.

The properties 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 plug-in object. Optionally, you can also define the retention requirements in the storage object.

Setup procedure

To set up an EMC Centera 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 iTanium hosts, the variable is either SHLIB_PATH or LD_LIBRARY_PATH
 - For all other HP-UX hosts, the variable is SHLIB_PATH
 - For AIX hosts, the variable is LIBPATH
2. Use Documentum Administrator to create the EMC Centera storage area.

Note: If you wish to create an EMC Centera storage area without specifying a value for the a_plugin_id property, use IDQL or IAPI instead of Documentum Administrator to create the storage area. Creating the storage area without specifying the plug-in is useful in distributed environments. Refer to , for details.

Use the following guidelines:

- You can define up to 62 values for the Content Property Names. The names are recorded in the a_content_attr_name property of the associated ca store object. The values defined for a_content_attr_name may not contain spaces.
- [Defining storage area retention requirements, page 247](#), describes how to set retention requirements.
- [Defining the connection string , page 248](#), describes how to set the connection string.

This section includes information about setting the connection string correctly to configure use of Centera clusters.

- [Configuring embedded blob use, page 251](#), describes how to configure embedded blobs.
- [Setting the C-clip buffer size, page 252](#), describes how set the C-clip buffer size.
- [Configuring write attempts in EMC Centera storage areas, page 253](#), describes how configure the number of write retries.

- [Overriding the Centera single-instancing configuration, page 253](#), describes how to configure the storage area to override the Centera host's single-instancing configuration.
- [Resetting the maximum socket connections allowed, page 254](#), describes how to configure the maximum number of socket connections the Centera SDK may use.
- [Configuring use of a memory map for write operations, page 254](#), describes how to configure the use of the memory map interface for certain write operations to the storage area.

For additional help on setting the storage area properties, refer to Documentum Administrator online help.

Defining storage area retention requirements

An EMC Centera storage area will enforce a retention period if you choose to require one for the storage area. When you create an EMC Centera 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 EMC Centera 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 these properties of the ca store object:

- `a_retention_attr_required`

The `a_retention_attr_required` property defines whether content saved to this storage area requires a retention date. This property 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 value as a date, the value is recorded in the `a_default_retention_date` property.

- `default_retention_days`

If you choose to specify a default retention value as a number of days, the value is recorded in the `default_retention_days` property.

- `a_retention_attr_name`

The `a_retention_attr_name` property 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 setting the content object properties. This is done by including the name of the field storing the retention period in the parameters argument of the `SET_CONTENT_ATTRS` administration method or the `DFC setContentAttrs` method. If the retention field name is set when you set the content object properties, the retention period is set to the value defined by the `a_default_retention_date` property.

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.

Note: If a document is associated with a retention policy and its content is stored in an EMC Centera storage area with a retention value, 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.

Defining the connection string

The connection string is the value used by Content Server to connect to the Centera host. You must define the connection string for the EMC Centera storage area. The connection string is recorded in `a_storage_params[0]`.

Content Server supports any valid connection string documented and accepted by Centera. The typical format for a repository with a single server is:

```
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. If there are multiple Centera hosts identified in the connection string, the Centera SDK connects to the first available Centera node in the list.

The *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. If there is no profile specified in the connection string, the connection to the Centera host uses the anonymous profile.

If there are multiple servers servicing the repository, use the following format for the connection string:

```
svr_config_name="valid_Centera_connection_string"{, svr_config_name="valid_Centera_connection_string"}
```

where *valid_Centera_connection_string* is the format described for a single server.

Note: The value defined for `a_storage_params[0]` is passed directly to the `FPPool_Open()` Centera SDK function. Contact your Centera administrator to determine the correct connection string for your environment.

Required privileges for connection strings

The anonymous profile and any user-defined profile specified in a connection string must have the following EMC Centera permissions:

- read
- write
- delete
- exists

A profile used by a connection that is requesting a privileged delete must have privileged delete permissions.

To ensure that security is maintained, do not give other applications any permissions to the EMC Centera pool that Documentum Content Server uses.

Defining a connection string supporting EMC Centera clusters

Support for EMC Centera clusters is configured through the connection string. This support allows you to configure a storage area that represents Centera clusters so:

- A Content Server can write to the EMC Centera cluster closest to itself
- A Content Server can read from the EMC Centera cluster closest to itself and, if needed, fail over to read from another Centera cluster.

Note: Configuring bi-directional replication between the EMC Centera clusters will avoid the need for read failovers on the part of Content Servers.

The configuration is accomplished by identifying the primary and secondary clusters for each Content Server in the connection string. The specified primary cluster is the cluster that the associated Content Server will use for write operations and read operations. The specified secondary cluster is the cluster the server will use if an attempt to read content from the primary cluster fails.

The connection string is specified in `a_storage_params[0]`, in the `ca` store object. The format for the connection string when you are identifying primary and secondary clusters for one or more Content Servers is:

```
srv_config_name="primary=cluster_id,secondary=cluster_id[?Centera_profile]"[,srv_config_name="primary=cluster_id,secondary=cluster_id[?Centera_profile]"]}
```

where

- The primary *cluster_id* is the name or IP address of the EMC Centera cluster to which the Content Server will write
- The secondary *cluster_id* is the name or IP address of the EMC Centera cluster from which the Content Server will read if it cannot read from the specified primary cluster

Note: Including a Centera profile specification is optional.

The `a_storage_params` property is 1024 characters. Consequently, you must assign names to the Centera cluster nodes that are short enough to allow the full connection string to fit within the property.

Example of use

Suppose you have single-repository distributed configuration with two Content Servers at different sites and a Centera cluster at each site. The server config names are:

- `sc1` for Content Server 1
- `sc2` for Content Server 2

The names for the Centera cluster nodes are:

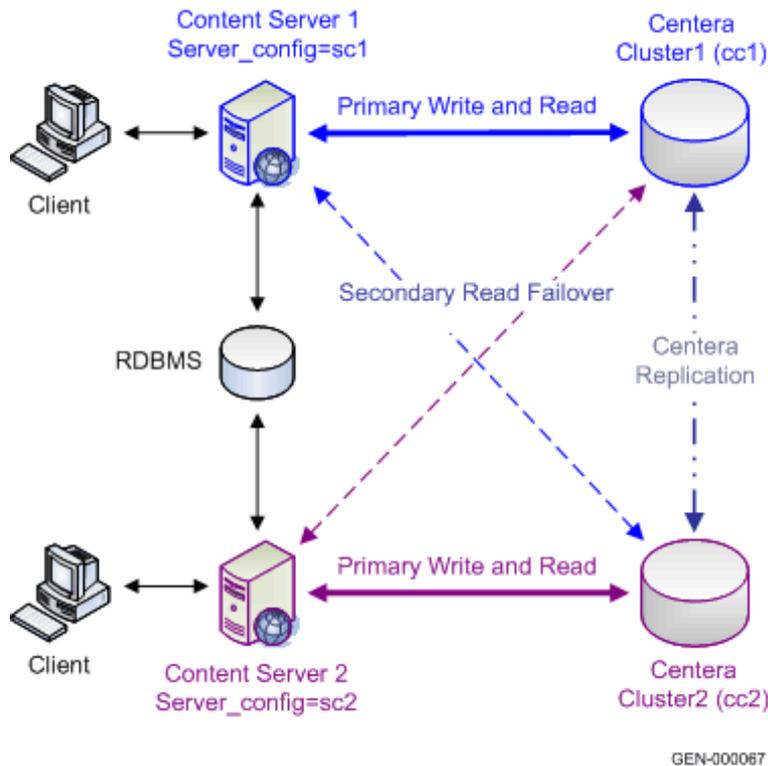
- `cc1` for Centera cluster 1
- `cc2` for Centera cluster 2

For Content Server 1, the server should write to Centera cluster 1 and Centera cluster 2 is the failover read cluster for the server. For Content Server 2, the server writes to Centera cluster 2 and Centera cluster 1 is the server's failover read cluster. To obtain that configuration, you set `a_storage_params[0]` to:

```
sc1="primary=cc1,secondary=cc2",sc2="primary=cc2,secondary=cc1"
```

[Figure 7, page 251](#), illustrates this configuration.

Figure 7. Content Server and EMC Centera cluster configuration in single-repository distributed environment



Configuring embedded blob use

Embedded blobs are a storage option for content stored in Centera storage. (Embedded blobs are described in [Whether to link or embed the content in the C-clip](#), page 214.) 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 embedded blob 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 Centera 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.

Setting the C-clip buffer size

The client (SDK) C-clip buffer is used by Centera to manage C-clips created by applications. The default size is 150K. To change the size, set a pool option in the `a_storage_params` property of the ca store object (at index position 1 or greater). The format is:

```
pool_option:clip_buffer_size:integer
```

where *integer* is an integer number representing the number of kilobytes. For example, suppose you want to enlarge the buffer to 200 kilobytes. You would specify the following in `a_storage_params`:

```
pool_option:clip_buffer_size:200
```

Setting `clip_buffer_size` sets the `FP_OPTION_BUFFER_SIZE` configuration parameter. This parameter controls the size of the client C-clip buffer.

If you require assistance determining the appropriate size for the C-clip buffer, contact Centera Technical Support.

Configuring write attempts in EMC Centera storage areas

By default, Content Server makes a maximum of three attempts to write content to an EMC Centera storage area. Between each attempt, Content Server waits a configured interval. You can reset the maximum number of attempts and the interval between attempts. 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 `dfc.content.castore.write_max_attempts` key in the `dfc.properties` file. The default value for that key is 3.

The interval between attempts is set by the `dfc.content.castore.write_sleep_interval` key. The value is specified in seconds and defaults to 3 if not set.

To direct Content Server to make a final attempt using Full Blob Name mode, set the `a_content_attr_name` property in the EMC Centera 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 Centera storage area using Full Blob Name mode.

Overriding the Centera single-instancing configuration

By default, the storage strategy configured on the Centera host is used when saving content to a Centera storage area. This configuration setting determines whether Centera checks for an existing instance of a content file on the system when saving a file or not. To override that setting and force Centera to save content without checking for duplicate instances of content on the host, set the following pool option in the `a_storage_params` property of the `ca` store object:

```
pool_option:use_collision_avoidance:T
```

If the option is set to `F` or not set, the storage strategy configuration defined in the Centera system is used.

Resetting the maximum socket connections allowed

By default, the maximum number of socket connections that the Centera SDK can establish with the Centera host is 99. You can reset the maximum, up to 999. To do so, add the following pool option in the `a_storage_params` property:

```
pool_option:max_connections:integer
```

where *integer* is the desired maximum number of connections.

Configuring use of a memory map for write operations

The memory map interface is only used when content is moved from an unencrypted, uncompressed file store to the EMC Centera storage area using `MIGRATE_CONTENT` or by setting an object's `a_storage_type`.

To use the memory map interface, you must set two pool options for the storage area. Pool options are set in the `a_storage_params` property of the `ca` store object associated with the storage area. You can use Documentum Administrator to set the options.

The two pool options that configure the use of this feature are:

- `use_mmap`

This pool option requires a Boolean value. It enables or disables the use of the memory map. For example, to enable the feature, set the following in `a_storage_params`:

```
pool_option:use_mmap:T
```

This option false by default, which means the feature is not enabled.

- `max_mmap_file_size`

This pool option defines the maximum size, in Kbytes, of the files that can be written to the storage using the memory map interface. For example, to set the maximum file size at 50 Kbytes, set the following in `a_storage_params`:

```
pool_option:max_mmap_file_size:50
```

The default size when this feature is enabled is 100 Kbytes.

The use of the memory map interface may require additional virtual memory.

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 NetApp SnapLock storage areas

To create a NetApp SnapLock storage area you must have purchased a Network Appliance SnapLock license and an EMC Connector license.

Using NetApp SnapLock storage requires a plugin object and a `dm_ca_store` object. Installing Content Server automatically installs the plugin object to use with NetApp SnapLock. The `ca` store object is created when you create a NetApp SnapLock 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 `netappplugin.dll`
- On Solaris, AIX, Linux, and HP Itanium, the library is `libnetappplugin.so`
- On HP-UX PA-RISC, the library is `libnetappplugin.sl`

The plug-in object is named SnapLock Connector. It stores the plug-in library to communicate with NetApp SnapLock. 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.

The properties in the `ca` store object identify the storage system, the location of the storage system, and the plug-in object. Optionally, you can also define the retention requirements in the storage object.

Setup procedure

To set up a NetApp SnapLock storage area:

1. Ensure 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 iTanium hosts, the variable is either SHLIB_PATH or LD_LIBRARY_PATH
 - For all other HP-UX hosts, the variable is SHLIB_PATH
 - For AIX hosts, the variable is LIBPATH
2. Use Documentum Administrator to create the NetApp SnapLock storage area.

Note: If you wish to create a NetApp SnapLock storage area without specifying a value for the a_plugin_id property, use IDQL or IAPI instead of Documentum Administrator to create the storage area.

Use the following guidelines:

- [Enabling content compression, page 256](#) describes setting content compression for a SnapLock store.
- [Defining SnapLock storage area retention requirements, page 256](#) describes how to set retention requirements.

For additional help on setting the storage area properties, refer to Documentum Administrator online help.

Enabling content compression

Content compression is a configuration option for a NetApp SnapLock store only if you have a Content Storage Services license. Content compression is enabled when the SnapLock storage area is created and cannot be reset later. [Content compression, page 206](#) provides additional information about content compression.

Defining SnapLock storage area retention requirements

A NetApp SnapLock storage area will enforce a retention period if you choose to require one for the storage area. When you create a SnapLock 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 NetApp SnapLock 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 these properties of the ca store object:

- `a_retention_attr_required`

The `a_retention_attr_required` property defines whether content saved to this storage area requires a retention date. This property is set to T (TRUE) 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 value as a date, the value is recorded in the `a_default_retention_date` property.

- `default_retention_days`

If you choose to specify a default retention value as a number of days, the value is recorded in the `default_retention_days` property.

- `a_retention_attr_name`

The `a_retention_attr_name` property records the name of the metadata field that stores the retention period for the content.

NetApp SnapLock supports extending the retention period; however, shortening the retention period is not allowed. An application can override a default retention date when setting the content object properties. This is done by including the name of the field storing the retention period in the parameters argument of the `SET_CONTENT_ATTRS` administration method or the `DFC setContentAttrs` method. If the retention field name is set when you set the content object properties, the retention period is set to the value defined by the `a_default_retention_date` property.

Note: If a document is associated with a retention policy and its content is stored in an NetApp SnapLock storage area with a retention value, 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.

Setting up turbo storage

Content stored in turbo storage is stored in the `i_contents` property of the associated content object or in the `i_contents` property 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` property of the object to `dm_turbo_store` before you issue a `setFile` method to associate the content with the object.

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 use public or linked storage areas, the stored files may also need file extensions so that your applications can access them directly.

Note: Linked storage areas are deprecated. DFC versions 6 and later do not support linked storage areas.

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` property 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` property for the area's storage object when you first define the storage area. This property is available for file store and linked store storage areas. It is a Boolean property. 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` administration 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 a storage area. The syntax also allows you to identify either a content object or instances of `SysObjects` or a `SysObject` subtypes as the target of the operation. If you specify a `SysObject` or subtype, 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 unchangeable objects.

For Documentum 6.5 repositories, administrators can optionally enter a Content Migration Threads value to enable the `MIGRATE_CONTENT` method to perform content migration in parallel. Use this option when performing mass migration of content to optimize execution time. When the value of this option is 0, which is the default, the migration process is sequential and only one object at a time is migrated. When the value of this option is greater than 1, the method executes in parallel using multiple internal sessions to concurrently migrate many content files. Each migration thread runs as an internal session, which is a thread on Windows or a process on UNIX. Each internal session requires at least one database connection and consumes additional system CPU cycles and memory. For this reason, it is recommended that mass migration be done when the system load is light or idle.

Note:

- The Content Migration Threads option displays only if you have a Content Storage Services license and the license is enabled on the Content Server.
- The number of internal sessions is limited to the Maximum Content Migration Threads value set in the server configuration object (`dm_server_config`), up to a maximum of 128.

MIGRATE_CONTENT lets you move files from a file store, EMC Centera store, NetApp SnapLock store, blob store, or distributed store. You can move the files to a file store, EMC Centera store, NetApp SnapLock store, or distributed store. You cannot move files to a blob store, and migration to and from external stores or XML stores is not supported. A file in a retention-enabled storage area can only be moved to another retention-enabled storage area. There are no restrictions on moving files out of a EMC Centera store or NetApp SnapLock store 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.

If you move a file that has a retention policy to an EMC Centera or NetApp SnapLock storage area, the retention set on the object is calculated from the retention policy even if the default retention for the storage area is further in the future. If there are multiple retention policies on the object, the method sets the retention on the file based on the policy with the retention date furthest in the future.

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 the *Content Server DQL Reference Manual*. You can execute the method using Documentum Administrator or DQL or DFC. You must be a Superuser to execute this method against a content object. To execute the method against the content of a SysObject or SysObject subtype, you must have Write permission on the object.

Note: If you are moving content from an unencrypted, uncompressed file store storage area to an EMC Centera store, you may wish to configure the storage area to use the memory-mapped interface. For more information, refer to [Configuring use of a memory map for write operations, page 254](#).

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, EMC Centera stores, NetApp SnapLock store, blob stores, and distributed stores to file stores, EMC Centera stores,

NetApp SnapLock stores, and distributed stores. Moving content to a blob store is not supported.

Note: You must have installed Content Server with a Content Storage Services license to use content migration policies.

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 administration method, page 259](#), on moving content files from retention-enabled EMC Centera or NetApp SnapLock 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 instances of `SysObjects` or `SysObject` subtypes. If the job runs against `SysObjects` or a subtype, 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 move the selected content files from, which can be a file store, EMC Centera store, NetApp SnapLock store, distributed store, or blob store
- The target storage area, which can be a file store, an EMC Centera store, NetApp SnapLock 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

- Optionally, the number of internal sessions to use to execute the method
The number of internal sessions is limited to the Maximum Content Migration Threads value set in the server configuration object (dm_server_config), up to a maximum of 128.

For detailed information about configuring this job, refer to the Documentum Administrator online help or to the *Documentum Administrator User 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/repository_id/sysadmin`.

Records migration jobs

Records migration jobs are an alternate way to move a large number of content files if you do not 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 451](#).

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 explicitly register to audit the event. You can register to audit the event for a particular content object (representing one content file) or a SysObject, or the object types (dmr_content and dm_sysobject). The methods that register events for auditing are found in the IDfAuditTrailManager interface.

If you audit the event for 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 audit the event for 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.

Maintenance operations for storage areas

This section contains information and instructions to maintain storage areas. Some maintenance operations, such as [Moving file store storage areas, page 264](#), typically are not necessary very often. Others, such as [Removing orphaned content objects and files, page 266](#), should be performed on a regular basis.

Changing the state of a storage area

A storage area can have one of three states:

- Online: Users can store files and copy files out when needed.

- **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.
- **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 IDfSession.apply() method. If you want to execute the method directly, refer to 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 property 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

You can move an entire file store storage area. For example, to 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` property 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` property contains the object name of the location object associated with the storage area.

5. Reinitialize the server and make the change visible.
6. Put the storage area back online:

```
EXECUTE set_storage_state
WITH store = 'filestore_name', offline = FALSE
```

You may remove the old storage area if you have no problems retrieving the contents of documents from the new storage area.

Enabling forced deletion in EMC Centera storage areas

Forced deletion allows a user to delete a document whose retention period is unexpired. If the content is stored in an EMC Centera storage area, the user performing the forced delete must connect through Documentum Administrator using a Centera profile that grants the privileged delete permission. *Content Server Fundamentals* has a complete description of forced deletions.

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_storage_params` property of the `ca` store object.

Instructions for setting the connection string in `a_storage_params` are found in [Defining the connection string](#), page 248.

- 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 Centera 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.

An orphaned content object is a content object that is not referenced by any SysObject. SysObjects have a property 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.

Documentum provides two automated system administration tools for this purpose. These tools automate two utilities:

- `dmclean`
- `dmfilesan`

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 11, Administration Tools](#). This section describes how to use the utilities manually.

There are multiple options for executing the `dmclean` and `dmfilesan` utilities:

- Documentum Administrator
- System administration tools
- DQL EXECUTE statement
- An `IDfSession.apply()` 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, an 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, EMC Centera storage, and NetApp SnapLock 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 EMC Centera and NetApp SnapLock 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 retention type storage areas, page 268](#), 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 21, page 267](#), lists the arguments that you can include to change the defaults.

Table 21. 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.
<code>-no_content</code>	Directs the utility not to remove orphaned content objects and files.
<code>-no_wf_templates</code>	Directs the utility not to remove orphaned SendToDistribution templates.

Argument	Description
-include_ca_store	Directs the utility to include orphaned content objects representing files stored in EMC Centera or NetApp SnapLock 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.
-clean_deleted_lwso	Directs the utility to remove deleted lightweight SysObjects and their parents.

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 property 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 retention type storage areas

The dmclean utility does not operate on content stored in EMC Centera or NetApp SnapLock storage areas by default. If you want to remove orphaned content files from these retention type 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 retention type 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 506](#), for information about the administration tool.

Running dmclean using an 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 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, an apply 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.

Using dmfilesan

Use dmfilesan utility to find orphaned content files—content files that have no associated content object. By default, the utility looks for all orphaned files that are older than one week. Executing the utility generates a 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, an IDfSession.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 property 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 22, page 270, lists the arguments to the dmfilesan utility.

Table 22. 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.

Argument	Meaning
<i>-from directory1</i>	Subdirectory in the storage area at which to begin scanning
<i>-to directory2</i>	Subdirectory in the storage area at which to end scanning
<i>-force_delete 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.
<i>-no_index_creation Boolean</i>	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. The default is F. Refer to Using the -no_index_creation argument, page 272 for details of use.
<i>-grace_period integer</i>	Defines the grace period for allowing orphaned content files to remain in the repository. The default is 1 week, expressed as hours. Dmfilesan removes orphaned files whose age exceeds 1 week or the value defined in the <i>-grace_period</i> argument. The integer value for this argument is interpreted as hours.

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 233](#).

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 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\dmsadmin\data\testora\content_storage_01\000003c9\80\00\04\8f
rem storage_id = '280003c980000100' and data_ticket = -2147482421
del \dm\dmsadmin\data\testora\content_storage_01\000003c9\80\00\04\cb
rem storage_id = '280003c980000100' and data_ticket = -2147482211
del \dm\dmsadmin\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 225](#).

Logging 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` property in the `dm_ssa_policy` object. Use Documentum Administrator to set that property.

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.storagepolicy.enable=false
```

All DFC-based applications that use that `dfc.properties` file are affected.

Configuring behavior on encountering a rule error

Each assignment policy defines one or more rules for content storage. At runtime, if the policy engine encounters an error in a rule, for example, because of a bad property name, by default, the policy engine returns an error, and the save operation fails. You can change this behavior by setting the following `dfc.properties` key to T:

```
dfc.storagepolicy.ignore_rule_errors=T
```

The key is F (FALSE) by default. If set to T (TRUE), when the policy engine encounters an error in a rule, the policy engine ignores the faulty rule and continues by evaluating the next rule in the policy.

Configuring the update interval for the policy information cache

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 30 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.

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 `IDfSession.archive` and `IDfSession.restore` methods and the Archive administration 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 EMC Centera or NetApp SnapLock storage option, rather than the archiving capabilities described in this section. The EMC Centera or NetApp SnapLock storage option is capable of storing massive amounts of data with a defined retention period. For information about this option, refer to [EMC Centera storage, page 212](#).

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

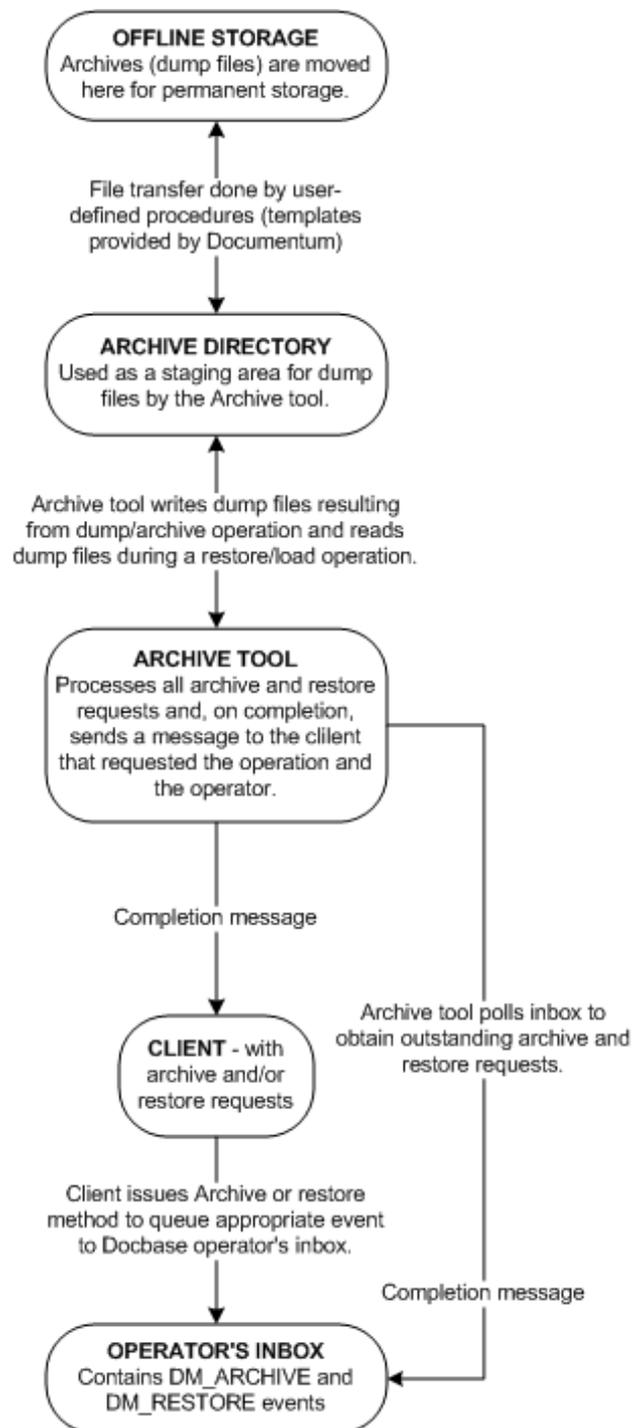
When you archive a document, these components interact as follows:

1. The client sends an archive request.
The client can be a user selecting a custom menu item requesting archiving (which issues an IDfSession.archive method) or an application issuing an 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. An IDfSession.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 8, page 279](#), illustrates the archive and restore mechanism. The following sections describe the process in more detail.

Figure 8. The archive and restore process



The archive and restore methods

Executing an `IDfSession.archive` method queues a `DM_ARCHIVE` event in the inbox of the repository operator, and executing a `IDfSession.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.

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 283, 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` property to `TRUE`
- Sets three properties 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, 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 = 'carolinew'
```

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='carolinew' 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='carolinew') 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 property 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 a 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 property of the content

object to FALSE. It does not reset the content object's `a_archive` or `set_file` properties, 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 = 'carolinew'
```

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 = 'carolinew')
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.

Documentum provides three `DO_METHOD` procedures to move files out of and into the archive directory. Each procedure is a script that contains comments about its use and an example command that is commented out. To use these scripts, edit the procedures. The scripts are located in `%DOCUMENTUM%\product\version\bin` (`$DOCUMENTUM/product/version/bin`). Copy the scripts 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 offline 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 150](#), and [Running administration jobs on demand, page 454](#), 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 requests

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. 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 restore method, the server assumes the operator is the user named in the `operator_name` property of its server config object. By default, this property is set to the repository owner. Documentum users with Sysadmin or Superuser privileges can change this property. Use Documentum Administrator to change the property. You must reinitialize the server after changing the property.

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 282, 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` property of the content object that links the document and its content. The `set_file` property 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` property. 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` property 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 property 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 282.

Starting the Archive tool

The Archive tool is active by default. For more information, refer to [Activation](#), page 453.

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 282 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` property 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` property to `TRUE` and removes the content from the storage area. The `set_file` property 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 properties defined for the storage area types. The user-defined program is represented in the repository by a `dm_method` object. The storage area properties are:

- `get_method`
- `offline_get_method`

The `get_method` property contains the name of the method object representing the user-defined program in the repository.

The `offline_get_method` property is a Boolean property 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.

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 290](#)
- [User privilege levels, page 291](#)
- [Privileged groups, page 294](#)
- [Adding users, page 295](#)
- [Granting and revoking user privileges, page 301](#)
- [Modifying users, page 303](#)
- [Renaming users, page 304](#)
- [Deleting users, page 304](#)
- [Deactivating, locking, and reactivating users, page 305](#)
- [Adding groups, page 307](#)
- [Modifying groups, page 309](#)
- [Deleting groups, page 309](#)
- [Changing the membership setting of a dynamic group, page 310](#)
- [Querying groups, page 311](#)

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 *Content Server 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 repository. User objects have four properties that define user names:

- `user_name`

The `user_name` property 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` property 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.

- `user_os_name`

The `user_os_name` property 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` property may be blank.

- `user_login_name`

The `user_login_name` property 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 repository session. Every Documentum user must have a `user_login_name`. The default value of the `user_login_name` is the `user_os_name`.

- `user_db_name`

The `user_db_name` property 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` property 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 properties to have the same value; however. 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 properties must be compatible with the code page defined in the `server_os_codepage` property 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 301](#), 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 23, page 291](#), lists the basic user privileges.

Table 23. 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 	8

Privilege	Description	Corresponding Integer Value
Superuser	<ul style="list-style-type: none"> • Manipulate workflows or work items, regardless of ownership • Manage any object's lifecycle • Set the a_full_text property <p>User can</p> <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 376](#), 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

These privileges define security-related operations that users can perform. [Table 24, page 293](#), lists the extended user privileges.

Table 24. Extended user privileges

Privilege	Description	Corresponding Integer Value
Config Audit	User can execute the 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 a set of privileged groups. Privileged groups are groups whose members are allowed to perform privileged operations even though the members do not have the privileges as individuals. The privileged groups are divided into two sets.

The first set of privileged groups are those that are available for use in applications or for administration needs. With two exceptions, these privileged groups have no default members when they are created. You must populate the groups. [Table 25, page 294](#), describes these groups.

Table 25. Privileged groups

Group	Description
dm_browse_all	Members of this group can browse any object in the repository. The dm_browse_all_dynamic is a member of this group by default.
dm_browse_all_dynamic	This is a dynamic role group whose members can browse any object in the repository. The dm_browse_all_dynamic group is a member of the dm_browse_all group.
dm_retention_managers	Members of this group can: <ul style="list-style-type: none"> • Own retainer objects (representing retention policies) • Add and remove a retainer from any SysObject. • Add and remove content in a retained object • Change the containment in a retained virtual document This is a non-dynamic group.
dm_retention_users	Members of this group can add retainers (retention policies) to SysObjects. This is a non-dynamic group.
dm_superuserusers	Members of this group are treated as Superusers in the repository. The dm_superuserusers_dynamic group is a member of this group by default.
dm_superuserusers_dynamic	A dynamic role group whose members are treated as Superusers in the repository. The dm_superuserusers_dynamic group is a member of the dm_superuserusers group.

Group	Description
dm_sysadmin	Members of this group are treated as users with Sysadmin user privileges.
dm_create_user	Member of this group have Create User user privilege.
dm_create_type	Member of this group have Create Type user privilege.
dm_create_group	Member of this group have Create Group user privilege.
dm_create_cabinet	Member of this group have Create Cabinet user privilege.

The second set of privileged groups are privileged roles that are used internally by DFC. You may not add or remove members from these groups. The groups are:

- dm_assume_user
- dm_datefield_override
- dm_escalated_delete
- dm_escalated_full_control
- dm_escalated_owner_control
- dm_escalated_full_control
- dm_escalated_relate
- dm_escalated_version
- dm_escalated_write
- dm_internal_attrib_override
- dm_user_identity_override

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 or be a member of the dm_create_users group.

New users can be added through Documentum Administrator or DQL. If you use an LDAP directory server to authenticate and manage users, then you add the new users as entries in the LDAP directory server using the procedures from the directory server documentation. The new users in LDAP are then propagated automatically to the repository.

If you are not using an LDAP directory server, Documentum Administrator is the fastest and most convenient way to add one user or a few users. 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 properties

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` property. 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` property. This name is used by Content Server to authenticate a user when he or she connects to the repository. If a domain name is not required in the repository, the `user_login_name` must be unique within the repository. If a domain name is required in the repository, the combination of `user_login_name` and `user_domain_name` must be unique within the repository.

The user's address is defined in the `user_address` property. 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 properties 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 does not identify an alternate storage location. If you do not 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

The *Documentum System Object Reference Manual* has a complete list of the properties of the user object type. 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 291](#), lists the valid user privileges.

If you are setting the `user_privileges` property directly, using DQL, 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 301](#), contains information on how to do this.

Defining the default ACL

You can set two security properties for new users: `acl_name` and `acl_domain`. [Table 26, page 297](#), describes these properties and how they are handled for a new user.

Table 26. Security properties for new users

Property	Description
<code>acl_name</code>	<p>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 property, the server creates an internal ACL in the user's domain as the default ACL.</p> <p>The ACL grants the following permissions: <code>dm_world</code> receives Read permission; the owner receives</p>

Property	Description
	Delete permission; and the user's group receives Version permission.
acl_domain	The name of the user who owns the ACL identified by acl_name. If you do not set this property, the server sets it to the repository owner or the user's name, based on the ACL specified in acl_name.

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 property. This property holds the user's RDBMS login name.

Setting default_folder

The default_folder property specifies 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 property 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.

Note: When a user who is restricted to designated folders starts a repository session through Webtop, the user is also given the ability to access the /System folder and their default folder or cabinet for the length of the repository session.

Setting client capability

The client capability setting is used by Documentum client products, such as Webtop, 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

Creating a new user with DQL is a two-step process.

To add a user using DQL:

1. 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 properties.

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 properties are `user_name`, `user_login_name`, and `user_address`. The *Documentum System Object Reference Manual*, contains a list of all user properties. [Setting user properties, page 296](#), contains a discussion of other commonly set properties.

To set additional properties later or to modify a property setting, use the UPDATE...OBJECT statement. [Modifying users, page 303](#), contains more information.

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 property 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 27, page 300](#).

Table 27. Default values for new users

Argument	Default
<code>user_login_name</code>	<i>username</i>
<code>privileges</code>	0 (None)
<code>folder</code>	<i>/username</i>
<code>group</code>	docu
<code>client_capability</code>	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. Each line in the file has format:

```
property_name:value
```

The entry for each user must begin with:

```
object_type:dm_user
```

That line marks the beginning of the description of the properties for a user. Anything between `object_type:dm_user` and the next line that starts with `object_type:dm_user` describes one user object for the server to create.

The properties you can set through the LDIF file are:

```
user_name  
user_os_name  
user_os_domain  
user_login_name  
user_login_domain  
user_password  
user_address  
user_db_name
```

```

user_group_name
user_privileges (set to integer value)
default_folder
user_db_name
description
acl_domain
acl_name
user_source (set to integer value)
home_dobase
user_state (set to integer value)
client_capability (set to integer value)
globally_managed (set to T or F)
alias_set_id (set to an object ID)
workflow_disabled (set to T or F)
user_xprivileges (set to integer value)
failed_auth_attempt (set to integer value)

```

The properties `user_name` and `user_os_name` are required.

The properties may be included in any order after the first line (`object_type:dm_user`). The Boolean properties are specified using T (for true) or F (for false). Use of “true”, “false”, “1”, or “0” is deprecated.

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 be system ACLs. They cannot be private ACLs.

Any groups that you identify in `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 291](#), 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 DFC.

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 property of the user's dm_user object. The user's extended user privileges are stored in the user_xprivileges property.

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 309](#), 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 DFC

To grant or revoke user privileges through DFC, use the `setUserPrivileges` and `setUserXPrivileges` methods in the `IDfUser` interface. The user's basic user privileges, set by `setUserPrivileges`, are recorded in the `user_privileges` property. The user's extended user privileges, set by `setUserXPrivileges`, are recorded in the `user_xprivileges` property. The `user_privileges` and `user_xprivileges` properties are integer properties. Consequently, each method takes an integer argument, representing the integer value representing the privilege or privileges you want to assign. ([Table 23, page 291](#), and [Table 24, page 293](#), 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 argument to 1. To assign the user the privileges to create types and create cabinets, set argument to 3 (1 for Create Type plus 2 for Create Cabinet).

The value in the argument for either method overwrites the current value of the corresponding property. To add or revoke a user's privileges, check the value of the property 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` property 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` property to 4 (subtracting the value of Create Type from the current setting and resetting the property to the remainder).

Modifying users

You can use Documentum Administrator, DQL, or DFC 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 301](#), 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. 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
```

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 523](#).

You cannot rename the repository owner, installation owner, or yourself.

Deleting users

You can delete users using Documentum Administrator, DQL, or DFC. 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 305](#), 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 a property 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. However, the recreated user will not inherit the previous user's group memberships and object permissions.

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
```

The *Content Server DQL Reference Manual* has the full syntax.

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` property 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 a property.

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.

A user may be deactivated automatically if the value in the user's failed_auth_attempt property exceeds the maximum number of failed authentication attempts defined in the repository configuration, in the max_auth_attempt property of the doabase config object. ([Limiting authentication attempts, page 356](#), 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 login 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 property 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 property to 1, 2, or 3. Setting the property 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 property 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
```

The *Content Server DQL Reference Manual* has a full description of the UPDATE OBJECT statement.

Adding groups

You can use Documentum Administrator, DQL, or DFC 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` property 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 as 'room private groups'. A room private group is any group for which the `group_native_room_id` property 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 361](#), 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` property.) 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 310](#), contains instructions on changing the default membership setting.)

The members of a group can be individual users or other groups. A non-dynamic group may 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. Group names are not case sensitive. All group names are stored in the repository in lowercase.

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 property 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 307](#), for information about public and private groups.

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 DFC 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 property, either as an individual or as a member of a group specified in the property

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 property 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
```

The *Content Server DQL Reference Manual* has a full description of the ALTER GROUP statement.

Deleting groups

You can use Documentum Administrator, DQL, or DFC 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 properties 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 `engr`:

```
1>drop group engr
2>go
```

The *Content Server DQL Reference Manual* has a full description of the `DROP GROUP` statement.

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` property of the group object.

When a dynamic group is created, the property 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 property 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 property, 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 property. This property 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 property's value is cached on the server and client side. If the property 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 the you query the property 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 an `IDfSession.revert` method on the group object before querying the computed property. Issuing the revert method invalidates the cached copy, forcing the server to recompute the property'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_supergroups_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_supergroups_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 313](#)
- [The assume user and change password programs, page 315](#)
- [Using the default authentication mechanism, page 316](#)
- [Using a custom external password checking program, page 319](#)
- [Using Windows domain authentication for UNIX users, page 320](#)
- [Using an LDAP directory server, page 323](#)
- [Using authentication plug-ins, page 346](#)
- [Using an in-line password, page 351](#)
- [Trusted logins, page 351](#)
- [Unified logins, page 351](#)
- [Managing encrypted passwords, page 352](#)
- [Limiting authentication attempts, page 356](#)

Authentication options

User authentication typically occurs when a user attempts to connect to a repository. (It can also occur when another client application, 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. [Using the default authentication mechanism, page 316](#), 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 319](#), 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. Content Server 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 323](#), 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 two authentication plug-ins with Content Server; one for EMC RSA and one for CA SiteMinder.

The EMC RSA Plugin for Documentum allows you to use the RSA Access Manager with Content Server for user authentication.

The CS Siteminder plug-in allows you to use CA SiteMinder with Content Server.

Both plug-ins support 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 RSA plug-in, page 348](#), contains a listing of the platforms supported by the RSA plugin and instructions on using the plugin.

[Using the CA SiteMinder plug-in, page 348](#), contains a listing of which platforms are supported by the CA SiteMinder 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 349](#), contains instructions.

In-line password

A user can be authenticated using an encrypted password that is stored in the `user_password` property of the user object. [Using an in-line password, page 351](#), contains instructions for using in-line passwords.

The assume user and change password programs

The assume user and change password programs are two programs that Documentum provides and implements by default for your installation.

Assume user

The assume user program is called whenever a client (end user or application) executes a `DO_METHOD` function for a procedure for which the `run_as_server` attribute is set to `FALSE`. When the procedure is executed (and the server knows where to find the assume user program), the server runs the procedure as the logged-in user. If the server cannot find the assume user program because `assume_user_location` is set to null, or if the `run_as_server` attribute is set to `TRUE`, the procedure executes under the server's account.

This feature is enabled by default. The Setup program copies the assume user program to the \$DOCUMENTUM/dba directory and sets the assume_user_location property of the server config object to this location.

If you want to disable this feature, set the assume_user_location property in the server config object to blank and reinitialize the server.

Change Password

Change password is called whenever the server receives a request to change a password.

This program is copied by the Setup program to \$DOCUMENTUM/dba. The installation script also sets the change_password_location property of the server config object to point to the file in \$DOCUMENTUM/dba.

If you want to disable this capability, set the change_password_location property to blank and reinitialize the server.

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 stored in a manner that requires root-level privileges to access. Rather than requiring Content Server be setuid to root, an external password checking program is provided that is setuid to root. This program calls the appropriate operating system calls to verify the users' passwords. The source code for this program is also provided so you can verify its functionality and add any required customizations.

The files in which the passwords are actually stored are:

- /etc/shadow on Solaris and Linux
- /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. A location object, named validate_user, created during installation, points to the location of the

check password program. The location object is referenced in the server config object by the `user_validation_location` property. When a user is authenticated, the server uses the location object named in the `user_validation_location` property to find the password checking program.

`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 319](#), 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 320](#), 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` property 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` property. 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` property. The property 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` property.

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` property 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` property in the `docbase` config object. If the repository is in the default mode, this property is blank. If the repository is in the domain-required mode, this property'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 properties manually. The script not only resets the `auth_protocol` property, 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.



Caution: 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 property 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 property in the docbase config object to unix_domain_used.
3. Set up a domain controller map.
4. Modify the user_source property 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.



Caution: 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` property 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. The `dm_auth_config` object is 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 properties:

- `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` properties 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 properties 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 property

The value in the user_source property 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 property for each UNIX user to one of the following:

- 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 property. 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 is not 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 to all the repositories using the directory server by the `dm_LDAPsynchronization` job.

The LDAP support provided by Content Server allows you to map LDAP user and group attributes to user and group repository properties or a constant value. When the user or group is imported into the repository or updated from the directory server, the repository properties are set to the values of the LDAP properties or the constant. The mappings are defined when you create the LDAP config object. You can also change them later, adding additional mapped properties, changing their mapping, or deleting mappings.

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 properties for users and groups are the same in each participating repository

Note: Changing or deleting the mapping for `user_name`, `user_login_name`, or `group_name` after the first synchronization for the user or group is not recommended. Doing so may result in inconsistencies in the repository.

Constraints

Using an LDAP directory server has the following constraints:

- The `changePassword` method is not supported for users managed through an LDAP directory server.
- Dynamic groups are supported only on Sun Java System directory servers.

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 synchronization occurs.

Using multiple LDAP directory servers

It is possible to have multiple LDAP servers accessible to the Content Server or Servers servicing a repository. The LDAP servers that are available to a particular server are defined in the server config object, in the `ldap_config_id` and `extra_directory_config_ids` properties.

Typically, LDAP synchronization synchronizes with all the LDAP servers specified in these properties. However, you can configure synchronization to synchronize with any or all of the directory servers identified in a server config object. [Dm_LDAPSynchronization, page 479](#), contains instructions.

You can also define secondary LDAP servers for authentication failover. Those servers are specified in the ldap config object. For information on how authentication failover works and which properties are used, refer to [Authentication failover, page 330](#).

User and group synchronization

This section describes how synchronization of LDAP users and groups occurs and is managed within a repository.

Synchronization 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.

dm_LDAPSynchronization job

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 479](#), contains details about the job and [Activating the dm_LDAPSynchronization job, page 341](#), 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 344](#), contains information on using first-time synchronization.

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 or attributes to which the user's `user_name` property is mapped. For groups, renaming occurs if a change is detected in the LDAP attribute or attributes to which the group's `group_name` property is mapped.

- Inactivate users in the repository if they are deleted in the directory server.

If you are using iPlanet, you must enable the changelog feature to use the inactivation operation. (Instructions for enabling the changelog feature are found in the vendor's iPlanet Administration Guide.)

You can also define the operations performed by setting properties 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 sets any user properties that are mapped to LDAP attributes or a constant. It can also create the user's default group and folder if needed. The job will create the folder and group if the associated user properties, `default_folder` and `user_group_name`, are mapped to LDAP attributes and the value resolves to a non-existent folder or group. If `default_folder` is not mapped to a value, the job sets the default folder to `/Temp`. There is no default value for `user_group_name`.

When the job creates a new group in the repository from an LDAP group, 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 LDAPSynchronization job completes.

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.

How the synchronization job determines which LDAP servers to use

The dm_LDAPSynchronization job has an argument called `-source_directory`. This argument accepts a list of ldap config object names or the value `dm_all_directories`. When the job runs, it checks to determine which of the directories specified in the argument are found in the following properties of the server config object:

- `ldap_config_id`
This is a single-valued property.
- `extra_directory_config_id`
This is a repeating property.

The job uses all the directories specified in the argument that are also identified in one of those properties for synchronization. If the value of the `-source_directory` argument is `dm_all_directories`, the job synchronizes with all the LDAP directories identified in the properties. The job connects to each LDAP server in turn and synchronizes users and groups in the repository based on the users and groups in the LDAP server entries.

Attributes set by the dm_LDAPSynchronization job

The dm_LDAPSynchronization job sets the following properties in addition to the mapped repository properties:

- For users:
 - `user_source`
 - `user_login_domain`
 - `user_global_unique_id`
- For groups:
 - `group_directory_id`
 - `group_global_unique_id`

The job sets the `user_source` property of users to LDAP when it updates or creates users in a repository. The `user_source` property is examined by Content Server to determine which authentication mechanism to use for authentication (unless a plug-in module is specified explicitly in the connection request). For LDAP users, `user_source` must be set to LDAP.

For users, the job sets the `user_login_domain` to the object name of the ldap config object representing the Content Server's primary LDAP directory server. For groups, the job sets the `group_directory_id` to the object name of the ldap config object.

The job also sets the `user_global_unique_id` property for users and the `group_global_unique_id` property for groups. In both cases, the property value is set to:

ldap_config_id:objectGUID

where *ldap_config_id* is the object ID of the ldap config object representing the LDAP server from which the user or group was synchronized, and *objectGUID* is the unique ID of the user or group object in the LDAP server.

On-demand user synchronization

Content Server supports on-demand user synchronization, a feature allowing directory users to be synchronized in the repository between scheduled synchronization jobs. On-demand user synchronization means that users added between job runs may still be allowed to connect to a repository because they are synchronized—added to the repository—on-demand, when they attempt to connect to the repository.

On-demand synchronization is controlled by the setting of the `dir_user_sync_on_demand` property of the `docbase` config object. When the `dir_user_sync_on_demand` property is `T` (`TRUE`) and a new unsynchronized LDAP user attempts to connect to the repository, the server searches for the user in the LDAP directory servers listed in the `ldap_config_id` and `extra_directory_config_id` properties of its server config object. (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.

User authentication

This section describes the implementation and behavior of user authentication when an LDAP directory server is used to authenticate users.

Authentication and federations

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 Content Server determines which server to use for authentication

A Content Server uses the following properties to authenticate an LDAP user:

- `dm_user.user_login_domain`
- `dm_server_config.ldap_config_id`
- `dm_server_config.extra_directory_config_id`

The values in those properties are used for authentication in the following manner:

1. Content Server examines the value of `user_login_domain`.

For LDAP users, this property is set to the object name of the LDAP directory server to be used for authenticating the user. This server is considered the primary LDAP server for that user's authentication.

Note: By default, if the user is not found in the repository, authentication fails. However, you can configure on-demand synchronization, so that users in the LDAP server that have not yet been synchronized (created in the repository) are synchronized when they attempt to connect to the repository. Refer to [On-demand user synchronization, page 328](#), for more information.

2. Using the object name, Content Server obtains the object ID of the ldap config object for that server.
3. Content Server then examines the values in `ldap_config_id` and `extra_directory_config_id` to determine if the object ID of the ldap config object is recorded in one of those properties.

- If the object ID is present, Content Server uses that LDAP server to authenticate the user.

Content Server makes a number of attempts to connect to the primary LDAP server. The value in the `retry_count` property of the ldap config object determines how many times Content Server attempts to contact the chosen LDAP server.

If these attempts fail and there are secondary, failover LDAP servers configured for the primary LDAP server, Content Server attempts to use one of those servers. If there are no failover LDAP servers configured or if the user cannot be authenticated using a failover server, user authentication fails. [Authentication failover, page 330](#), describes how authentication failover servers are specified and chosen.

- If the object ID is not present in either property, Content Server determines if any other authentication scheme can be used to authenticate the user.

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 332](#), contains instructions for setting up use of an LDAP directory server.

Connection retry attempts

Two ldap config properties control how many times Content Server attempts to contact LDAP server chosen as the primary server for user authentication. These properties are:

- `retry_count`
- `retry_interval`

The `retry_count` property defines how many attempts Content Server makes to contact the LDAP server if the first attempt fails. This property is set to 3 by default. You can change the default. However, if you set this to 0, Content Server does not make multiple attempts. Instead, Content Server immediately reports that it failed to contact the primary LDAP directory server if the first attempt fails.

The `retry_interval` property defines the interval between each attempt. This property is set to 2 for the interval between the first and second attempts. You can change this value. You can also override the value by setting two environment variables. The variables allow you to define different time intervals between each attempt. For instructions, refer to [Setting the `retry_interval` property for user authentication, page 343](#).

Authentication failover

You can configure secondary, failover LDAP servers for the primary LDAP servers used for authentication. Each primary LDAP server can have one or more failover LDAP

directory servers for authentication. However, an LDAP server created as a secondary server can only be designated as the failover server for one LDAP server. You cannot use the same LDAP server as a failover server for multiple LDAP servers. Content Server will use the failover LDAP servers if the LDAP server initially chosen for authentication is not available.

The failover behavior is controlled by two properties in the ldap config object. These two properties are:

- failover_ldap_config_ids
- failover_use_interval

The failover_ldap_config_ids property records the object IDs of the secondary LDAP servers that will serve as the failover servers for the LDAP server represented by the ldap config object.

The failover_use_interval property defines how long Content Server will use a failover server for authentication before attempting again to contact the primary LDAP server chosen for authentication.

If the binding attempts to the primary server fail, Content Server attempts to contact the secondary LDAP directory servers in the order that they are configured in failover_ldap_config_ids. After it binds to a secondary LDAP directory server, Content Server uses that server for any subsequent attempts to authenticate the user for the duration of time configured in the failover_use_interval property. When the specified time period has elapsed, Content Server attempts to contact the primary LDAP directory server for user authentication.

If the secondary LDAP directory server is unavailable or becomes unavailable within the specified use interval, Content Server contacts the remaining secondary LDAP directory servers. The servers are contacted in the order in which they are listed in failover_ldap_config_ids of the primary server. Content Server only contacts the secondary LDAP directory servers that are configured for the primary LDAP directory server; it will not contact the LDAP servers listed in the failover_ldap_config_ids property of the secondary servers.

If all attempts to contact the secondary LDAP directory servers fail, Content Server contacts the primary LDAP server again. If all LDAP servers are unavailable, user authentication fails.

Note: LDAP failover is supported for user authentication only. LDAP failover is not supported for user or group synchronization or on-demand user synchronization.

Failover for extra directory LDAP servers

When the user_login_domain or the user_global_unique_id points to one of the LDAP directory servers as configured in the extra_directory_config_id property, Content Server will use that LDAP directory server for authentication. If the extra LDAP directory server

is down, it checks for any secondary LDAP directory servers that might be configured for the extra LDAP directory server and attempts to contact them for user authentication.

Tracing failover

Failover details, including information on the switches between LDAP servers, are captured in the authentication trace in the Content Server log.

Implementing an LDAP directory server

This procedure creates an LDAP config object. If this is the first ldap config object configured for the Content Server, it also sets the server's ldap_config_id property in the server config to the object ID of that object. If it is not the first ldap config object created through that server, the object ID of the ldap config object is recorded in the extra_directory_config_ids property of the 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 and optionally, mapped properties for the repository.
Use the LDAP configuration facilities in Documentum Administrator. Refer to [Defining the set-up values, page 333](#), for information about the set-up values. [Mapping LDAP attributes to repository user or group properties, page 335](#), has information about defining mapped attributes. (Refer to the Documentum Administrator online help for instructions on using the user interface.)
You must be connected as a Superuser to perform this step.
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 341](#), to build and install an LDAP-enabled dm_check_password program.
 - b. Set the use_ext_auth_prog property 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 properties, page 335](#), 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 340](#), 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 340](#), 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 453](#), contains instructions on changing the schedule. [Activating the dm_LDAPSynchronization job, page 341](#), 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 334](#)
- [Search bases and filters, page 334](#)
- [The secure connection properties, page 335](#)

For information about mapping LDAP attributes to repository properties, refer to [Mapping LDAP attributes to repository user or group properties, page 335](#).

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\repository_name\ldap_ldapconfigid.cnt` (`$DOCUMENTUM/dba/config/repository_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` property 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` property is set to `bind_by_dn`, the user's distinguished name is obtained directly from the user's `user_ldap_dn` property. The `bind_type` property 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 properties available for searches. Some properties are indexed by default. However, there are three properties that you must explicitly index if you are using Oracle Intranet Directory as the LDAP directory:

- `createTimestamp`
- `modifyTimestamp`
- `operationTime`

The `createTimestamp` and `modifyTimestamp` properties must be indexed in order for the LDAP synchronization job to run correctly. The `operationTime` property must be indexed if you selected the Deactivate users option for the LDAP configuration. In addition to these properties, you can also index additional properties for use in a search filter. The *Oracle Intranet Directory Admin Guide* contains instructions on indexing properties.

For more information about search bases and the format of a filter definition, refer to the LDAP directory server's documentation.

The secure connection properties

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` property, 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, select Secure as the SSL mode. When you select Secure as the SSL mode, the interface enables you to edit the SSL port field.

SSL port, represented in the ldap config object by the `ssl_port` property, 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` property, identifies the location of the certificate database. The property 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.

Mapping LDAP attributes to repository user or group properties

LDAP directory servers enable you to define attribute values for user and group entries in the directory server. Content Server supports mapping those directory server values to user and group properties in the repository. Using mapping automates setting user and group properties.

Mappings between LDAP attributes and repository properties are defined when you create the ldap config object in Documentum Administrator. You can map the LDAP values to system or user-defined properties. You can map a single directory value to a repository property or you can map multiple directory values to a single repository property.

For example, you can map the LDAP attribute `homepage` to a custom property called `web_page`. A common use of the ability is to map the LDAP attributes `givenname` and `sn` (surname) to the `dm_user.user_name` property.

To map multiple LDAP properties to a single repository property, you use an expression. For example, the following expression uses the LDAP attributes `sn` and `givenname` to generate a `user_address` value:

```
${sn}_${givenname#1}@company.com
```

If the user's sn (surname) is Smith and the givenname is Patty, the expression above resolves to smith_p@company.com. The "1" at the end of givenname directs the system to only use the first letter of the given name.

Note: You can specify an integer at the end of an LDAP attribute name in an expression to denote that you want to include only a substring of that specified length in the resolved value. The integer must be preceded by a pound (#) sign. The substring is extracted from the value from the left to the right. For example, if the expression includes \${sn#5} and the surname is "Anderson", the extracted substring is "Ander".

Values of repository properties that are set through mappings to LDAP attributes may only be changed either through the LDAP entry or by a user with Superuser privileges.

Note: Changing mappings for the user_name, user_login_name, or group_name after the user or group is synchronized for the first time is not recommended. Doing so may cause inconsistencies in the repository.

Required mappings

Content Server requires three properties to be defined for a user and one property to be defined for a group. The mandatory properties are:

- user_name
- user_login_name
- user_address
- group_name

When you define an LDAP configuration object in Documentum Administrator, default mapped values are provided for these properties. You can change the defaults, but you must provide some value or mapping for these properties. Users cannot be saved to the repository without values for these three properties, nor can a group be saved to the repository without a group name.

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.

Defining mapping rules

For each mapped repository user or group property, you specify a mapping rule. The rule determines what is an acceptable mapped value for that property. If a mapped value fails to meet the criteria specified in the rule, the user or group is not created or

updated in the repository by the synchronization job. If there are multiple mapped properties for a user or group and the mapping for any property fails to satisfy its rule, the LDAPSynchronization job rejects the user or group entry and does not update or create the user or group in the repository.

The choices for the mapping rule are:

- Reject synchronization of the entry if the LDAP attribute is empty or has insufficient characters.
- Reject synchronization of the entry if the LDAP attribute is empty.
- Never reject the entry.

For the mandatory properties, the rule is set to reject entry synchronization if any LDAP attribute referenced in the property's mapping is empty or has an insufficient number of characters. You cannot change the rule for mandatory properties.

All rejected entries are captured in the LDAPSynchronizationDoc.txt file. You can find this file in %DOCUMENTUM%\dba\log\repository_id\sysadmin (\$DOCUMENTUM/dba/log/repository_id/sysadmin).

Mapping guidelines

In addition, the following rules apply when mapping LDAP group or user attributes to a repository property:

- In expressions, spaces are not required between references to LDAP attributes due to the bracket delimiter. If there is a space between mapped values, it will be appear in the result of the mapping.
- The length of the expression mapped to a single repository property cannot exceed 64 characters. Expressions that exceed 64 characters will be truncated.

The expression is recorded in the map_val property of the ldap config object. This property has a length of 64.

- All standard Documentum property lengths apply to the mappings. For example, the mapping string for user_name must resolve to 32 characters or less.

Repository storage of mappings

Mappings between LDAP attributes and repository properties are recorded in five properties of an ldap config object. The properties are repeating properties and the values across the properties at any particular index position represent the mapping for one repository property. These ldap config properties are:

- map_attr

This identifies the repository property that is the target of the mapping.

- `map_val`

This is the data that is mapped to the repository property. The data is one of: an LDAP attribute name, an expression referencing LDAP attributes, or an actual value.
- `map_attr_type`

This identifies the source of the property named in `map_attr`. The value is `dm_user`, a subtype of `dm_user`, or `dm_group`. Any user subtype identified in this property must also be identified in `user_subtype` of the `ldap` config object.
- `map_val_type`

This identifies the kind of data stored in `map_value`. It is one of `A`, meaning LDAP attribute; `E`, meaning expression; or `V`, meaning actual value.
- `map_rejection`

This records the rejection rule chosen for the mapping. Valid values are:

 - `0`, meaning synchronization always occurs even if an LDAP attribute referenced in the mapping is missing, empty, or does not have sufficient characters.
 - `1`, meaning that synchronization does not occur if an LDAP attribute in the mapping is missing or empty. However, synchronization will occur even if an LDAP attribute in the mapping has fewer characters than specified in the mapping.
 - `2`, meaning that synchronization does not occur if an LDAP attribute referenced in the mapping is missing, empty, or has fewer characters than specified in the mapping.

Mapping examples

This section provides some mapping examples that use expressions. In these examples, setting up some mappings for user properties is shown. If the example demonstrates an error, the reason the entry is rejected is explained. If the mapping is accepted and the entry is created, the example also shows how the values are recorded in the repository in the `ldap` config object. Assume that mapped entries are recorded in the `ldap` config object in the order in which the accepted examples are shown.

Example 9-1. A simple expression mapped to `default_folder`

Suppose you want to name the default folders for new users using the user's given name and surname. You can map the LDAP attributes `givenname` and `sn` to the default folder. The expression for that mapping is:

```
${givenname} ${sn}
```

In this case, the rejection rule is to reject the entry if either attribute is empty.

The values in the `ldap` config object properties are:

```
map_attr[0]=default_folder
map_val[0]={givenname} ${sn}
map_attr_type[0]=dm_user
map_val_type[0]=E
map_rejection[0]=1
```

For example, if a user entry in the LDAP server has a givenname of Patsy and a surname of Doe, then the value set in default_folder by the synchronization job is Patsy Doe.

Example 9-2. An expression with a string limitation

You also want to map the given name and surname to the user_name property. However, you only want to include the first 2 characters of the given name and you want the surname to appear first. The expression for that mapping is:

```
${sn}, ${givenname#2}
```

Because the user_name is a mandatory property, the rejection rule is to reject the entry if either attribute is empty or if either has insufficient characters.

The values in the ldap config object properties are:

```
map_attr[1]=user_name
map_val[1]={sn}, ${givenname#2}
map_attr_type[1]=dm_user
map_val_type[1]=E
map_rejection[1]=2
```

For a givenname of Patsy and an sn (surname) of Doe, the job sets the value in user_name to Doe, Pa.

If the user's givenname is less than 2 characters, the LDAPSynchronization job rejects the entry and the user is not created (or updated) in the repository.

Example 9-3. A more complex expression

Now, suppose you also want to create the user_address value by mapping the first initial of the given name and the full surname, separated by an underscore. The expression is:

```
${givenname#1}_ ${sn}@mycompany.com
```

Because the user_address is a mandatory property, the rejection rule is to reject the entry if either attribute is empty or if either has insufficient characters.

The values recorded in the ldap config object properties are:

```
map_attr[2]=user_address
map_val[2]={givenname#1}_ ${sn}@mycompany.com
map_attr_type[2]=dm_user
map_val_type[2]=E
map_rejection[2]=2
```

For our user Patsy Doe, the job sets user_address to: P_Doe@mycompany.com

Example 9-4. Mapping to a single attribute

Finally, you want to map the LDAP attribute cn (common name) to the custom property user_nickname. In this case, the rejection choice is to never reject the entry—some people

do not have nicknames and a nickname can be any length. There is no expression required for this mapping. The values recorded in the ldap config object for the mapping are:

```
map_attr[3]=user_nickname
map_val[3]=cn
map_attr_type[3]=dm_user
map_val_type[3]=A
map_rejection[3]=0
```

Patsy Doe has no nickname, but the job creates her user object in the repository anyway, as the rule is to never reject for this mapping. Her best friend Elizabeth Dingle, however, has a nickname, Betty, which is recorded in the LDAP attribute cn. When the job processes her LDAP entry, it sets Elizabeth's user_nickname to Betty.

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 cert.db file.
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 453](#), 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.

Note on using LDAP directory servers with multiple Content Servers

If multiple Content Servers are running against a particular repository, you must perform some additional steps to enable LDAP authentication regardless of the particular Content Server to which a user connects.

To enable LDAP authentication with multiple Content Servers:

1. Install the Content Server software and create a Content Server and repository.
2. Install the LDAP directory server and follow the directions in [Implementing an LDAP directory server, page 332](#) to properly configure the directory server and repository for LDAP authentication.
3. Create the nonprimary Content Servers.
4. Using Documentum Administrator, connect to one of the nonprimary Content Servers.
5. Navigate to the existing ldap config object.
6. Re-enter the Binding Name and Binding Password for the LDAP directory server.
7. Save the ldap config object.
8. Perform steps 4 to 7 for each nonprimary Content Server.

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` property 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.

Setting the `retry_interval` property for user authentication

The `retry_interval` property in an ldap config object defines the time interval between attempts by Content Server to contact the LDAP server represented by the ldap config object. This value is used when the LDAP server is chosen as the primary server for a user's authentication if the first attempt to contact the server fails. By default, the value is set to 5 seconds. So, if `retry_count` is set to 3 (its default), Content Server waits 5 seconds between each attempt to contact the primary LDAP server.

You can change the default. You can also override the property setting, to define different intervals between each attempt.

To override the property setting, you set two environment variables:

- `LDAP_RECONNECT_TIME_SECONDS`
- `LDAP_RECONNECT_INCREMENT_SECONDS`

How the environment variables are used

If you set the `LDAP_RECONNECT_TIME_SECONDS` and `LDAP_RECONNECT_INCREMENT_SECONDS`, the server uses them to calculate the length of time between attempts to connect to the primary LDAP server. The formula for the calculation is:

$$\text{LDAP_RECONNECT_TIME_SECONDS} + (\text{retry_no} * \text{LDAP_RECONNECT_INCREMENT_SECONDS}) = \text{interval length}$$

where `LDAP_RECONNECT_TIME_SECONDS` is the value in that variable; `retry_no` is the sequential number of the retry attempt, starting from 0 for the first attempt; and `LDAP_RECONNECT_INCREMENT_SECONDS` is the value in that environment variable.

For example, suppose that you set `LDAP_RECONNECT_TIME_SECONDS` to 3 and `LDAP_RECONNECT_INCREMENT_SECONDS` to 4, and that `retry_count` is set to 3. If the first attempt at connection fails, Content Server waits for 3 seconds before trying again:

$$3 + (0 * 4) = 3$$

If the second attempt fails, Content Server waits 7 seconds before attempting for a third time:

$$3 + (1 * 4) = 7$$

How to set the environment variables

On Windows, set the variables as system-level environment variables.

On UNIX, set the variables by adding the following lines to the `dm_start_repositoryname` script:

```
LDAP_RECONNECT_TIME_SECONDS=value export LDAP_RECONNECT_TIME_SECONDS
LDAP_RECONNECT_INCREMENT_SECONDS=value export
LDAP_RECONNECT_INCREMENT_SECONDS
```

where *value* is an integer representing a number of seconds.

You must restart Content Server to make the variable values take effect.

Enabling first-time synchronization rules

First-time synchronization matches non-directory users and groups in the repository to directory users and groups in the LDAP directory server, converts them to directory users and groups and updates them from the directory server. For users, a match occurs when the `user_name` or `user_login_name` matches an LDAP entry (after applying attribute mapping) and the value of `user_login_domain` is an asterisk (*), empty, or the name of the ldap config object.

To force the first-time synchronization rules to be used on a subsequent synchronization, set the `first_time_sync` property 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 <certdb_file_directory_path>
```

where *certdb_file_directory_path* is the directory path of the certx.db file, and *x* is the version number of the file.

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. If `method_trace_level` is set to a value greater than zero for the job, it also generates a log file for the job. Both the report and regular trace file can be easily accessed from Documentum Administrator. [Reports and trace log files, page 451](#), contains instructions on viewing job reports and log files and information about where they are stored in the repository.

If `dfc.tracing.enable` is set to T in the `dfc.properties` file, trace messages are also recorded in the DFC trace file.

Examining the ldap config object can also be useful when troubleshooting.

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.

Two plug-ins are provided with Content Server. One supports RSA. The other supports CA SiteMinder. [Using the RSA plug-in, page 348](#), describes how to use the RSA plug-in. [Using the CA SiteMinder plug-in, page 348](#), describes how to use the CA SiteMinder plug-in.

You can also write and install custom modules. [Implementing a custom authentication plug-in, page 349](#), 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` property 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 CA SiteMinder plug-in with a repository called `enr_db`, move the CA SiteMinder module to the `%DOCUMENTUM%\dba\auth\enr_db` (`$DOCUMENTUM/dba/auth/enr_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` property 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` property 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` property 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 require a password.

Using the RSA plug-in

The EMC RSA Plugin for Documentum allows Content Server to authenticate users based on RSA ClearTrust single sign-on tokens instead of passwords.

EMC Documentum Webtop and WebPublisher, and other Documentum WDK-based applications, support the RSA plug-in, with some configuration required. The configuration requirements are described in the *Web Development Kit and Client Application Development Guide*.

The Content Server installation procedure stores the plug-in in the %DM_HOME%\install\external_apps\authplugins\RSA (\$DM_HOME/install/external_apps/authplugins/RSA) directory. There is a README.txt file that describes how to install the plug-in. [Table 28, page 348](#), lists the files for the plug-in on the supported platforms.

Table 28. RSA plug-in modules

Platform	Plug-in module
Windows	dm_rsa.dll
Solaris, Linux, and AIX	dm_rsa.so
HP-UX PA-RISC	dm_rsa.sl
HP Itanium	dm_rsa.so

Using the CA SiteMinder plug-in

Documentum provides an authentication plug-in supporting authentication against a CA SiteMinder Policy Server. The plug-in enables validation of CA SiteMinder tokens sent to Content Server by Web-based clients.

Note: The CA SiteMinder plug-in is not supported on the HP Itanium platform.

EMC Documentum Webtop supports the CA SiteMinder Plug-in, with some configuration required. The configuration requirements are described in the *Web Development Kit Deployment Guide*.

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 29, page 349](#), lists the files for the plug-in on the supported platforms.

Table 29. CA SiteMinder plug-in files

Platform	File
Windows	dm_netegrity_auth.dll
Solaris, Linux, and AIX	dm_netegrity_auth.so
HP-UX on PA-RISC	dm_netegrity_auth.sl

The directory also includes the CA SiteMinder 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 property in users to the plug-in identifier. ([Identifying a plug-in for use, page 347](#), contains more information.)

The plug-in identifier for the CA SiteMinder 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 319](#), contains instructions.)



Caution: 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 350](#), contains instructions.
2. Install the plug-in.
[Plug-in scope, page 346](#), contains information about where to install an authentication plug-in.
3. Enable its use.

[Identifying a plug-in for use, page 347](#), 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` property of the user object must be set. Use Documentum Administrator, DFC (`IDfUser.setUserPassword`), 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 Documentum Desktop 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` property of the server config object to T.
2. Set the Use Windows Login option on the Desktop client interface for each repository for which you want to use unified login.

Refer to the Desktop online help for instructions.

For users on Webtop, you can configure single sign-on (SSO) on to provide functionality similar to unified logins.

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 30, page 352](#), lists the password files whose content is encrypted by default. All the files are found in `%DOCUMENTUM%\dba\config\repository_name` (`$DOCUMENTUM/dba/config/repository_name`). You must be the installation owner to access or edit these files.

Table 30. Password files encrypted by default

File	Description
<code>dbpasswd.txt</code>	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.)
<code>docbase_name.cnt</code>	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 <code>dm_operator.cnt</code> file is not.
<code>dm_operator.cnt</code>	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, <code>docbase_name.cnt</code> files are not used.

File	Description
federation.cnt	<p>Contains the information, including passwords, used by a governing repository server to connect to member repositories. The file is stored with the governing repository.</p> <p>The format of the file's content is:</p> <pre>member_repository_name:user_name:password:[domain]</pre>
ldap_object_id.cnt	<p>Contains the password used by Content Server to bind to an LDAP server.</p>

Using encryptPassword

Use `encryptPassword` to encrypt any password that you want to pass in encrypted form to the one of the following methods:

- `IDfSession.assume`
- `authenticateIn IDfSessionManager, IDfSession, or IDfClient`
- `IDfSession.changePpassword`
- A method to obtain a new or shared session
- `IDfPersistentObject.signoff`

Passwords encrypted with `encryptPassword` cannot be decrypted explicitly by an application or user. There is no method provided to perform decryption of passwords encrypted with `encryptPassword`. DFC decrypts those passwords internally when it encounters the password in the arguments of one of the above methods.

Passwords encrypted with `encryptPassword` are prefixed with `DM_ENCR_PASS`.

For complete information about using this method, refer to the Javadocs.

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 30, page 352, 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 30, page 352](#), 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]]  
-docbase 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:

- | | |
|--|---|
| <code>-location <i>AEK_location</i></code> | Identifies the location of the AEK file to be used to encrypt the password. If this argument is not set, the environment variable <code>DM_CRYPTPO_FILE</code> must be set. |
| <code>-passphrase <i>passphrase</i></code> | 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 <code>dm_crypto_boot</code> 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. |

<code>-docbase <i>repository_name</i></code>	Identifies the repository for which the password is being encrypted. Do not include this argument if you include the <code>-file</code> argument.
<code>-remote <i>remote_repository_name</i></code>	Identifies the file to operate on as <code>%DOCUMENTUM%\dba\config\<i>repository_name</i>\remote_repository_name</code> You must include the <code>-docbase</code> argument if you include <code>-remote</code> .
<code>-operator</code>	Identifies the file to operate on as <code>%DOCUMENTUM%\dba\config\<i>repository_name</i>\dm_operator.cnt</code> You must include the <code>-docbase</code> argument if you include <code>-operator</code> .
<code>-rdbms</code>	Identifies the file to operate on as <code>%DOCUMENTUM%\dba\config\<i>repository_name</i>\dbpasswd.txt</code> You must include the <code>-docbase</code> argument if you include <code>-rdbms</code> .
<code>-file <i>file_name</i></code>	Identifies the file on which to operate. Do not include this argument if you include the <code>-docbase</code> argument.
<code>-encrypt <i>password</i></code>	Defines the password to encrypt. If specified, the password is encrypted and written to the file identified in the <code>-file</code> 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 properties:

- max_auth_attempt in the docbase config object
- failed_auth_attempt in the user objects

The max_auth_attempt property 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 property to the maximum number of authentication failures you wish to allow.

The failed_auth_attempt property records how many authentication failures have occurred on behalf of a user. The property is set to 0 when a user is created and reset to 0 each time a user is successfully authenticated.

The failed_auth_attempt property 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 property exceeds the value in max_auth_attempt. The inactivation is recorded in the server log file. The failed_auth_attempt property is reset to 0 when a system administrator reactivates the user by setting the user_state property 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 property 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` property to -1 for the user.

Note: The property 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 *Content Server Fundamentals*. This chapter includes the following topics:

- [Overview of repository security, page 359](#)
- [Turning repository security on and off, page 362](#)
- [Turning folder security on and off, page 363](#)
- [Setting the default permission level for application-level control of SysObjects, page 364](#)
- [Object-level permissions, page 364](#)
- [Managing ACLs, page 368](#)
- [Table permits, page 386](#)
- [Auditing, page 387](#)
- [Implementing signature support, page 417](#)
- [Managing the encryption keys, page 429](#)
- [Managing the login ticket key, page 436](#)
- [Configuring a repository's trusted repositories, page 437](#)
- [Configuring login ticket use, page 437](#)
- [Configuring application access control token use, page 439](#)
- [Troubleshooting an application access control token, page 444](#)

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.

Note: Repositories are configured with security turned on by default.

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 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. (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 368](#).

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 364](#) in this manual and a more general description 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` property 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` property 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 member list as members of the group unless the application issues an `IDfSession.removeDynamicGroup` call. (For information about setting this property, refer to [Changing the membership setting of a dynamic group, page 310](#).)

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 property 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 363](#).

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 a repository and can be changed later if desired. User privileges are described in detail in [Chapter 8, Users and Groups](#).

Table permits

The table permits define access to RDBMS tables through Content Server. There are five levels of table permits, from none to delete. Table permits can be set for three user levels: owner, group, and world. Table permits are described in [Table permits, page 386](#).

Turning repository security on and off

The `security_mode` property in the docbase config object controls whether object-level security is imposed on the repository. This property has two possible settings, listed in [Table 31, page 362](#).

Table 31. 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. This is the default.

By default, the property is set to `acl`. If the security mode is `none`, ACLs are not enforced. Neither, by extension, is folder security. The security setting is recorded in the `security_mode` property of the `docbase` config object. Use DQL to change the security setting if desired.

Turning folder security on and off

Folder security is a supplemental level of repository security. It is turned on by default when the repository is installed, but only functions when repository security is also turned on. When folder security is turned on, the server performs the permission checks required by repository security 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 or Change Folder Links 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 or Change Folder Links permission on the target folder or cabinet.
- Unlinking an object from a folder or cabinet requires Write permission or Change Folder Links permission on the affected folder or cabinet.
- Moving an object using link and unlink methods requires Write permission or Change Folder Links 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 or Change Folder Links 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 or Change Folder Links permission on the primary folder of each version removed by a prune method.
- Copying an object using a `saveAsNew` method or with a drag and drop operation or a key sequence in the Documentum clients requires Write permission or Change Folder Links permission on all the folders or cabinets to which the new object will be linked. The new copies 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. A repository's folder security setting is recorded in the `folder_security` property in the `docbase` 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` property 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 property 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 basic permissions and six extended permissions.

Object-level permissions are defined in ACLs. (For information about setting permissions through ACL entries, refer to [Managing ACLs, page 368](#).)

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 properties. [Table 32, page 365](#), describes the seven access levels.

Table 32. Object-level permissions

Level	Permission	Description
1	None	No access is permitted.
2	Browse	The user can look at property 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 366 .
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 33, page 365](#), presents a simple overview of the operations allowed at each permission level.

Table 33. Permitted operations for object-level permissions

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

Content Server 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 34, page 366](#), describes the extended permissions.

Table 34. 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 descriptions of the link and unlink methods in the Javadocs.</p>
change_owner	The user can change the owner of the object.

Permission	Description
change_permit	The user can change the basic permissions of the object.
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 Procedure permission by default for that object.

Viewing extended permissions

Table 35, page 367, lists the computed properties that return information about the extended permissions.

Table 35. The extended permission computed properties

Property	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

Property	Single/ Repeating	Description
<code>_allow_change_owner</code>	S	A Boolean value indicating whether the user has the Change Ownership permission
<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 property 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 369](#)
- [How ACL entries are evaluated, page 374](#)
- [Disabling ACL restrictive entries, page 377](#)
- [External and internal ACLs, page 378](#)

- [System, public, and private ACLs, page 379](#)
- [Template ACLs, page 379](#)
- [Creating ACLs, page 380](#)
- [How ACLs and objects are connected, page 380](#)
- [The default ACLs, page 381](#)
- [Modifying an ACL, page 384](#)
- [Destroying an ACL, page 385](#)

The ACL object type

Access control lists are stored as persistent objects of type `dm_acl`. The single-valued properties of an ACL object contain information about the ACL, such as its name and its owner. The repeating properties 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 properties in an ACL. The values at one index position across the properties 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` property 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. 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`. Entries of type `RequiredGroup` cannot have an alias as the value for the `r_accessor_name`. A required group entry must have an actual group name for the `r_accessor_name` value.

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:RequirdGroupSet
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`. Entries of type `RequiredGroupSet` cannot have an alias as the value for the `r_accessor_name`. A required group set entry must have an actual group name for the `r_accessor_name` value.

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 a 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 algorithm described in this section if the owner is not a superuser. If the owner is also a superuser in the repository, the algorithm used is described in [Evaluating a Superuser's permissions, page 376](#).

To determine access permissions for an owner who is not a superuser, Content Server:

1. Checks that the owner belongs to any required groups or a required group set.
If the owner does not 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. If an entry attempts to restrict an owner to None, it is disregarded and the owner is given Browse permission by default.
 - 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.
Note: In addition to restrictions that may be imposed by `AccessRestriction` entries, the `owner_xpermit_default` key in the `server.ini` file may impose a restriction on the extended permissions accorded to an object owner. Refer to [owner_xpermit_default, page 101](#), for details.
 - 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` property to TRUE. When that property 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 property's setting. When Content Server is installed with a Trusted Content Services license, the property 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 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` property 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 381](#), 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 directing the server not to assign a default ACL, and then grants 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. (The *Documentum System Object Reference Manual* for all 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` property 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.

Note: An alias cannot be defined for the `r_accessor_name` value in ACL entries of type `RequiredGroup` or `RequiredGroupSet`.

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` property 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 380](#), 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` property to 1 and set the `r_accessor_name` values to aliases.

The *Content Server Fundamentals* has a complete description of aliases, the format of an alias specification, and a description of how the server resolves an alias.

How ACLs and objects are connected

Every object with an ACL has two properties that define which ACL is associated with that object. The two properties are:

- `acl_domain`

This property identifies 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 property 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 properties define the ACL that contains the access permission for the object. For user and type definitions, the properties identify the associated default USER and TYPE ACLs. (For types, these properties 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 the repository.)

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.

You can 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` properties.

- 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` properties.

- 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, at the least, you define an ACL for the `dm_sysobject` 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 property 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 property.

For details about using useACL, refer to *Content Server Fundamentals*. For information about setting the default_acl property, refer to [Identifying the default ACL for use](#), page 383.

Identifying the default ACL for use

If a user does not identify an ACL for a new object or does not explicitly indicate that the object should not have a default ACL, the server attempts to assign the object a default ACL. To identify which ACL to assign as the default, the server uses the value in the default_acl property of its server config object. This property 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 property 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` property 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. `Grant` methods add an entry to an ACL. `Revoke` methods remove entries for a particular user or group, or remove 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 DFC 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 DFC, use a `grant` method (`IDfSysObject.grant` or `grantPermit`). For the syntax and usage notes, refer to the Javadocs.

Removing entries

You can use Documentum Administrator or DFC to remove access control entries. You must own the ACL or have Superuser privileges to remove entries from an ACL.

If you use Documentum Administrator, refer to the online help for instructions. If you use DFC, use a `revoke` method (`IDfSysObject.revoke` or `revokePermit`). For the syntax and usage notes, refer to the Javadocs.

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 the `IDfPersistentObject.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.

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'
```

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 36, page 386](#).

Table 36. Table permits

Level	Permit	Description
0	None	No access is permitted
1	Select	The user can retrieve data from the table.
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.

Setting table permits

The table permits are defined for owner, group, and world access through properties of the registered object representing the registered table. (ACLs are not used.) The properties 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 properties.

You can set these properties at any time after you create the registered table. The properties have an integer datatype. When you set them, you define the integer value that corresponds to the permit level you want to assign. To assign more than one permit, you add together the integers representing the permits you want to assign and set the appropriate property to the total. For example, if you want to give the owner of a registered table the insert and update permissions, you would assign the value "6" to the `owner_table_permit` property of the `dm_registered` object representing the RDBMS table.

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 F, System Events](#).)

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.

Auditing properties

Whether property values are audited, and which properties are audited, is controlled by two factors:

- Whether you register properties for auditing when you start auditing for the event.

- Whether the `audit_old_values` property in the `docbase` config object is set to true or false.

The default for this property is T (true).

Note: On repositories created on RDBMS platforms with a page size of less than 8K, the `audit_old_values` property is always false and cannot be set to true.

Table 37, page 389, summarizes the behavior for each combination of these factors.

Table 37. Behaviors for audit properties combinations

	<code>audit_old_values=T</code>	<code>audit_old_values=F</code>
One or more properties are registered for auditing	Old and new values of the registered properties are recorded in the audit trail entries. If <code>audit_old_values</code> is T, page 389, describes this option in detail.	The values of the properties registered for auditing are recorded in the <code>attribute_list</code> property of the audit trail entry. If <code>audit_old_values</code> is F, page 390, describes this option in detail.
No properties are registered for auditing	Old and new values of any changed property, with the exception of internal (i_) properties, are recorded in the audit trail entry. If <code>audit_old_values</code> is T, page 389, describes this option in detail.	No property values are recorded in the audit trail entry.

If `audit_old_values` is T

The `audit_old_values` property of the `docbase` config object is true by default. When the property is true and there are no properties registered for auditing, by default, changes to properties are audited for objects of type `dm_document` and any subtype of the `dm_document` type. The audit trail records changes for all properties except those whose names begin with `i_` or `r_` prefixes. Custom properties are audited also.

Both old and new values are recorded in the audit trail by default. The property name and new value are recorded in the `attribute_list` property of the audit trail entry. The property name and old value are recorded in the `attribute_list_old` property of the audit trail entry. If the list of changed properties is longer than the length of the `attribute_list` and `attribute_list_old` properties, the overflow is recorded in an audit trail `attrs` object.

The audit trail entry records the values of the audited properties at the time the event occurred.

If you only want to audit specific properties, you can register those properties for auditing when you initiate auditing for the event. If you register specific properties for auditing, only changes to those properties are recorded in the audit trail. Any other property that changes is not recorded in the audit trail.

To illustrate how auditing specific properties behaves, suppose you have a `dm_document` subtype called `vendor_info` with a property named `vendor_number`, and you want to audit all changes in `vendor_number`. You also want to record both the current and old values of the `vendor_number`. First, ensure that `audit_old_values` is set to `true` in the `docbase` config object. Then, register the `dm_checkin` and `dm_save` events for auditing whenever a `vendor_info` object is the target of the `checkin` or `save` and identify the `vendor_number` as an audited property. Each time a user checks in a `vendor_info` document, Content Server creates an audit trail entry. In that entry, the server records the old `vendor_number` value in the `attribute_list_old` property and the new `vendor_number` value in the `attribute_list` property. For example, if the `vendor_number` is changed from 048125 to 04837S the entries in the audit trail are:

```
attribute_list = vendor_number="04837S"  
attribute_list_old = vendor_number="048125"
```

The `attribute_list` property records the new value of the property. When you specify properties for auditing only changed properties are audited. If the value of an audited property is not changed, its value is not recorded in the audit trail entry.

If multiple properties are audited, the `attribute_list` property records the new values of the properties at the time the event occurred and the `attribute_list_old` property records the previous values of those audited properties that changed. The audited properties and their values are recorded in a comma-separated list in each audit trail property. For example:

```
attribute_list=vendor_number="04837S",vendor_type="hardware",
```

For application events, the application is responsible for recording the names and values of audited properties in the audit trail entry.

If audit_old_values is F

By default, if `audit_old_values` is `F`, the audit trail entry does not record changes to properties unless you specifically register the properties for auditing when you register the event for auditing. If you register properties for auditing, then the properties are recorded in the `attribute_list` property of the audit trail entry. If the number of audited properties overflows the `attribute_list` property, the overflow is recorded in a `dmi_audittrail_attrs` object.

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.

Default auditing

Content Server audits the following events by default:

- All executions of methods that register or unregister events for auditing.
The event names are `dm_audit` and `dm_unaudit`.
- All executions of a `IDfPersistentObject.signoff` method
The event name is `dm_signoff`.
- All executions of an `addDigitalSignature` 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, register the event explicitly for auditing and set the argument to require signing.
- All executions, successful or unsuccessful of an `addESignature` method.
If the execution is successful, the event name is `dm_addesignature`. The audit trail entries for the `dm_addesignature` event are signed by Content Server automatically.
If the execution fails, the event name is `dm_addesignature_failed`
- 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

- A default set of events on objects of type `dm_document` and its subtypes. This default set of events is represented by the event named `dm_default_set`. This event is registered against the `docbase config` object, and Content Server audits the events in the set for `dm_document` type and its subtypes. [Table 38, page 392](#), lists the events that are included in this set.

Table 38. Events audited by the `dm_default_event`

Object type	Audited events		
<code>dm_docbase_config</code>	<code>dm_archive</code>	<code>dm_checkin</code>	<code>dm_restore</code>
	<code>dm_assemble</code>	<code>dm_checkout</code>	<code>dm_save</code>
	<code>dm_bp_attach</code>	<code>dm_destroy</code>	<code>dm_setfile</code>
	<code>dm_bp_demote</code>	<code>dm_freeze</code>	<code>dm_signoff</code>
	<code>dm_bp_promote</code>	<code>dm_link</code>	<code>dm_unfreeze</code>
	<code>dm_bp_resume</code>	<code>dm_lock</code>	<code>dm_unlink</code>
	<code>dm_bp_suspend</code>	<code>dm_mark</code>	<code>dm_unlock</code>
	<code>dm_branch</code>	<code>dm_prune</code>	

Turning off default auditing

With the exception of user login failure and the `dm_default_set` events, there are no default registry objects for the events audited by default, and consequently, you cannot turn off auditing for them. There are registry objects for user login failures and `dm_default_set`.

To turn off auditing of either login failures or `dm_default_set` events, you must have Config Audit permission. Use one of the methods in the `IDfAuditTrailManager` interface to remove the appropriate registry object.

For log in failures, the event name is `dm_logon_failure`.

Turning off auditing of the `dm_default_set` event for the `docbase config` type turns off auditing of all events represented by the `dm_default_set` event. It is not possible to turn off a subset of the events represented by `dm_default_set`. If you want to audit only some of those events, you must remove the registration for the `dm_default_set` event, and then add individual audit registrations for those events you do wish to audit. The individual registrations should name the specific object type as the target, rather than the `docbase config` type.

Modifying the default auditing

You can modify the default auditing by registering against the events audited by default. If you do, Content Server creates the audit trail entry based on the criteria you define for the event and target. Similarly, you can remove your registration. Content Server will return to creating the default audit trail entry for the event and target.

Auditing system events

Content Server recognizes a wide range of system events. System events are associated with DFC methods, lifecycles, workflows, and jobs. (Refer to [Appendix F, System Events](#), 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 393](#), describes how that is done.

To initiate auditing of a system event, you must have Config Audit privileges. You can use Documentum Administrator or a method in the IDfAuditTrailManager interface. Instructions for using Documentum Administrator to register for auditing are found in the Documentum Administrator online help. To use DFC, refer to the description in the IDfAuditTrailManager interface in the Javadocs.

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 416](#), 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 a method in the IDfAuditTrailManager interface 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 do not have to rewrite and recompile your application each time.

When the event occurs, use an `IDfAuditTrailManager.createAudit` method to create the audit trail entry. If you use `createAudit`, Content Server sets all the entry properties except the generic properties (`string_n` and `id_n`) and `attribute_list`.

If you wish to record values for audited properties, create the `audittrail` object explicitly, instead of using a `createAudit` method. If the total length of the property names and values that you want to record is greater than the size of the `attribute_list` property, create the `audittrail attrs` object also. If you require an `audittrail attrs` object for overflow, you must create and save that object before you save the `audittrail` object.

If you create the `audittrail` object directly and set and save its properties, Content Server sets only the `r_gen_source` property automatically. The `r_gen_source` property 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 `signAudit` argument when issuing a method to register for auditing to request a signature on audit trail entries for the event. This argument is `false` by default. Setting it to `true` directs Content Server to sign the audit trail entries generated by that audit registration.

Note: You cannot set the `signAudit` argument to `true` 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 property 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` property 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 415.

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 described), 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 `auditSubtypes` argument set to true and do not identify any properties in the `attributeList` argument. And, suppose you have also registered the `dm_save` event on `sop_doc`, a `dm_document` subtype and have identified three properties in the `attributeList` 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 properties for inclusion in the audit trail entry. The other defines a policy ID and defines the property 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 properties.

If you are unsure which registration was used to generate an audit trail entry, examine the `registry_id` property of the audit trail entry. The property records the object ID of the `dm_registry` object that generated the event.

Stopping auditing

To stop auditing a system event, use the methods in the `IDfAuditTrailManager` interface. You can also use these methods to destroy a registry object that represents an application event. Stopping auditing through Documentum Administrator or using DFC methods 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 information about using the `IDfAuditTrailManager` interface, refer to the Javadocs.

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` property. 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 properties of the audit trail entries are excluded from view for both system and application events:

- `acl_name`
- `acl_domain`
- `attribute_list`
- `attribute_list_old`
- `attribute_list_id`
- `chronicle_id`
- `object_name`
- `object_type`
- `owner_name`
- `session_id`
- `version_label`

Querying and retrieving audit trail entries

You can query and retrieve audit trail objects directly, using a DQL SELECT statement or DFC. 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 property values for a particular event, including any stored in a `dmi_audittrail_attrs` object, query the computed property, `_property_list_values`.

Interpreting audit trails of DFC method, workflow, and lifecycle events

Audit trail entries for DFC method, 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 properties in an audit trail object describe the event.

Audit trail properties with a common purpose

An audit trail object has many properties that are used by all system events and are available for use in user-defined audit trail entries. The properties that have a common purpose for entries generated by Content Server include all the properties other than the `string_x` and `id_x` properties. For example, the `user_id` property contains the `dm_user` object ID of the user who caused the event to happen.

Properties available for varied purposes

Each audit trail object has five ID-datatype properties and five string properties. These properties 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 properties to store information specific to the particular kinds of events. Some of the events generated by methods other than workflow or lifecycle-related methods use these generic properties and some do not. A user-defined event can use these properties to store any information wanted.

Use in audit trails for events generated by non-workflow or lifecycle methods

Table 39, page 398, describes how Content Server uses the generic string and id properties in audit trails generated by methods other than workflow or lifecycle methods. Only those events that use the generic properties are listed. If an event is not listed in this table, Content Server does not use the generic properties in audit trails generated by that event.

Table 39. Usage of generic properties by API events

Event	Generic property 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>
Addsignature (success)	<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>
Addsignature (failure)	<p>string_1, User who signed the object</p> <p>string_2, Error message indicating why the failure occurred</p> <p>string_3, The number of the signature, text indicating the reason for the audit entry, and the pre-signature hash argument. For example: <i>2/ENTRY_FOR_FAILED_ESIGN_OPERATION/pre-signature hash_argument</i></p>

Event	Generic property use
Addnote	id_1 , Object ID of document to which the note is attached
	id_2 , Object ID of the note
Addrendition	id_1 , Object ID of the content object
	string_1 , Format of the rendition
	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.
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

Event	Generic property use
Connect	<p>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</p> <p>string_4, Host name of the client machine from which the user connected</p> <p>string_5, IP address of the client machine from which the user connected</p>
Destroy	<p>Destroy uses the generic properties only when the object to be destroyed is a dm_audittrail object or a subtype of dm_audittrail. For details, see the Purge Audit entry in this table.</p>
Getcontent	See Getfile
Getfile	id_1 , Object ID of the content object for the content file
Getlogin	<p>id_1, Object ID of the user object representing the user identified in string_1.</p> <p>string_1, Name of the user for whom the ticket was requested</p>
Getpath	See Getfile
Insertcontent	See Setfile
Insertfile	See Setfile

Event	Generic property use
Insertpart	<p>id_1, Object ID of the virtual document to 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</p>
Kill	<p>id_1, Session ID of the terminated session</p>
Link	<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>
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>

Event	Generic property use
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>
Removenote	<p>id_1, Object ID of the object to which the note was attached</p> <p>id_2, Object ID of the note</p>
Removepart	<p>id_1, Object ID of the virtual document from which you are removing 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, If specified, the order_no that identifies the component to be removed</p>
Removerendition	<p>id_1, Object ID of the content object representing the rendition's content file</p> <p>string_1, Rendition format</p>
Removeretention	<p>id_1, Object ID of the retainer object</p>
Setcontent	See Setfile

Event	Generic property use
Setfile	<p>id_1, Object ID of the content object for the content file</p> <p>string_1, Name of the API</p> <p>string_2, Name of the file (unused for Appendcontent, Bindfile, Inserttcontent, Setcontent)</p> <p>string_3, Page number of the content file</p> <p>string_4, Format of the content file</p>
Setpath	See Setfile
Setoutput	<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_5, Output port name</p>
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>

Event	Generic property use
Unmark	string_1 , Name of the version label.
Updatepart	id_1 , Object ID of the virtual document into 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, if specified string_5 , The order_no if specified

Use in lifecycle audit trails

[Table 40, page 404](#), describes how Content Server uses the generic string and id properties in audit trails generated by lifecycle events.

Table 40. Usage of generic properties by lifecycle events

Event	Generic property 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 properties.
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

Event	Generic property use
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 properties.
Validate	string_1 , Number of states in the business policy.

Use in workflow audit trails

Table 41, page 405, describes how Content Server uses the generic string and id properties in audit trails generated by workflow events.

Table 41. Usage of generic properties by workflow events

Event	Generic property use
Abort workflow	id_1 , Object ID of the workflow
Add attachment	id_1 , Object ID of the workflow id_5 , Object ID of the attached object string_5 , Name of the attached object
Add note	id_1 , Object ID of the workflow id_5 , Object ID of the note object string_1 , Activity sequence number string_2 , Activity name string_5 , Value of the note's keep_permanent flag

Event	Generic property use
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, Names of the components identified in the componentIds argument. This is only set if 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>
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 dm_cond_expr 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>

Event	Generic property use
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> property 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> property 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>
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>

Event	Generic property use
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> property.)</p> <p>string_1, Activity sequence number</p> <p>string_2, Activity name</p> <p>string_3, Comma-separated list of work item properties and their values. The properties 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> property 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>
Delegated 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, Activity type (Manual or Automatic)</p> <p>string_4, Name of the workqueue, if any, to which the task is assigned</p> <p>string_5, Name of the user to which the task is delegated</p>
Finish workflow	id_1 , Object ID of the workflow
Install workflow or activity definition	Does not use the generic properties.

Event	Generic property use
Invalidate workflow or activity definition	Does not use the generic properties.
Pause 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>
Port select	<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, 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.</p>
Pseudo-completed work item	<p>id_1, Object ID of the workflow</p> <p>id_2, Object ID of the work item</p> <p>string_1, Sequence number of the activity</p> <p>string_2, Name of the activity</p> <p>string_3, State of the work item prior to pseudo-completion</p> <p>string_5, Name of the task owner</p>
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>

Event	Generic property use
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>
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>

Event	Generic property use
Save a workqueue	<p>id_1, Object ID of the workqueue</p> <p>id_2, Value of the <code>wq_policy_id</code> property of the workqueue object</p> <p>string_1, Name of the workqueue</p> <p>string_2, Number of users in the workqueue</p>
Save a workqueue policy	<p>id_1, Object ID of the workqueue policy object</p> <p>string_1, Name of the workqueue policy</p> <p>string_2, Maximum threshold of the policy</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> <p>string_4, Name of the workqueue, if any, to which the task is assigned</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 <code>id_5</code>.</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>

Event	Generic property use
Started 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 activities, the object ID of the <code>dm_method</code> object for the method executed by the activity. For manual activities, this is null.</p> <p>string_1, Activity sequence number</p> <p>string_2, Activity name</p> <p>string_3, Comma-separated list of work item properties and their values. The properties are: <code>r_priority</code>, <code>a_wq_name</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_4, Name of the performer of the work item</p> <p>string_5, Value in the <code>dependency_type</code> property of the corresponding queue item object.</p>
Start workflow	id_1 , Object ID of the workflow
Suspend workqueue task	<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_4, Workqueue name</p>
Uninstall workflow or activity definition	Does not use the generic properties.

Event	Generic property use
Unsuspend workqueue task	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_4 , Workqueue name
Validate workflow or activity definition	Does not use the generic properties.

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 properties defined for the `dm_audittrail_acl` object type store information about an audited ACL. The properties 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 property 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 property 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` property records the event description, in this case, `Create`, indicating the creation of an ACL. The `accessor_operation` property describes what operation was performed on each accessor identified at the corresponding index position in `accessor_name`. The `accessor_permit` and `accessor_xpermit` properties record the permissions assigned to the user (or group) identified in the corresponding positions in `accessor_name`. Finally, the `is_group` property 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 properties:

```
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
```

```

is_group      [1]:0
              [2]:0
              [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 properties 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 property defines what operation was performed on the member at the corresponding names property.

Verifying signed audit trail entries

A signed audit trail entry is an entry that has a value in the audit_signature property. 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 IDfAuditTrailManager.verifyAudit method. This method regenerates the signature and compares the newly generated signature with the signature stored in the audit_signature property. 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` property 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 456. 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 the reference description 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 multi-repository 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 417](#). Electronic signatures are not supported on the Linux and HP Itanium platforms.

- Digital signatures using addDigitalSignature

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 427](#).

- 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 428](#).

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.

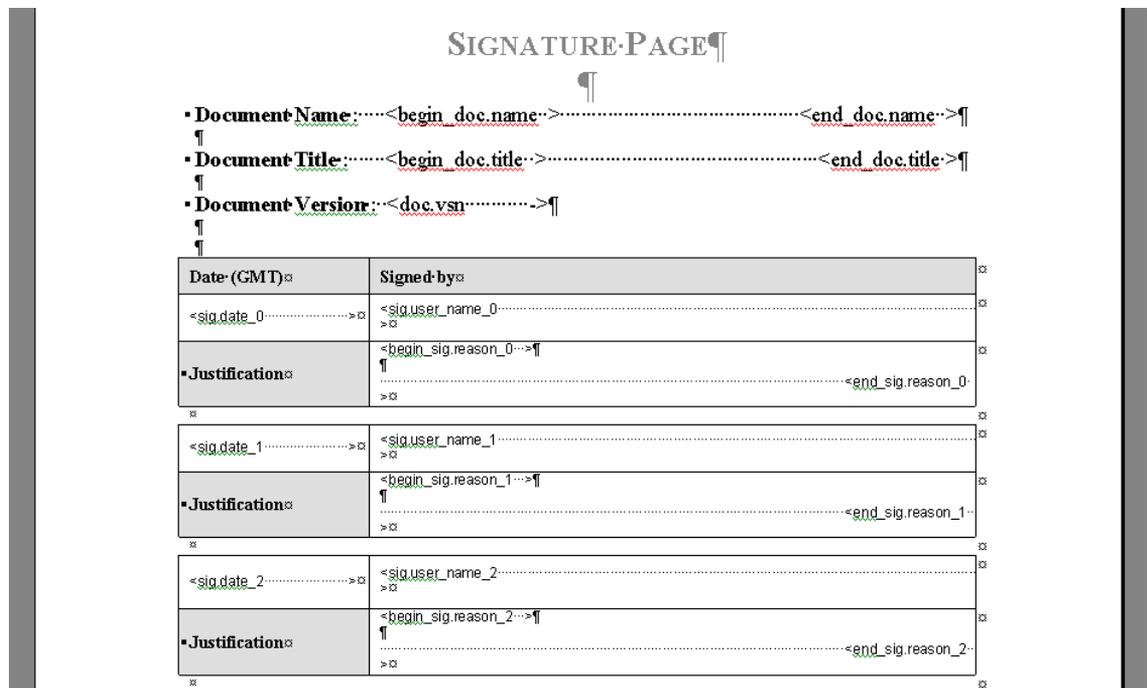
For instructions on customizing the default functionality, refer to [Customizing the default functionality, page 418](#).

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 425](#), 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. [Figure 9, page 418](#), shows a portion of the default signature page template.

Figure 9. Default signature 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 property, and <doc.vsn> is replaced with the object’s numerical version label. Some of the properties are values of properties 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 properties 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 42, page 419](#), lists the system-defined properties associated with documents, and [Table 43, page 420](#), lists those associated with signatures. In addition, you can add custom properties. For instructions, refer to [Adding or removing properties on the page, page 420](#).

Table 42. 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

Property	Simple/textbox	Replaced with
doc.modifier	Simple	Name of the last person to have modified the object being signed (the value of the r_modifier property)
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 43. 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 an 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 422.](#))
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.

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 properties 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 property to the desired delimiter for the beginning of the property placeholders.

3. Set the end_tag property 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 property of the rendition to dm_sig_template.

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 carot 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 property 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 property of the template. A signature page template is used to generate signature pages for the object type (and its subtypes) defined in this property. For example, the property 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` property 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 properties of the template object.

The number of signatures allowed is controlled by the `max_signatures` property. 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 will not 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` property of the template object. By default, this property is T (TRUE), meaning to append the signature page to the end of the content. Setting this property to F (FALSE) causes the signature creation method to prepend the signature page to the beginning of the content.

You can set either property using DFC 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



Caution: 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 44, page 425](#), to the method.

Table 44. 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.

Parameter	Description
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...n • sig.no_0...n • sig.user_id_0...n • sig_user_name_0...n • sig.reason_0...n • sig.date_0...n • sig.utc_date_0...n <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 property 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 136](#), 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).

The tracing information for the `addESignature` and `verifyESignature` 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.

You can also set the `DM_DEBUG_ESIGN_METHOD` environment variable. When that is set to 1, Content Server logs additional information about electronic signatures generated by `addESignature` to the repository log file. You must restart Content Server after setting this variable.

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. The only system administration task is registering the `dm_addsignature` event for signature by Content Server. This is an optional task. When an application issues the `addDigitalSignature` 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 registration must be issued for the `dm_addsignature` 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 394](#). *Content Server Fundamentals* has a description of how applications implement the use of digital signatures.

Customizing simple signoffs

Simple signoffs are implemented using a `IDfPersistentObject.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.

- Use an argument in a registration method to force Content Server to sign the `dm_signoff` events or to add additional information to the entries

Note: 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 property in the server config object to point to the location object created in Step 2.

The signature_chk_loc property 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
"audited_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. Using a custom passphrase is recommended. [Changing a passphrase, page 434](#), has instructions for changing the passphrase.

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 434](#), for instructions.
3. Run the `dm_crypto_boot` utility using the `-all` argument.
Refer to [Using dm_crypto_boot, page 432](#), 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` property of the `docbase` config object. This property 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 uses 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 432](#), 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 433](#), 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 434](#), 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 do not 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.key` (`$DOCUMENTUM/dba/secure/aekey.key`).

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 do not 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 do not 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 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 45, page 435](#).

Table 45. 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\sek.key
-newpassphrase custom_pass1 -noprompt
```

This next example changes the passphrase from one custom passphrase to another:

```
dm_crypto_create -location %DOCUMENTUM%\dba\secure\sek.key
-passphrase genuine -newpassphrase glorious
```

This final example changes the passphrase from a custom passphrase to the default:

```
dm_crypto_create -location %DOCUMENTUM%\dba\secure\sek.key
-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.

Note: *Content Server Fundamentals* describes login tickets, application access control tokens, and how each are used.

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. To import the key, use `IMPORT_TICKET_KEY`. Use Documentum Administrator to execute these methods. These methods are also available as DFC methods (`exportTicketKey` and `importTicketKey`) in the `IDfSession` interface.

You must restart Content Server after importing a login ticket key to make the new login ticket key take effect.

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.

You must restart Content Server after resetting a login ticket key.

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` property (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` property is set to `FALSE`, and you must provide the names of the trusted repositories. Those names are recorded in the `trusted_docbases` property 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 *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` property 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` property. The value of that property is set to 5 minutes by default when a repository is installed.

Neither the property value nor the argument value can exceed the maximum interval defined in the `max_login_ticket_timeout` property defined in the server's server config object. This property 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` properties, 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.

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` property 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` property of the doabase config object.

Troubleshooting a login ticket

To troubleshoot a login ticket, use the `IDfSession.getLoginTicketDiagnostics` method. This method returns details about the ticket extracted from the login ticket.

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 *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` property of its server config object.

Enabling machine-only AAC tokens

An application access control token can be created to be valid only when sent from a particular host machine. This token authentication mechanism is dependent on knowledge of the host's machine ID. If you are using such tokens, you must set the `dfc.machine.id` key in the `dfc.properties` file used by client applications on that host so that DFC can include that when sending the application token to Content Server. Set the key to the host's machine ID.

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`.
For instructions on using the utility, refer to [Generating tokens for storage, page 441](#).
2. Configure the retrieval behavior in the `dfc.properties` file.
 - a. Set `dfc.tokenstorage.enable` to `true`.
 - b. Set `dfc.tokenstorage.dir` to the desired token storage directory.

The `dfc.tokenstorage.enable` key is a Boolean key that controls whether the client library can retrieve tokens from storage. If it is set to `true`, 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 `false`, the client library does not retrieve tokens from storage.

The `dfc.tokenstorage.dir` 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 DFC on a client host machine for the first time, the installation sets the `dfc.tokenstorage.enable` key to `false` and the `dfc.tokenstorage.dir` to `<user-selected drive>:\Documentum\apptoken`. If you upgrade or reinstall 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 DFC. The DFC installation process sets the `dfc.tokenstorage.enable` key in the `dfc.properties` file on the server host to `false` and the `dfc.tokenstorage.dir` to `%Documentum%\apptoken` (`$DOCUMENTUM/apptoken`). However, when the Content Server installer runs, it resets the `dfc.tokenstorage.enable` key to `true`. The `dfc.tokenstorage.dir` is not reset.

The `dfc.properties` file on the server host is used by the internal methods to connect to repositories. The `dfc.tokenstorage.enable` and `dfc.tokenstorage.dir` settings in this

dfc.properties file affect methods associated with dm_method objects. Replication and federation methods are not affected by the settings in this dfc.properties file because these jobs execute as a Superuser.

Note: *Content Server Fundamentals* has additional information about how internal methods are affected by this setting.

If you are installing an upgrade to Content Server or a previous DFC installation occurred and a dfc.properties 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. The 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 46, page 441, lists the arguments accepted by this utility.

Table 46. 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.

Argument	Description
<i>-b repository_name</i>	Name of the repository to which to connect to create the token. This is a required argument.
<i>-a 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.
<i>-s 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 login ticket key as the repository in which the token was generated. This is an optional argument. The default is <code>global</code> .
<i>-t 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.
<i>-m machine_only</i>	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. This is an optional argument. The default is F.
<i>-o output_file</i>	The name of the generated XML token file. You can specify a full file path or only the file name. The file name must be either: <code>default.tkn</code> or

Argument	Description
	<p><code>repository_name.tkn</code></p> <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 443, 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 connection request. 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 `dfc.tokenstorage.dir` key in the client's `dfc.properties` 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 `dfc.tokenstorage.dir` 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 `dfc.tokenstorage.dir` key is set automatically when the DFC is installed. For more information about its setting, refer to [Enabling token retrieval by the client library, page 440](#).

Troubleshooting an application access control token

To troubleshoot a login ticket, use the `IDfSession.getApplicationTokenDiagnostics` method. This method returns details about the token extracted from the application access control token.

Administration Tools

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 446
- [Activating and scheduling administration tools](#), page 453
- [Running administration jobs on demand](#), page 454
- [Archive](#) , page 454
- [Audit Management](#) , page 456
- [Consistency Checker](#), page 459
- [Content Replication](#), page 465
- [Content Warning](#) , page 468
- [Create Full-Text Events](#), page 470
- [Data Dictionary Publisher](#), page 475
- [Database Space Warning](#) , page 476
- [Dm_LDAPsynchronization](#), page 479
- [Dmclean](#) , page 482
- [Dmfilescan](#) , page 487
- [File Report](#) , page 493
- [Group Rename](#), page 498
- [Index Agent Startup](#), page 498
- [Log Purge](#) , page 499
- [Queue Management](#) , page 503
- [Remove expired retention objects](#), page 506
- [Rendition Manager](#) , page 509
- [State of the Repository Report](#) , page 514
- [Swap Info](#) , page 518
- [ToolSetup](#), page 519

- [Update Statistics](#) , page 519
- [User Chg Home Db](#), page 522
- [User Rename](#), page 523
- [Version Management](#) , page 525
- [Tool maintenance and troubleshooting](#), page 529

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. Each tool identifies a job to run on a specific schedule.

The tools are described briefly in [Table 47, page 446](#). 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 47. 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 file stores
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

Tool	Description
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.
Data Dictionary Publisher	Publishes data dictionary information to make it visible to users and applications
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 a 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.
Group Rename	Renames a group in the repository. This is executed when you change a group name through Documentum Administrator.

Tool	Description
Index Agent Startup	Starts the index agent. This is not available for manual execution. It is called automatically whenever Content Server is started.
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
Remove Expired Retention Objects	Removes objects from the repository whose content, stored in an EMC Centera storage area, has an expired retention date.
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
Update Stats	Updates stored statistics on the underlying RDBMS tables, to aid in query performance.
User Chg Home Db	Changes a user's home repository. This tool is executed through Documentum Administrator, when changing a user's home repository. You may not execute the tool directly.
User Rename	Changes a user's name in the repository. This is executed when you change a user name through Documentum Administrator.
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 properties that define an execution schedule for the method. Refer to [Chapter 4, Methods and Jobs](#), for more information about jobs and methods.

Jobs are automatically executed on the schedule defined by their properties. 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 453](#) for instructions on setting job schedules. Information about running tools on demand is in [Running administration jobs on demand, page 454](#).)

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 48, page 449](#), to the job's method.

Table 48. 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 property 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` property), the Inbox and email notifications are sent to the user identified in the `operator_name` property of the server's server config object. (That property 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` property). 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
- Dmfilesan
- LogPurge
- SwapInfo

All 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. Job reports are 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. Trace logs file for jobs are created whenever the `method_trace_level` argument in the job is set to any value except 0. A trace level of 0 turns off tracing. The default trace level for all jobs in the tool suite is 0.

To view a trace log file, use Documentum Administrator.

Note: If the `dfc.properties` file used by the Content Server has a trace file location specified, 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.

Storage

In the repository, job log files are stored in `/Temp/Jobs/tool_nameTrace`. 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 repository 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 150](#). Refer to [Introducing job sequences, page 146](#) 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` property 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 450](#) 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 277](#).

The Archive tool is active by default, and runs once daily.

Arguments

[Table 49, page 454](#) describes the arguments for the tool

Table 49. 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.

Argument	Datatype	Default	Description
<i>-Username</i>	string(64)	-	Identifies the user executing dmarchive. If unspecified, the current user is assumed.
<i>-Ppassword</i>	string(64)	-	Password for the user executing dmarchive. If unspecified, the user is prompted.
<i>-archive_dir directory</i>	string	-	The location of the archive directory. Use a full path specification.
<i>-queue_name name</i>	string	-	Identifies the repository operator. Specify the operator's repository user name. If unspecified, the tool assumes the user named in the <code>operator_name</code> property of the server config object.
<i>-queue_event 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.
<i>-do_archive_events</i>	Boolean	-	TRUE directs the tool to process only archive events.
<i>-do_restore_events</i>	Boolean	-	TRUE directs the tool to process only restore events.
<i>-verbose</i>	Boolean	T	TRUE directs the tool to print trace messages.
<i>-window_interval</i>	integer	720	Defines window in which the tool can run. Value is interpreted in minutes.
<i>-queueperson</i>	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 runs under the Documentum installation owner's account to execute 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. All executions of the tool are audited. The generated audit trail entry has the event name `dm_purgeaudit`.

The `cutoff_days` and `custom_predicate` arguments determine which audit trail objects to remove. 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

Table 50, page 457, lists the arguments to the Audit Management tool.

Table 50. 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>	string(32)	-	User who receives email and inbox notifications from the tool. The default is the user specified in the <code>operator_name</code> property of the server config object.
<code>-custom_predicate</code> <i>qualification</i>	string	-	A WHERE clause <i>qualification</i> for the query that selects audit trail entries for deletion. The qualification must be a valid qualification and can reference only audit trail object type properties. For example, a valid qualification is <code>event='approved'</code> or <code>name='dmsadmin'</code> . (Refer to the general discussion of the

Argument	Datatype	Default	Description
			tool, above, for details of setting this argument.)
			Refer to the <i>Content Server DQL Reference Manual</i> for complete information about WHERE clause qualifications.
-cutoff_days	<i>integer</i>	90	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.
			To delete all audit trail objects, set this value to zero (0).

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 nonexistent 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 property
positions
Checking for types with invalid property 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 do not 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 propertys 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 property
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.

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` property identifies the local server performing the replication using the format `repository.serverconfig@hostname`.

The `ContentReplication` job is inactive by default.

Arguments

[Table 51, page 467](#), describes the arguments for the tool.

Table 51. Content Replication arguments

Argument	Datatype	Default	Description
-window_interval	integer	120	Defines window in which the tool can run. Value is interpreted in minutes.
-queueperson	string(32)	-	User who receives email and inbox notifications from the tool. The default is the user specified in the operator_name property 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 property. 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

Table 52, page 469, describes the arguments for the tool.

Table 52. 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 property of the server config object.
-window_interval	integer	720	Execution window for the tool, expressed in minutes. Refer to The window interval, page 450 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 generates events for unindexed objects in a repository. The Create Full-Text Events tool (`dm_FTCreateEvents`) may be used in two ways:

- To complete an upgrade by generating events for any objects missed by the pre-upgrade indexing operations

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_FTCreateEvents` 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 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. If you run the tool to reindex the repository, schedule the job to run daily and do not change the default batch size of 50,000 items.

The tool itself does not update the index; it only generates the events that will be used to 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 System Installation and Administration Guide* 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 or later repository with `full_reindex` set to `TRUE`, you must create a high-water-mark queue item (`dmi_queue_item`) manually and then specify the `r_object_id` of the queue item as the `-high_water_mark_id` argument of the Create Full-Text Events 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

[Table 53, page 471](#). describes the tool's arguments. 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 53. Create Full-Text Events arguments

Argument	Datatype	Default	Description
<code>-max_events_per_run</code>	integer	50000	The maximum number of events generated each time the job runs. If <code>max_events_per_run</code> is not set or is set to zero, the job creates

Argument	Datatype	Default	Description
-high_water_ mark_id		-	<p>events for all objects changed or created between min_date and max_date.</p> <p>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."</p> <p>In full reindex mode, you must provide the r_object_id of the manually-created high-water-mark queue item.</p>
-min_date	Date	When not in full reindex mode, the value of the date_sent property of the migration-mode queue item for indexing.	The earliest date on which objects were modified for which the job creates events.

Argument	Datatype	Default	Description
-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: <pre>-file full_path_of_file</pre>
-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 objects in reverse chronological order by the value of the r_modify_date.

Argument	Datatype	Default	Description
-current_only	Boolean	FALSE	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 property of the server config object.
-window_interval	integer	12000	Execution window for the tool, expressed in minutes. Refer to The window interval, page 450 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 properties 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 properties store the data dictionary information. Client applications that use the data dictionary information can reference or query these objects and their properties. (For more information about the data dictionary, refer to *Content Server Fundamentals*.)

Data Dictionary Publisher generates a status report that is saved in the repository in `/System/Sysadmin/Reports/DataDictionaryPublisher`.

Arguments

[Table 54, page 475](#), describes the tool's argument.

Table 54. Data Dictionary Publisher argument

Argument	Datatype	Default	Description
<code>-window_interval</code>	integer	720	Execution window for the tool, expressed in minutes. Refer to The window interval, page 450 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

[Table 55, page 477](#), lists the tool's arguments.

Table 55. 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 RDMBS table may have before being reported as fragmented.
<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> property of the server config object.
<code>-window_interval</code>	integer	720	Execution window for the tool, expressed in minutes. Refer to The window interval, page 450 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: <dmdadmin@bigcat>
X-UIDL: 827349620.001
Date: Wed, 20 Mar 1996 11:18:54 -0800
From: dmdadmin@bigcat (Documentum 2.0)
To: stevex@tiger
Subject: Event FragedTables has occurred
on DBWarning.Doc
(090000018006ee33) by dm20
```

```
DOCBASE:      test1
EVENT:        FragedTables
NAME:         DBWarning.Doc
SENT BY:      dmdadmin
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 directory server that have occurred since the last execution of the tool and propagates those changes to the repository. If necessary, the tool creates default folders and groups for new users. If there are mapped user or group properties, those are also set.

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 inactivation operation. Instructions for enabling the changelog feature are found in the vendor's iPlanet Administration Guide.

The behavior of the tool is determined by the property settings of the dm_ldap_config object. The tool has four arguments that you can use to override the property settings controlling which operations the tool performs. These are listed in [Table 56, page 480](#).

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 333](#).

Arguments

[Table 56, page 480](#), describes the arguments for the tool.

Table 56. 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 property 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 property in the ldap config object.
-queueperson	string(32)	-	User who receives email and inbox notifications from the tool. The default is the user specified in the operator_name property of the server config object.

Argument	Datatype	Default	Description
-rename_group_option	Boolean	FALSE	<p>Set to TRUE, directs the job to rename groups in the repository if their names have changed in the directory server.</p> <p>Setting this overrides the rename_group_option property in the ldap config object.</p>
-rename_user_option	Boolean	FALSE	<p>Set to TRUE, directs the job to rename users in the repository if their names have changed in the directory server.</p> <p>Setting this overrides the rename_user_option property in the ldap config object.</p>
-source_directory		dm_all_directories	<p>Controls which LDAP servers are synchronized when the job runs.</p> <p>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.</p> <p>Explicitly specifying LDAP servers in -source_directory, page 482, describes how specify multiple servers individually.</p>
-window_interval	integer	1440	<p>Execution window for the tool. Refer to The window interval, page 450 for a complete description.</p>

Executing dm_LDAPsynchronization manually

You can execute the dm_LDAPsynchronization tool manually from the command line. The syntax is

```
java com.documentum.ldap.LDAPSync -docbase_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.

Explicitly specifying LDAP servers in -source_directory

Use the LDAP server's object name to identify the server in the -source_directory argument. If you want to identify multiple servers, use the following syntax:

```
ldap_config_obj_name+ldap_config_obj_name{+ldap_config_obj_name}
```

where *ldap_config_object_name* is the object name value of the ldap config object for the LDAP directory server. For example:

```
ldap_engr1+ldap_engr2+ldapQA
```

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 a 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 57, page 483, describes the arguments for the tool.

Table 57. 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.

Argument	Datatype	Default	Description
-clean_now	Boolean	TRUE	Controls whether the tool removes the orphaned objects. The dmclean utility generates a 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.
-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 EMC Centera storage areas in the operation. T means that expired, orphaned content in EMC Centera 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 property of the server config object.
-window_interval	integer	120	Execution window for the tool. Refer to The window interval, page 450 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 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 270](#).)

Dmfilescan also generates a status report that lists the files it has removed. The report is saved in the repository in /System/Sysadmin/Reports/DMFilescan.

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/DMFilescanLondon.

Dmfilescan is installed in the inactive state.

Arguments

Table 58, page 488, lists the arguments for the tool. Refer to the description of the dmfilescan utility in [Using dmfilescan, page 270](#), for instructions on specifying values for the -from and -to arguments.

Table 58. Dmfilescan arguments

Argument	Datatype	Default	Description
-s <i>storage_name</i>	string	-	Specifies a target storage area. If this argument is not included, all storage areas are scanned.
-from <i>directory_path</i>	string	-	Starting subdirectory for the scan operation. Refer to Identifying the subdirectories of the scanned storage areas, page 271 for information about using this argument.
-to <i>directory_path</i>	string	-	Ending subdirectory for the scan operation. Refer to Identifying the subdirectories of the scanned storage areas, page 271 for information about using this argument.

Argument	Datatype	Default	Description
-scan_now	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.
-queueperson	string(32)	-	User who receives email and inbox notifications from the tool. The default is the user specified in the operator_name property of the server config object.
-window_interval	integer	120	Execution window for the tool. Refer to The window interval, page 450 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 272 for details of use.
-grace_period	integer	168	Defines the grace period for allowing orphaned content files to remain in the repository. The

Argument	Datatype	Default	Description
			<p>default is 168 hours (1 week), expressed as hours. The job removes orphaned files whose age exceeds 1 week or the value defined in the <code>-grace_period</code> argument.</p> <p>The integer value for this argument is interpreted as hours.</p>

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 `Dmfilesan` report.

```
DMFilesan Report For DocBase boston2
As Of 9/17/96 11:08:54 AM
Generated DMFilesan script will be executed...
The trace level is set to 5...
DMFilesan utility syntax: apply,c,NULL,DO_METHOD,
METHOD,S,dmfilesan
Executing DMFilesan...
Executing DMFilesan script...
sh /u106/dm/dmadmin/dba/log/00000962/sysadmin/
0900096280012800.bat
>/u106/dm/dmadmin/dba/log/00000962/sysadmin/
0900096280012800.txt
Generated script from the DMFilesan method:
```

```
----- Start
/u106/dm/dmadmin/dba/log/00000962/sysadmin/
0900096280012800.bat output
-----
#!/bin/sh -x
#
# Documentum, Inc.
#
# This script is generated by dmfilesfan 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 494](#), 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 494](#).

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 biweekly).

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 properties and arguments as needed. Provide a new name for the copy that identifies it meaningfully.

[Creating jobs and job sequences, page 145](#), provides instructions for creating new jobs, and the *Documentum System Object Reference Manual* contains a description of the `dm_job` object's properties. You can create or copy a job using Documentum Administrator.

Using a report to restore a document

The following procedure describes how to use 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 a DFC session on the server machine to restore the content.
If you need to restore renditions of the document pages manually, use an `addRendition` method.
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 DFC methods to restore it.

Arguments

[Table 59, page 495](#), lists the arguments for the tool.

Table 59. File Report arguments

Argument	Datatype	Default	Description
<code>-folder_name</code> <i>folder_path</i>	string	-	Identifies a folder path on which to run the report. May be used in conjunction with the <code>-storage-area</code> argument.

Argument	Datatype	Default	Description
-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
-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.

Argument	Datatype	Default	Description
-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 450 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 renames repository groups. This tool works in conjunction with Documentum Administrator. To rename a group, you must use the Groups pages 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 a 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 the 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 60, page 499](#), lists the log files deleted by Log Purge.

Table 60. 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 500, 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

Table 61, page 500, lists the arguments for the tool.

Table 61. Log Purge arguments

Argument	Datatype	Default	Description
-cutoff_days	integer	30	Controls what logs are deleted. All logs older than the specified number of days are deleted.
-queueperson	string(32)	-	User who receives email and inbox notifications from the tool. The default is the user specified in the operator_name property of the server config object.
-window_interval	integer	120	Execution window for the tool. Refer to The window interval, page 450 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 property 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.

The `cutoff_days` and `custom_predicate` arguments determine which `dmi_queue_items` are removed. 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

[Table 62, page 504](#), lists the arguments for the tool.

Table 62. Queue Management arguments

Argument	Datatype	Default	Description
-custom_predicate <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 dmi_queue_item object. For example, a valid qualification is "event='APPROVED'" or "name='dmadmin'".</p> <p>Refer to the <i>Content Server DQL Reference Manual</i> for complete information about WHERE clause qualifications.</p>
-cutoff_days	integer	90	<p>Defines a minimum age, in days, for dequeued items. All dequeued dmi_queue_items 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>
-queueperson	string(32)	-	<p>User who receives email and inbox notifications from the tool. The default is the user specified in the operator_name property of the server config object.</p>
-window_interval	integer	120	<p>Execution window for the tool. Refer to The window interval, page 450 for a complete description.</p>

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 an EMC Centera 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 EMC Centera storage areas that require a retention date. You can also direct the tool to operate on EMC Centera 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 EMC Centera 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 a 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 properties:

- r_object_id
- object_name
- a_storage_type
- r_creation_date
- retention_date

The retention_date property is a computed property.

The tool is installed in the inactive state.

Arguments

[Table 63, page 507](#), describes the arguments for the tool.

Table 63. Remove Expired Retention Objects arguments

Argument	Datatype	Default	Description
<i>-query_qualification</i>	string	-	Identifies which objects are selected for possible removal. This is a DQL where clause qualification.
<i>-include_zero_retention_objects</i>	Boolean	F (FALSE)	Setting this to T (TRUE) directs the job to consider objects stored in an EMC Centera storage area that allows but does not require a retention period.
<i>-queueperson</i>	string(32)	-	User who receives email and inbox notifications from the tool. The default is the user specified in the <i>operator_name</i> property of the server config object.
<i>-window_interval</i>	integer	1440	Execution window for the tool. Refer to The window interval, page 450 for a complete description.

Guidelines

An EMC Centera or NetApp SnapLock 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 property 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 property 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 property 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 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 *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 64, page 509, describes the arguments for the Rendition Management tool.

Table 64. 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 property 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 property in a content object is set to 3 when a rendition is added to content with the keepRendition 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 an 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 <code>operator_name</code> property of the server config object.
-window_interval	integer	120	Execution window for the tool. Refer to The window interval, page 450 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
foo717bbbmdadmincrtext07/24/96 20:48:46server
ContentWarningdmadmincrtext07/18/96 18:01:19server
...
...
SwapInfodmadmincrtext07/27/96 13:20:47server
RenditionMgtdmadmincrtext07/27/96 15:48:09server
rend722 dmadmnmactext07/02/96 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 Doibase 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 property 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/StateOfDocbase.

The State of the Repository tool is installed in the active state.

Arguments

Table 65, page 514, describes the tool's argument.

Table 65. State of the Repository arguments

Argument	Datatype	Default	Description
-window_interval	integer	720	Execution window for the tool. Refer to The window interval, page 450 for a complete description.

Report sample

Here is a sample report from the State of the Repository tool. The example shows all of the sections in the tool's report, but truncates entries in some sections in the interests of space.

```
StateOfDocbase Report For Docbase dm_master As Of 4/12/1999 09:26:32

Docbase Configuration:
Description:The Test Repository
Federation Name:<dm_master is not in a Federation>
Docbase 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 Name Table Owner Owner: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

GTOTAL Content:	2,127
GTOTAL 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

[Table 66, page 518](#), describes the arguments for the tool.

Table 66. 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> property of the server config object.
<code>-window_interval</code>	integer	120	Execution window for the tool. Refer to The window interval, page 450 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. The tools are installed as part of the installation procedure, and under typical circumstances, running this utility is not needed. This utility is not implemented as a job.

However, if needed, 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 RDMBS 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

[Table 67, page 521](#), lists the arguments for the tool.

Table 67. Update Statistics arguments

Argument	Datatype	Default	Description
-dbreindex	string	READ	<p>Controls whether the tool actually updates statistics or only reports on RDBMS tables that need updating.</p> <p>READ generates only the report. This setting is valid only for SQL Server and DB2 databases.</p> <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 property of the server config object.</p>
-window_interval	integer	120	<p>Execution window for the tool. Refer to The window interval, page 450 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 do not 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 pages. 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 a 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
propertys referencing user dm_autorender_mac: user_name
-----
==== ACL Objects referencing user dm_autorender_mac ====
-----
Object type : dm_acl
Object id : 450016218000010c
Name : dm_450016218000010c
propertys 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
propertys 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
propertys referincing 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
property referincing 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
property referincing 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
property referincing 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 *Content Server*

Fundamentals for a discussion of how versioning works.) 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 482.

The arguments you define for the tool determine which versions are deleted. For example, one argument (keep_slables) 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 *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

Table 68, page 526, lists the arguments for the tool.

Table 68. Version Management arguments

Argument	Datatype	Default	Description
-keep_slables	Boolean	TRUE	Indicates whether you wish to keep versions of documents with symbolic labels. The default is TRUE.
-custom_predicate <i>qualification</i>	string	-	Defines a WHERE clause qualification for the query that selects versions for deletion. The qualification must be a valid qualification. Refer to the <i>Content Server DQL Reference Manual</i> for information about WHERE clause qualifications.

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 property of the server config object.
-window_interval	integer	120	Execution window for the tool. Refer to The window interval, page 450 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 execution of the tool 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 property 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 properties are found in [Activating and scheduling administration tools](#), page 453.

Changing the default settings

The jobs that make up the tool suite are installed with default property 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 properties for a tool's job is using Documentum Administrator. However, you can also use DQL 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 properties, use the `SELECT` statement to retrieve the property values in which you are interested.

To change a tool's properties, 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 the *Content Server DQL Reference Manual*.

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.

Logging and Tracing Facilities

This chapter discusses the tracing and logging facilities supported by Content Server and DFC. It includes the following topics:

- [Introduction, page 531](#)
- [Content Server logging and tracing, page 531](#)
- [DFC logging, page 536](#)
- [DFC tracing, page 536](#)

Introduction

EMC Documentum supports robust tracing and logging facilities for both Content Server and DFC operations, to ensure that you can obtain helpful information when needed.

The logging facilities record information generated automatically by Content Server or DFC. This includes error, warning, and informational messages. Logging operations that log the error, warning, and informational messages occur automatically.

The tracing facility records information that is explicitly requested by a user. That is, trace information is recorded only if a user, such as a system administrator, or an application, through a specific method call, requests tracing. Sysadmin or Superuser privileges are required to turn tracing on and off.

Content Server logging and tracing

Content Server logging and tracing provides information about Content Server operations. Content Server records logging information and tracing information, with a few exceptions, in the following files:

- Repository log file

Content Server records information about root server activities in this file. This is also sometimes referred as the Content Server log file.

- Session log files

A session log file records all informational, warning, error, and fatal error messages and, by default, all SQL commands generated from DQL commands.

Note: It is possible, though not recommended, to turn off recording of the SQL commands.

Session log files are 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 exceptions that do not record their log and trace information in these files are a few Content Server features. These exceptions, such as jobs, that record their tracing and logging information in files that are specific to those features.

Starting and stopping Content Server tracing operations

There are several ways to start and stop tracing for Content Server operations:

- From the server startup command line
- With a `IDfSession.setServerTraceLevel`
- With a `SET_OPTIONS` administration method
- With a `MODIFY_TRACE` administration method

Note: `MODIFY_TRACE` is used only to start or stop tracing full-text operations. For information about this method, refer to the *Documentum DQL Reference Manual*.

The way you choose to initiate tracing will depend on what you are tracing. Each option controls a slightly different set of tracing functionality.

Starting and stopping tracing from the startup command line

You can initiate tracing a variety of operations from the Content Server startup command line. To start tracing at server start-up, specify the trace flags for the options with the `-o` option on 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 69, page 533, lists the trace flags that you can use with the `-o` option at server start-up and the tracing information each provides. The generated trace information is recorded in the repository log file.

Table 69. Server start-up trace options

Option name	Description
<code>debug</code>	Session shutdown, change check, launch and fork information
<code>docbroker_trace</code>	Connection broker information
<code>i18n_trace</code>	Session locale and client code page usage
<code>lock_trace</code>	Windows locking information
<code>net_ip_addr</code>	IP addresses of client and server for authentication
<code>nettrace</code>	Turns on RPC tracing (traces Netwise calls, connection ID, client host address, and client host name)
<code>rpctrace</code>	Turns on tracing of RPC calls.
<code>sqltrace</code>	SQL commands sent to the underlying RDBMS for subsequent sessions, including the repository session ID and the database connection ID for each SQL statement.
	This option is turned on by default when a server starts.
<code>ticket_trace</code>	Traces import and export operations for login ticket keys and operations using single-use login tickets
<code>trace_authentication</code>	Detailed authentication information
<code>trace_complete_launch</code>	UNIX process launch information
<code>trace_workflow_agent</code>	Trace messages generated by the workflow agent

To stop tracing from the command line, you must remove the trace flag from the server command line and then stop and restart the server.

Using `setServerTraceLevel`

The `setServerTraceLevel` method is defined for the DFC `IDfSession` interface. The method takes two arguments, a trace level, defined as an integer value, and a facility name. Table 70, page 534, lists the possible trace levels, and Table 71, page 534, lists the facilities that you can specify.

Table 70. Trace level severity values

Severity level	Meaning
0	No messages, tracing is turned off. Use this value to turn off tracing for the specified facility.
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

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.

Table 71. Valid facilities for setServerTraceLevel

Facility	Description
ALL	Traces all available trace facilities
DM_CONTENT	Traces content operations. The information is recorded in the session log file.
DM_DUMP	Traces dump operations. The information is recorded in the session log file.
DM_LOAD	Traces load operations. The information is recorded in the session log file.
DM_QUERY	Traces query operations. The information is recorded in the session log file.
SQL_TRACE	Traces generated SQL statements. The information is recorded in the session log file.
	Tracing for this facility is turned on by default. Turning it off is not recommended by Documentum Technical Support.
DM_SYSOBJECT	Traces SysObject operations. The information is recorded in the session log file.

Using SET_OPTIONS

The SET_OPTIONS administration method turns on or off a wide range of Content Server tracing options, including tracing EMC Centera and NetApp SnapLock storage area operations, digital shredding operations, and connection broker information, for example. For a complete list of the trace options available for SET_OPTIONS, refer to its reference information in the *Content Server DQL Reference Manual*.

The method can be executed from Documentum Administrator or using the DQL EXECUTE statement or an IDfSession.apply method.

Although not recommended by Documentum Technical Support, you can use SET_OPTIONS to turn off SQL tracing.

Turning tracing on and off using SET_OPTIONS affects only new sessions. Current sessions are not affected.

Examples of server tracing

Here is a server log excerpt for DM_LOAD tracing:

```
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 shows a log file excerpt for query tracing. The example assumes that the following query is issued:

```
SELECT "user_os_name" FROM "dm_user" WHERE "user_name"='zhan'
```

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."
```

Determining which tracing options are turned on

To determine whether a particular Content Server tracing option is on turned on, use an `IdfSession.isServerTraceOptionSet` method. This method takes as an argument the name of the option. For example, if you wish to determine if the `docbroker_trace` tracing option is turned on, you would specify that option name as the argument to the method.

You cannot specify a facility name with the `isServerTraceOptionSet` method.

DFC logging

DFC logging is controlled by the `log4j.properties` file. The location of the generated log file is defined in the `log4j.properties` file. The default location defined in the file is `${user.dir}/documentum/logs/documentum.log`.

Refer to the standard Java documentation for information about the keys that may be set in this file, to control logging.

If you wish to record debug information generated by specific packages or classes, it is recommended that you direct that information to the DFC trace file, rather than the `log4j` log file. Instructions are found in [Directing categories to the trace file, page 547](#).

DFC tracing

DFC supports a set of tracing configuration options that allow you to obtain desired trace information about DFC operations, formatted in a manner most helpful to you. For example, you can turn on tracing for individual users or individual threads and specify which methods to trace. You can also configure characteristics of the trace file such as how timestamps within the file are formatted or how method entrances and exits are recorded.

All DFC tracing is controlled by entries in a properties file. The entries are dynamic. You can add, remove, or change a tracing key entry and the change is effective immediately. It is not necessary to stop and restart DFC or the application.

The logger and logging appender

Entries in the trace log file are recorded by a logger and its associated logging appender created by DFC when you initiate tracing. The configuration of the appender determines some characteristics of the log file, such as its file name and its maximum size. [Configuring the logging appender, page 537](#), describes the entries that configure the appender.

Enabling tracing

Before you actually start tracing, it is recommended that you set the keys in `dfc.properties` that configure the logger and define what you want to trace. If you do not set these, the tracing facility is initialized with the default values. The keys that configure the logging appender are described in [Configuring the logging appender, page 537](#). The keys that define what to trace are described in [Defining what is traced, page 542](#).

To initialize the DFC tracing facility and start tracing, set the `dfc.tracing.enable` key to true:

```
dfc.tracing.enable=true
```

This key is false by default.

Configuring the logging appender

The configuration of the logging appender controls the name and location of the trace file and its format. [Table 72, page 537](#), lists the configuration options for the appender.

Table 72. Logging appender configuration options

Option	Key	Description
File name prefix	<code>dfc.tracing.file_prefix</code>	Defines the base name of the trace file. Refer to Trace file names, page 539 , for details about choosing a base name. The default value is <code>dfctrace</code> .

Option	Key	Description
Location of file	dfc.tracing.dir	Identifies the directory in which to store the trace file. If unset, the file is stored in the directory identified by <code>\$(dfc.data.dir)/logs</code> .
Creation mode	dfc.tracing.file_creation_mode	Valid values for this property are any valid directory path. Setting this property is recommended. Controls whether trace information is logged in one file or multiple files. Defining file creation mode, page 539 , describes this in detail.
Maximum file size	dfc.tracing.max_file_size	Defines the maximum size of a trace file before it is rolled over. Valid values are any string accepted by <code>log4j</code> as <code>MaxFileSize</code> . If an integer value is specified with no unit of measure, the integer value is interpreted as bytes. The default is 100MB.
Maximum number of backups for the file	dfc.tracing.max_backup_index	Defines the number of back-up files the logger will retain for the log file. When the maximum number is reached, the oldest backup is destroyed when a new back-up is created.

Option	Key	Description
Date format for timestamp	<code>dfc.tracing.date_format</code>	Valid values are any positive integer. The default value is 1. Defines a date format if <code>dfc.tracing.timing_style</code> , in <code>dfc.properties</code> , is set to <code>date</code> . Refer to Defining the timestamp format, page 540 , for complete information about setting <code>dfc.tracing.timing_style</code> and its interaction with <code>dfc.tracing.date_format</code> in the <code>dfc.properties</code> file.

Trace file names

Trace log file names have the following format:

```
file_prefix[.identifier].timestamp.log
```

The file prefix is prepended to the beginning of the name of each trace file generated by DFC. For example, if you are recording all information in one log file, the name of the file is in the format:

```
file_prefix.timestamp.log
```

The default value for the file prefix is `dfctrace`. You can define a different prefix by setting the `dfc.tracing.file_prefix` key:

```
dfc.tracing.file_prefix=value
```

An identifier is included in the file name only if you are generating log files for individual users or threads. The identifier is the user name if the logs are being generated for individual users, or the thread name if logs are being generated for individual threads.

The *timestamp* records when the file was created.

Defining file creation mode

The `dfc.tracing.file_creation_mode` key controls whether the log entries are recorded in one log file or several. By default, the default value for this key is `standard`, which means

that all DFC trace entries are recorded in one log file. [Table 73, page 540](#), lists the valid values for this key and the effects of each on log file creation.

Table 73. Valid values for `dfc.tracing.file_creation_mode`

Value	Description
standard	All log entries are recorded in one log file. This is the default.
thread	<p>DFC creates a log file for each thread that has trace data to log. The log file for each thread is named using the following format:</p> <pre><i>file_prefix.threadname. timestamp.log</i></pre> <p>Tracing by thread is only recommended if you also set the <code>dfc.tracing.thread_name_filter</code> key.</p>
user	<p>DFC creates a log file for each user that has trace data to log. The log file for each user is named using the following format:</p> <pre><i>file_prefix.username default. timestamp.log</i></pre> <p>If the specific user cannot be determined for a particular call, the trace for that call is logged in a separate log file, with username specified as 'default'.</p> <p>Tracing by user is only recommended if you also set the <code>dfc.tracing.user_names_filter</code> key.</p>

Defining the timestamp format

Each entry in a log file has a timestamp. If the file is recorded in compact mode, there are two columns for the timestamp. The first column records the entry time and the second column records the duration of the method call—the difference between method entry and method exit time. If the log file is recorded in standard mode, there is one column for the timestamp in the entry. The value recorded is either the entry time or the exit time, depending on whether the log entry is recording the method's entry or exit.

By default, timestamps are expressed in seconds. You can configure the timestamp display. It is controlled by the `dfc.tracing.timing_style` key. [Table 74, page 541](#), describes the valid values for `dfc.tracing.timing_style`.

Table 74. Valid values for `dfc.tracing.timing_style`

Value	Description
nanoseconds	<p>The timestamp values are those returned by a call to <code>System.nanoTime()</code>.</p> <p>These values may or may not actually have nanosecond granularity, depending on the underlying operating system. The values cannot be correlated to absolute time.</p>
milliseconds	<p>The timestamp values are those returned by a call to <code>System.currentTimeMillis()</code>.</p>
milliseconds_from_start	<p>The timestamp is recorded as the number of milliseconds from the start of the JVM process.</p>
seconds	<p>The timestamp values are displayed as the number of seconds from the start of the process. The value is a float.</p> <p>This is the default timing style.</p>
date	<p>The timestamp values are displayed as a date string. The default format is:</p> <pre>yyyy/MM/dd-HH:mm:ss.S</pre> <p>(Refer to the Javadocs for the <code>SimpleDateFormat</code> class for details.)</p> <p>The date setting may only be used if <code>dfc.tracing.mode</code> is set to standard. If you set <code>dfc.tracing.timing_style</code> to date and the mode is compact, the timing style defaults to milliseconds.</p> <p>If the timing style is set to date, you can also set <code>dfc.tracing.date_format</code> and optionally, <code>dfc.tracing.date_format_width</code>, to define an alternate date format. Defining a date format, page 542, describes how to define an alternate format.</p>

Defining a date format

If you have set `dfc.tracing.timing_style` to `date`, you may also set `dfc.tracing.date_format` and `dfc.tracing.date_column_width`. Use the `dfc.tracing.date_format` property to define a format for the dates entered in the trace files that is different from that default format. The format you specify must conform to the syntax supported by the Java class `java.text.SimpleDateFormat`.

The default column width allotted to dates in the trace file entries is 14. If the format you specify in `dfc.tracing.date_format` is wider than 14 characters, set `dfc.tracing.date_column_width` to the required number of characters. You can set this to any positive integer value.

Defining what is traced

There are a variety of keys that you can use to define what is traced and to set a few more options for the format of the file. You can, for example, configure how methods are traced, whether trace information is placed in one file or broken out into multiple files by thread or user, and define maximum number of users or threads to trace.

Configuring method tracing

The following keys control method tracing:

- `dfc.tracing.max_stack_depth`
- `dfc.tracing.mode`
- `dfc.tracing.method_name_filter`

Defining maximum stack depth to trace

The `dfc.tracing.max_stack_depth` key controls the depth of tracing into the DFC call stack. The default value for this key is 1, meaning only the first level of method calls is traced.

To change the default, reset the property to the desired depth. The first method call in a stack is considered to have a depth level of 1. So, for example, suppose you have an application like the following:

```
call methodA (stack depth=1)
  methodA calls methodB (stack depth=2)
    methodB calls methodC (stack depth=3)
      methodC calls methodD (stack depth=4)
```

If you want to trace calls in the stack to the third invocation, where methodB calls methodC, you set:

```
dfc.tracing.max_tracing_depth=3
```

Setting `max_stack_depth` to 3 causes DFC to record the calls to methodA, methodB, and methodC in the log file. The call to methodD is not traced because that is the fourth level of depth, and tracing depth level is set to 3.

Setting the key to -1 means that the traced stack depth is unlimited.

Specifying method entry and exit tracing

The `dfc.tracing.mode` key controls how method entry and exit is recorded in the log file. There are two possible values for mode: compact and standard.

In compact mode, DFC adds one line to the file that records both entry and exit and return value for a method. The methods are logged in the order of method entrance. In compact mode, DFC buffers in memory the log entries for a sequence of method calls until the topmost method returns.

If you set mode to standard, DFC creates one line in the file for a method's entry and one line for a method's exit. In standard mode, the log entries for exit record the return values of the methods.

The default setting is compact.

Identifying which packages, classes, and methods to trace

The `dfc.tracing.method_name_filter[n]` key identifies the packages, classes, and methods or any combination of these to be traced. The key is a repeating key, which means you can put multiple entries for the key into the `dfc.properties` file. Each entry has one value, identifying a package, class or method to be traced. Like repeating repository properties, each `dfc.tracing.method_name_filter` entry is referenced using a square-bracketed integer, with the first entry beginning with zero.

For example, if you want to include three entries for this key, they would be:

```
dfc.tracing.method_name_filter[0]=value
dfc.tracing.method_name_filter[1]=value
dfc.tracing.method_name_filter[2]=value
```

The value is one or more string expressions that identify what to trace. The syntax of the expression is:

```
([qualified_classname_segment][*|*]) [.[method_name_segment][*]()]
```

The asterisk is a wild card.

Here are some examples of values for this key and what tracing results from each:

- This example traces all methods on DfPersistentObject and any lower level calls made within the context of those methods:
`com.documentum.fc.client.DfPersistentObject`
- This example traces the DfPersistentObject.save() method and all lower-level calls.
`com.documentum.fc.client.DfPersistentObject.save()`
- This example traces all methods on all classes whose package name starts with "com.documentum.fc.client".
`com.documentum.fc.client*`
- This example traces the save method on all classes in the package com.documentum.fc.client where the class name starts with DfP.
`com.documentum.fc.client.DfP*.save()`
- This example traces all methods whose name begins with "do" for all classes whose package is either com.documentum.fc.client or "sub-package" of com.documentum.fc.client.
`com.documentum.fc.client.*.do*()`
- This example traces the method setString() on any DFC class in any package.
`*.setString()`

Tracing users by name

When you want to trace only particular users, set the `dfc.tracing.user_name_filter` key to identify those users. The key is a repeating key, which means you can put multiple entries for the key in the file. Each entry specifies one regular expression that identifies a user or users to be traced. Like repeating properties in the repository, each `dfc.tracing.user_name_filter` entry is referenced using a square-bracketed integer, with the first entry beginning with zero.

For example, if you want to include three entries for this key, they would be:

```
dfc.tracing.user_name_filter[0]=expression
dfc.tracing.user_name_filter[1]=expression
dfc.tracing.user_name_filter[2]=expression
```

The expression is a regular expression, using the syntax for a regular expression as defined in the Java class `java.util.regex.Pattern`.

When this property is set, DFC traces activities of the users whose login names (`dm_user.user_login_name`) match the specified expression. The log entries will contain the user name.

Note: If you want to completely separate out various user entries, you can use `dfc.tracing.file_creation_mode` to create a separate log file for each user. Refer to [Defining file creation mode, page 539](#), for details.

The `dfc.tracing.user_name_filter` key is not set by default, which means that all users are traced.

The tracing output may not include all DFC calls made by a particular user. For some calls, it is not possible to identify the user. For example, most methods on the `DfClient` interface are unrelated to a specific session or session manager. Consequently, when tracing is constrained by user, these method calls are not traced.

When tracing by user, DFC cannot identify the user until one of the following occurs:

- A method call on an object that derives from `IDfTypedObject` (if the session associated with that object is not an API session)
- A method call that takes an `IDfSession` or `IDfSessionManager`
- A method call that returns an `IDfSession` or `IDfSessionManager`
- An `IDfSessionManager` method call that sets or gets an `IDfLoginInfo` object

Tracing threads by name

To trace specific threads, set the `dfc.tracing.thread_name_filter` key. The key is a repeating key, which means you can put multiple entries for the key into the file. Each entry has one value, identifying a thread or threads to be traced. Like repeating repository properties, each `dfc.tracing.thread_name_filter` entry is referenced using a square-bracketed integer, with the first entry beginning with zero.

For example, if you want to include three entries for this key, they would be:

```
dfc.tracing.thread_name_filter[0]=expression
dfc.tracing.thread_name_filter[1]=expression
dfc.tracing.thread_name_filter[2]=expression
```

Each `dfc.tracing.thread_name_filter` entry is set to a regular expression that identifies a thread or threads you want to trace. The regular expression must use the syntax for a regular expression defined in the Java class `java.util.regex.Pattern`.

The key is not set by default, which means that all methods in all traced threads are traced by default. Setting the key is primarily useful for standalone DFC-based applications that may spawn DFC worker threads.

Note: You can use `dfc.tracing.file_creation_mode` to further sort the entries for each thread into separate files. For information, refer to [Defining file creation mode, page 539](#).

Including the session ID

Log entries are identified, by default, with the user name associated with the operation being logged. You may also want each entry to include a session ID. To direct DFC to

include session information, set the `dfc.tracing.include_session_id` property to true. The property is false by default. Setting it to true causes DFC tracing to add the session ID and the identity hash code of the associated session manager to the log entries. The session ID is in the form *sn*; for example, *s1* and *s2*.

Tracing RPC calls

There are two keys that control tracing of RPC calls:

- `dfc.tracing.include_rpc_count`
- `dfc.tracing.include_rpcs`

The `dfc.tracing.include_rpc_count` is a Boolean key. If it is set to true, DFC adds an additional column to each log file entry that records the current RPC call count for the associated session. The logged value is "N/A" if DFC cannot determine the session. If the mode is set to compact, the logged RPC count is the count at the time the method exits. The default value for the key is false.

The `dfc.tracing.include_rpcs` key is a Boolean key. When set to true, DFC includes a separate line in the trace file for each RPC call that is executed. The default for this key is false.

Including stack trace for exceptions

By default, the DFC tracing facility only logs exception names and messages. Typically, the stack trace can be determined from the trace file. However, in some circumstances, you may want to obtain an exact stack trace for an exception. To do so, set the `dfc.tracing.print_exception_stack` property to true.

Setting this property to true causes the tracing facility to log the entire stack trace for the exception. The stack trace is recorded immediately following the line that logs the exit of the method that caused the exception. The trace is indented and prefixed with exclamation marks.

The default value for this property is false. Setting this property to true is only recommended when the exact stack trace is important and you think that some exceptions are not being logged by the DFC tracing facility.

Setting verbosity

Verbosity determines what set of classes are traced. The default verbosity setting, `false`, traces a standard set of classes. If verbosity is set to true, an additional set of classes is

traced. These additional classes are low-level classes. Tracing these classes adds only limited additional information to the file, but adds many additional entries, possibly slowing the system noticeably. Turning on verbose mode is only recommended when suggested by Support.

To turn on additional verbosity, set the following key to true:

```
dfc.tracing.verbosity=true
```

To return to standard verbosity levels, set the key to false:

```
dfc.tracing.verbosity=false
```

Directing categories to the trace file

In a `log4j.properties` file, you can define categories as the target of logging operations. For DFC, a category specification would typically be a class or package. It is recommended that if you want tracing information, particularly at the DEBUG level, for a DFC class or package, that you record that information in the trace file generated by the `dfc.properties` tracing facility rather than the `log4j` log file.

To enable that redirection, use the following `dfc.properties` keys:

- `dfc.tracing.log.category`
- `dfc.tracing.log.level`
- `dfc.tracing.log.additivity`

The `dfc.tracing.log.category` key identifies the category you want to trace in the DFC trace log file. The key is a repeating key, which means you can put multiple entries for the key into the file. Each entry has one value, identifying a category to be traced. Like repeating repository properties, each `dfc.tracing.log.category` entry is referenced using a square-bracketed integer, with the first entry beginning with zero. For example:

```
dfc.tracing.log.category[0]=class_or_package_name  
dfc.tracing.log.category[1]=class_or_package_name
```

The `dfc.tracing.log.level` indicates the level of tracing you wish to use. The `dfc.tracing.log.level` key defaults to DEBUG if not specified. The `dfc.tracing.log.additivity` key is the additivity setting for the category. The `dfc.tracing.log.additivity` key defaults to false if not specified.

The `dfc.tracing.log.level` and `dfc.tracing.log.additivity` keys are also repeating keys, and the values across one index position in the entries for the keys are the settings for category identified at the corresponding index position in `dfc.tracing.log.category`.

Interactions of tracing specifications

The actual entries in the trace log files are the intersection of all the `dfc.tracing` key values you have set. For example, if you set `dfc.tracing.max_depth` to 4 and `dfc.tracing.mode` to `thread`, DFC records method calls to a depth of 4 invocations and sorts the entries by thread into separate log files.

Log file entry format

Trace information is recorded in the log files in the following format:

```
timestamp [method_duration] [threadname]
<username|sessionID|session_mgr_id> (RPCs=count) [entry_exit_designation]
stack_depth_indicator qualified_class_name@object_identity_hash_code.
method(method_arguments) ==>return_value|exception
```

Note: Bolded characters appear in the log entry; they are not used to indicate optional items or choices.

[Table 75, page 548](#), describes the items found in entries.

Table 75. Log entry components

Component	Description
<i>timestamp</i>	Timestamp of the entry. Its format is dependent on the tracing configuration, as defined by <code>dfc.tracing.timing_style</code> .
<i>method_duration</i>	The total length of time to execute the method. This value is only included if <code>dfc.tracing.mode</code> is set to <code>compact</code> .
<i>threadname</i>	Name of the thread associated with the entry.
<i>username</i>	Name of the user associated with the entry.
<i>sessionID</i>	Identifies the session associated with the entry. This is only included if <code>dfc.tracing.include_session_id</code> is set to <code>true</code> .
<i>session_mgr_id</i>	The hash code identity of the session manager associated with the entry. This is only included if <code>dfc.tracing.include_session_id</code> is set to <code>true</code> .
RPCs=count	Total number of RPC calls at the time the entry was logged. This is only included if <code>dfc.tracing.include_rpc_count</code> is set to <code>true</code> .

Component	Description
<i>entry_exit_designation</i>	Keyword that identifies source of the log entry. The keyword is only included if the <code>dfc.tracing.mode</code> is standard. Valid values are: <ul style="list-style-type: none"> • ENTER, meaning the entry records a method entry • EXIT, meaning the entry records a method exit • !EXC!, meaning the entry records a call that results in an exception • RPC_ENTER, meaning the entry records an RPC call entry • RPC_EXIT, meaning the entry records an RPC call exit
<i>stack_depth_indicator</i>	Indicates the depth of call stack tracing. The stack depth indicator is a series of dots (for example, ...).
<i>qualified_class_name</i>	Fully qualified name of the class being traced
<i>object_identity_hash_code</i>	Hash code of the object on which the method is being called. If the method is static, this element does not appear in the log entry.
<i>method</i>	Name of the method being traced
<i>method_arguments</i>	Arguments to the specified method
<i>return_value</i>	The value returned by the method. This is included if <code>dfc.tracing.mode</code> is COMPACT or if the entry records the method's exit.
<i>exception</i>	The exception name and message, if any, of the exception thrown by the method.

The values in each column in an entry are separated by spaces, not tabs.

Trace file examples

Here are some examples of trace files generated by various combinations of property settings. Note that due to space limitations on the page, the individual entries are line-wrapped in the examples.

Example 12-1. Settings are: `dfc.tracing.enable=true`

```
1158701543173 <user1> [Thread-0] [ENTER]
.....com.documentum.fc.client.DfTypedObject@28821120.getData ()
1158701543173 <user1> [Thread-0] [EXIT]
.....com.documentum.fc.client.DfTypedObject@28821120.getData==>
```

```
com.documentum.fc.client.impl.typeddata.ITypedData@150ed68
```

Example 12-2. Settings are: dfc.tracing.enable=true; dfc.tracing.timing_style=millis_from_start

```
1141125 <user4> [Thread-5] [EXIT]
.....com.documentum.fc.client.impl.session.Session@4889213
.incrementReferenceCountIfNonZero ==> 2
```

Example 12-3. Settings are: dfc.tracing.enable=true; dfc.tracing.thread_name_filter=Thread[45]

```
1158718491103 <user4> [Thread-4] [ENTER]
.....com.documentum.fc.client.DfTypedObject@25700470.getData()
1158718491103 <user4> [Thread-4] [EXIT]
.....com.documentum.fc.client.DfTypedObject@25700470.getData
==> com.documentum.fc.client.impl.typeddata.ITypedData@618b08
1158718491150 <user5> [Thread-5] [EXIT]
.....com.documentum.fc.impl.util.io.MessageChannel@15999328.readSocket
==> <void>
1158718491166 [Thread-5] [EXIT]
.....com.documentum.fc.impl.util.io.MessageChannel@15999328
.readLength ==> 18
```

Example 12-4. Settings are: dfc.tracing.enable=true; dfc.tracing.include_rpc_count=true; dfc.tracing.include_session_id=true

```
1158702906324 <user5|s4|SM@25116828> [Thread-6] [ENTER] (RPCs=3269)
...com.documentum.fc.client.impl.connection.docbase.DocbaseConnection
@17535609.waitForCorrectTransactionContext()
```

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 557](#)
- [Workflow checks, page 557](#)
- [Object type checks, page 558](#)
- [Data dictionary checks, page 559](#)
- [Lifecycle checks, page 560](#)
- [Object type index checks, page 561](#)
- [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 76. 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

Consistency check number	Consistency check description	Severity
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 77. 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 <code>dm_acl_r</code> entries	Error
CC-0009	Check for sysobjects whose <code>acl_domain</code> 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 <code>dm_acl_s</code> entry	Error
CC-0013	Check for ACL objects with <code>r_accessor_permit</code> values that are missing <code>r_accessor_name</code> values	Error

Consistency check number	Consistency check description	Severity
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
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 78. 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

Consistency check number	Consistency check description	Severity
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
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 79. 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

Consistency check number	Consistency check description	Severity
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
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 80. 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

Consistency check number	Consistency check description	Severity
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 81. 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 82. 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

Consistency check number	Consistency check description	Severity
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
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 83. 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

Consistency check number	Consistency check description	Severity
CC-0050	Check whether any dmi_type_info objects reference non-existent dm_type objects	Error
CC-0051	Check whether any types have corrupted property positions	Error
CC-0052	Check whether any types have invalid property counts	Error

Data dictionary checks

These checks apply to the data dictionary.

Table 84. 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

Consistency check number	Consistency check description	Severity
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 85. 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

Consistency check number	Consistency check description	Severity
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 86. 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
CC-0075	Check for types with non-existent dmi_index object for _r table	Warning
CC-0076	Check for indexes with invalid property positions	Warning

Method object consistency checks

These consistency checks apply to method objects.

Table 87. Consistency checks for method objects

Consistency check number	Consistency check description	Severity
CC-0077	Check for methods using jview rather than Java	Warning

IDQL and IAPI

This appendix describes the Interactive DQL (IDQL) and IAPI utilities. The utility is useful if you want to test short scripts or perform short, small tasks in the repository. The IAPI utility allows you to execute Docbasic scripts.

The appendix includes the following topics:

- [Using IDQL, page 563](#)
- [Using IAPI, page 568](#)

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 88, page 564](#), describes the command line arguments.

Table 88. 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 or -ENV_CONNECT_USER_NAME	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.
-Pos_password or -ENV_CONNECT_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.
	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

Argument	Description
	uses the default password, NULL. Passwords are case sensitive.
-D <i>domain</i>	Identifies the user's domain.
<i>repository</i>	Identifies the repository. You can include the repository name on the command line, or you can use the
or	-ENV_CONNECT_DOCBASE_NAME argument, which directs the system to look for the repository name in the environment variable DM_DOCBASE_NAME.
-ENV_CONNECT_DOCBASE_NAME	
-R <i>inputfile</i>	Specifies the name of an input file containing DQL queries. This is an optional argument.
-X	Allows prompt for domain.
-E	Turns on echo.
-L <i>trace_level</i>	Turns tracing on for the session. <i>trace_level</i> can be any of the valid trace levels described in the Trace method description.
-zZ	Specifies Windows NT Unified Logon. This option overrides any user name and password specified on the command line.
-W <i>integer</i>	Sets the maximum column width when displaying results. <i>integer</i> is interpreted as number of bytes. The default is 2000 bytes.
or	
-w <i>integer</i>	Any value which exceeds the maximum row width is truncated when displayed.
-K <i>secure_mode</i>	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 Defining the secure connection default for connection requests , page 166.

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 89, page 566](#).

Table 89. 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: <code>describe [type] type_name</code> <code>describe table table_name</code>
@scriptfile	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	Turns on tracing for the current session. By default, tracing is set to level 5.
-Winteger or -winteger	Sets the maximum column width when displaying results. integer is interpreted in bytes. The default is 2000 bytes. 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.

Using IAPI

The IAPI utility is a tool that you can use to execute Docbasic scripts. This utility is included and installed with Content Server. It is found in `%DM_HOME%\bin` (`$DM_HOME/bin`).

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 90, page 569](#), describes the command line arguments.

Table 90. IAPI command line arguments

Argument	Description
- <i>Username</i>	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	
- <i>Pos_password</i>	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.
- <i>Logfile</i>	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.
-V-	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).
	You can also change modes from the IAPI prompt (refer to IAPI commands , page 571).
-Q	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.

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>	Identifies an input file containing API methods.
<code>-Ftrace_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>-Sinit_level</code>	Defines the connection level established when you start the API session. Valid init levels and their meanings are: api , which opens an API session. User must issue an explicit Connect to start a session. connect , meaning undetermined standard , which opens a session
<code>-Iescape_character</code>	Defines an escape character for certain symbols such as a forward slash (/).
<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 returned from a query entered with a ? command. <i>integer</i> is interpreted as number of bytes. The default is 2000 bytes.
or <code>-winteger</code>	
<code>-Ksecure_mode</code>	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

Argument	Description
	<ul style="list-style-type: none"> try_native_first <p>The default is try_native_first. For an explanation of the values, refer to Defining the secure connection default for connection requests, page 166.</p>

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 91, page 571](#).

Table 91. IAPI commands

Command	Description
<code>\$logfile</code>	Outputs to the specified file
<code>@scriptfile</code>	Reads input from the specified file
<code>%scriptfile</code>	Outputs a script
<code>-V[+ -]</code>	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).
<code>-Winteger</code> or <code>-winteger</code>	Sets the maximum column width when displaying results returned from a query entered with a <code>?</code> command. <code>integer</code> is interpreted in bytes. The default is 2000 bytes. Setting this affects the results display for all methods issued after this command.
<code>?</code>	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

Note: This use of IAPI is deprecated, as the DMCL that was the primary API for Content Server in versions prior to Documentum 6 is deprecated in Documentum 6.

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
```


Archiving scripts

This appendix contains sample scripts to support the archiving functionality. You must edit these generic scripts before running to them to localize them for your installation. Contact Documentum Professional Services or Documentum Developer support for assistance to customize these scripts.

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 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 92, page 589](#), lists the processes that have monitoring scripts, the script names, and their locations and command-line syntax.

Table 92. 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 com.documentum.server. impl.utils.ContentServerStatus -docbase_name xxxx -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 <code>dm_server_config.app_server_name</code> 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 com.documentum.server.impl. utils.IndexAgentCtrl -docbase_name</pre>

Process	Script	Syntax and Location
		<pre><i>repository_name</i> -user_name <i>user_name</i> -action status</pre>
		<p><i>repository_name</i> is the name of a repository served by the agent and <i>user_name</i> is the user account with which to connect to the repository.</p>

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](#)
- [Data dictionary population script, page 594](#)
- [Publishing the data dictionary information, page 606](#)

Populating the data dictionary

When you configure a repository, 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` property 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` property if needed.

Only some of the data dictionary information can be set using a population script. ([Summary of settable data dictionary properties, page 597](#), describes the data dictionary properties that you can set in a population script.) The information that you can not 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-NLS 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 a property'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 property in the German locale, the system sets `label_text` in the new property's `dd attr` info object in the German locale. It will not set `label_text` in the property'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 properties, the label text in the second file overwrites the label text in the first file. Unduplicated information is not 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 a property, the property. 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. [Table 93, page 596](#), lists the code pages for the data files provided with Content Server.

Table 93. Codepages for the data dictionary files provided with Content Server

Data file locale	Codepage
en (English)	ISO_8859-1
es (Spanish)	ISO_8859-1
fr (French)	ISO_8859-1
it (Italian)	ISO_8859-1
de (German)	ISO_8859-1
sv (Swedish)	ISO_8859-1
ja (Japanese)	Shift_JIS
ko (Korean)	EUC-KR
zh (Chinese)	MS936
ru (Russian)	Windows-1251
pt (Brazilian Portuguese)	Windows-1252

To define information for an object type, the format is:

```
TYPE=type_name
<data dictionary property settings>
```

To define information for a property, the format is:

```
TYPE=type_name
property=property_name
<data dictionary property settings>
```

If you want to define information for multiple properties of one object type, specify the type only once and then list the properties and the settings:

```
TYPE=type_name
property=property_name
<data dictionary property settings>
{property=property_name
<data dictionary property settings>}
```

The setting for one data dictionary property settings must fit on one line.

The next section, [Summary of settable data dictionary properties, page 597](#), lists the data dictionary properties 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 properties

Data dictionary information is stored in the repository in internal object types. When you set data dictionary information, you are setting the properties of these types. Some of the information applies only to object types, some applies only to properties, and some can apply to either or both.

Table 94, page 597, lists and briefly describes the data dictionary properties that you can set using a population script.

Table 94. Settable data dictionary properties using population script

property name	Reference in script as:	Which level	NLS-specific? Yes or no	Comments
ignore_ immutable	IGNORE_ IMMUTABLE	property	No	None
allowed_search_ ops	ALLOWED_ SEARCH_OPS	property	No	If allowed_ search_ops is set, you must set default_search_ops.
default_search_ ops	DEFAULT_ SEARCH_OP			
default_search_ arg	DEFAULT_ SEARCH_ARG			Default_search_arg, if set, must be set to one of the allowed_ search_ops.
read_only	READ_ONLY	property	No	None
is_hidden	IS_HIDDEN	property	No	None
is_required	IS_REQUIRED	property	No	None
not_null	NOT_NULL	property	Yes	Setting not_null sets the not_null data dictionary property to TRUE.
not_null_msg	REPORT= <i>string</i>			Including REPORT (not_null_msg) or ENFORCE (not_null_enf) is optional. If you include both, REPORT must appear first.
not_null_enf	ENFORCE= <i>string</i>			Valid values for ENFORCE are

property name	Reference in script as:	Which level	NLS-specific? Yes or no	Comments
map_data_string	MAPPING_TABLE	property	Yes	application and disable. 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.)
map_display_string	VALUE= <i>integer</i> DISPLAY= <i>string</i>			
map_description	COMMENT= <i>string</i>			
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 property	Yes	None
help_text	HELP_TEXT	Object type property	Yes	None
comment_text	COMMENT_TEXT	Object type property	Yes	None
is_searchable	IS_SEARCHABLE	Object type property	No	None

property 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: PRIMARY_KEY=<i>key</i> REPORT=<i>string</i> ENFORCE=<i>string</i></p> <p>If defined at property level: PRIMARY_KEY REPORT=<i>string</i> ENFORCE=<i>string</i></p>	Object type property	Yes	<p>If the primary key consists of one property, you can define the key at either the property or type level. At the type level, you must name the property in <i>key</i>. If you define the key at the property level, naming the property is not necessary.</p> <p>If the primary key consists of multiple properties, you must specify the key at the type level. Provide a comma-separated list of the properties 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>

property 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:</p> <p>UNIQUE_KEY=<i>key</i> REPORT=<i>string</i> ENFORCE=<i>string</i></p> <p>If defined at property level:</p> <p>UNIQUE_KEY REPORT=<i>string</i> ENFORCE=<i>string</i></p>	Object type property	Yes	<p>If the unique key consists of one property, you can define the key at either the property or type level. At the type level, you must name the property in <i>key</i>. If you define the key at the property level, naming the property is not necessary.</p> <p>If the unique key consists of multiple properties, you must specify the key at the type level. Provide a comma-separated list of the properties 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>

property 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 property level: FOREIGN_KEY= <i>type(attr)</i> REPORT= <i>string</i> ENFORCE= <i>string</i>	Object type property	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 property level.

If you set the key at the type level, you must identify which property of the type participates in the foreign key and which property in another type is referenced by the foreign key. The key phrase FOREIGN_KEY defines the property in the object type that participates in the foreign key. The keyword REFERENCES defines the property 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 property level, the data file would include the following lines:

```
TYPE=personnel_action_doc
property=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 property level information for user_name property
TYPE=dm_user
property=user_name
IS_REQUIRED
DEFAULT_SEARCH_OP=contains

#set property level information for dm_document
TYPE=dm_document
property=acl_domain
IGNORE_IMMUTABLE
property=acl_name
IGNORE_IMMUTABLE
property=title
IS_REQUIRED
DEFAULT_SEARCH_OP=contains
property=subject
DEFAULT_SEARCH_OP=contains
property=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 property Information for dm_user
TYPE = dm_user
property = alias_set_id
LABEL_TEXT = User Alias Set ID

# Set property Information for dm_group
TYPE = dm_group
property = alias_set_id
LABEL_TEXT = Group Alias Set ID

# Set property Information for dm_process
TYPE = dm_process
property = perf_alias_set_id
LABEL_TEXT = Performer Alias Set ID

# Set property Information for dm_workflow
TYPE = dm_workflow
property = r_alias_set_id
LABEL_TEXT = Performer Alias Set ID

# Set property Information for dm_activity
TYPE = dm_activity
property = 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
property = resolve_pkg_name
LABEL_TEXT = Alias Resolution Package

# Set property Information for dm_sysobject
TYPE = dm_sysobject
property = a_effective_label
LABEL_TEXT = Effective Label
property = a_effective_date
LABEL_TEXT = Effective Date
property = a_expiration_date
LABEL_TEXT = Expiration Date
property = a_effective_flag
LABEL_TEXT = Effective Flag
property = a_publish_formats
LABEL_TEXT = Publish Formats
property = a_category
LABEL_TEXT = Category
property = language_code
LABEL_TEXT = Language Code
property = authors
FOREIGN_KEY = dm_user(user_name)
REPORT = The author is not found in the repository.
ENFORCE = application

# Set property information for dmr_containment
TYPE = dmr_containment
property = a_contain_type
LABEL_TEXT = Containment Type
property = a_contain_desc
LABEL_TEXT = Containment Description

# Set property Information for dm_assembly
TYPE = dm_assembly
property = path_name
LABEL_TEXT = Path Name

# Set property Information for dm_relation
TYPE = dm_relation
property = r_object_id
LABEL_TEXT = Object ID
property = relation_name
LABEL_TEXT = Relation Name
property = parent_id
LABEL_TEXT = Parent ID
property = child_id
LABEL_TEXT = Child ID
property = child_label
LABEL_TEXT = Child Label
```

```
property = permanent_link
LABEL_TEXT = Permanent Link
property = order_no
LABEL_TEXT = Order Number
property = effective_date
LABEL_TEXT = Effective Date
property = expiration_date
LABEL_TEXT = Expiration Date
property = description
LABEL_TEXT = Description
```

This final example sets some constraints and search operators for the object types and their properties.

```
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
property = 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
property = 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 = property 1
```

```

HELP_TEXT = This property identifies the age of the user.
COMMENT_TEXT = You must provide a value for this property.
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 properties. They are located in `%DM_HOME%\bin` (`$DM_HOME/bin`). There is a file for each supported locale. The files are named with the following format:

```
data_dictionary_localesname.txt
```

where *localesname* is the name of the locale. For example, the data file for the German locale is named `data_dictionary_de.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 property

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 606](#).)

For details about using these statements, refer to 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 `publishDataDictionary` method or the PUBLISH keyword.

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` property of `dd` common info objects to determine which data dictionary objects need to be republished. If the property is set to `TRUE`, the job automatically publishes the data dictionary information for the object type or property 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 properties 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 `publishDataDictionary` method

The `publishDataDictionary` 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 property. For information about using this method, refer to the Javadocs.

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 property.

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 property and any other pending changes for that type or property.

If you do not 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 property
- The Data Dictionary Publisher job runs.
- An API method is executed to obtain data dictionary information about the changed object type or property.

System Events

This appendix lists and describes the system events recognized by Content Server. You can register most of these events for auditing. You can also use an `IdfSysobject.registerEvent` method to register to receive an Inbox notification for any of the DFC, workflow, or lifecycle events.

Those events which cannot be registered for auditing or notification are noted in their descriptions.

This appendix has the following topics:

- [dm_default_set event, page 609](#)
- [DFC events, page 610](#)
- [Workflow events, page 615](#)
- [Lifecycle events, page 618](#)
- [Events sent to the fulltext user, page 619](#)

dm_default_set event

This event name represents a set of events that are audited by default. [Table 95, page 609](#), lists the events in that default set.

Table 95. Events represented by the dm_default_set event

Object type	Audited events		
docbase config			
Note: The dm_default_set event is registered against the docbase config object; however, it causes auditing of the	dm_archive	dm_checkin	dm_restore
	dm_assemble	dm_checkout	dm_save
	dm_bp_attach	dm_destroy	dm_setfile
	dm_bp_demote	dm_freeze	dm_signoff
	dm_bp_promote	dm_link	dm_unfreeze

Object type	Audited events		
listed events for dm_document object type and its subtypes.	dm_bp_resume	dm_lock	dm_unlink
	dm_bp_suspend	dm_mark	dm_unlock
	dm_branch	dm_prune	

DFC events

Table 96, page 610, lists the events arising from DFC methods. (Events arising from methods that are specific to workflows are listed separately in Table 97, page 615.)

Table 96. Events arising from DFC methods

Event	Target object type	Event description	Trigger
dm_all (or all)	0 or omitted	All	All auditable events in repository
	dm_sysobject		Any event on any Sysobject or the specified Sysobject
dm_add_dynamic_group	dm_group	Add To Dynamic Group	Execution of an IDfSession.addDynamicGroups call
dm_adddigitalsignature	dm_sysobject	Add Digital Signature	Execution of the addDigitalSignature method.
			addDigitalSignature methods are always audited. It is not possible to unregister this event.
dm_addesignature	dm_sysobject	Add Electronic Signature	Execution of the addEsignature method.
			addEsignature methods are always audited and the entries are always signed by Content Server. It is not possible to modify this behavior.
dm_addesignature_failed	dm_sysobject	Add Electronic Signature Failed	An attempt to execute the addEsignature method failed. These failures are always audited. It is not possible to modify this behavior.

Event	Target object type	Event description	Trigger
dm_addnote	dm_sysobject dm_process	Add Note	Addition of a note to a SysObject. When the target object is a process object, the SysObject is in a workflow package.
dm_addrendition	dm_sysobject	Add Rendition	Addition of a rendition to an object.
dm_addretention	dm_sysobject	Add Retention	Object placed under control of a retention policy
dm_appendpart	dm_sysobject	Apend to Virtual Document	Addition of a component to a virtual document
dm_archive	dm_sysobject	Archive Objects	Execution of an archive method
dm_assemble	dm_sysobject	Assemble Virtual Document	Execution of assemble method
dm_assume	dm_user	Assume User	Execution of Assume method
dm_audit	all object types	Audit Event	Execution of method to register for auditing Registrations for auditing are always audited. It is not possible to unregister this event.
dm_authenticate	dm_user	Authenticate User	Execution of an authenticate method Executing an authenticate method also generates a dm_connect method if the authentication requires a repository connection.
dm_branch	dm_sysobject dm_retainer	Branch Version	Execution of a branch method
dm_checkin	dm_sysobject dm_retainer	Checkin Object	Checking in an object
dm_checkout	dm_sysobject dm_retainer	Checkout Object	Checking out an object

Event	Target object type	Event description	Trigger
dm_connect	dm_user	Logon	Establishing a repository connection Note: An authenticate method may also establish a repository connection, which will generate a dm_connect event in addition to the dm_authenticate event.
dm_destroy	dm_sysobject dm_acl dm_user dm_group dm_retainer	Destroy Object	Execution of a destroy method
dm_disassemble	dm_sysobject	Disassemble Virtual Document	Execution of a disassemble method
dm_disconnect	dm_user	Logoff	Terminating a repository connection
dm_fetch	dm_sysobject dm_retainer	Fetch Object	Fetching an object from the repository
dm_freeze	dm_sysobject	Freeze Object	Execution of a freeze method
dm_getfile	dm_sysobject	View/Export	Occurs when a document is viewed or exported or when the a getContent and getPath method is executed.
dm_getlogin	dm_user	Get Login	Execution of a getLoginTicket method
dm_insertpart	dm_sysobject	Insert into Virtual Document	Execution of an insertPart method
dm_install	dm_policy dm_process dm_activity	Install	Execution of an install method
dm_invalidate	dm_process dm_activity	Invalidate	Execution of an invalidate method
dm_kill	not applicable	Kill Session	Execution of a Kill method Note: You cannot audit a Kill method that flushes the cache.
dm_link	dm_sysobject	Link To	Execution of a link method
dm_lock	dm_sysobject	Lock Object	Execution of a lock method

Event	Target object type	Event description	Trigger
dm_logon_failure	dm_user	Logon Failure	User attempts to start a repository connection using invalid credentials.
dm_mark	dm_sysobject	Add Version Label	Execution of a mark method
dm_move_content	dmr_content	Move Content	Execution of a MIGRATE_CONTENT administration method
dm_prune	dm_sysobject dm_retainer	Prune Versions	Execution of a prune method
dm_purgeaudit	dm_audittrail dm_audittrail_acl dm_audittrail_group	Purge Audit	Execution of a destroy method or PURGE_AUDIT administration method to remove audit trail entries Purging an audittrail entry is always audited. It is not possible to stop auditing this event.
dm_readonlysave	dm_sysobject	no event description provided	Generated when Content Server changes a property value of an immutable object. This event is generated automatically for the fulltext user.
dm_remove_dynamic_group	dm_group	Remove From Dynamic Group	Execution of an IDfSession.removeDynamicGroups call
dm_removecontent	dm_sysobject	Remove Content	Execution of a removeContent method
dm_remove_note	dm_sysobject dm_process	Remove Note	Execution of a removeNote method
dm_removepart	dm_sysobject	Remove from Virtual Document	Execution of a removePart method
dm_remove_rendition	dm_sysobject	Remove Rendition	Execution of one of the removeRendition methods
dm_remove_retention	dm_sysobject	Remove Retention	Object removed from control of a retention policy
dm_restore	dm_sysobject	Restore Object	Execution of a restore method

Event	Target object type	Event description	Trigger
dm_save	dm_sysobject dm_acl dm_group dm_user dm_retainer	Save Object	Execution of a save method. Note: For new objects, Content Server stores the value 'Create' in the audit trail attribute string_1. For existing objects, the value in the attribute is 'Save'.
dm_saveasnew	dm_sysobject dm_acl dm_retainer	Copy Object	Execution of a saveAsNew method
dm_setfile	dm_sysobject	Set Content	Execution of the following methods: appendContent appendFile bindFile iInsertFile insertContent setContent setPath
dm_setoutput	dm_process	Setoutput	Execution of a Setoutput method.
dm_setretention-status	dm_retainer	Set Retention Status	Change to status of a retainer object.
dm_signoff	dm_sysobject	Signoff	Execution of a signoff method signoff methods are always audited. It is not possible to stop this auditing.
dm_unaudit	all object types	Unaudit Event	Removing an auditing registration. Methods that remove (unregister) an audit registration are always audited. It is not possible to stop this auditing.
dm_unfreeze	dm_sysobject	Unfreeze Object	Execution of an unfreeze method
dm_uninstall	dm_policy dm_process dm_activity	Uninstall	Execution of an uninstall method
dm_unlink	dm_sysobject	Unlink From	Execution of an unLink method
dm_unlock	dm_sysobject	Cancel Checkout	Execution of a cancelCheckOut method
dm_unmark	dm_sysobject	Remove Version Label	Execution of an unMark method

Event	Target object type	Event description	Trigger
dm_updatepart	dm_sysobject	Update in Virtual Document	Execution of an updatePart method
dm_validate	dm_policy dm_process dm_activity	Validate	Execution of a validate method

Workflow events

Table 97, page 615, lists the events specific to workflows. Note that several API events listed in the previous table are also applicable to workflows.

Table 97. Workflow events

Event	Target object type	Event description	Trigger
dm_all_workflow	dm_process (or not included)	All Workflow Events	All events on workflows generated from the specified process object or all events in the repository Note: This does not include dm_validate, dm_install, dm_invalidate, and dm_uninstall events.
dm_abortworkflow	dm_process	Abort Workflow	Aborting a workflow
dm_addattachment	dm_process	Add Attachment	An attachment is added to a running workflow or work item.
dm_addpackage	dm_process	Add Workflow Package	Execution of an addPackage method
dm_autotransactivity	dm_process	Automatic Workflow Activity Transition	An automatic activity transition occurs

Event	Target object type	Event description	Trigger
dm_changedactivityinstancestate	dm_process	Change Workflow Activity to Failed State	An automatic activity changes state because the error handling flag is set to zero and the work item returns a non-zero value.
dm_changepriority-workitem	dm_process	Change Workitem Priority	The priority value of a work item is changed at runtime.
dm_changestateactivity	dm_process	Change Workflow Activity State	An activity instance changes state to a state other than failed or changes state due to an automatic transition.
dm_changestateprocess	dm_process	Change Workflow Template State	A process definition changes state
dm_changestateworkflow	dm_process	Change Workflow State	A workflow changes state by a method other than a save, execute, or abort.
dm_changeworkflow-supervisor	dm_process	Change Workflow Supervisor	Execution of a Setsupervisor method
dm_completed-workitem	dm_process	Complete Workitem	Execution of a Complete method
dm_createworkflow	dm_process	Create Workflow	A workflow is created.
dm_delegated-workitem	dm_process	Delegate Workitem	A work item is delegated.
dm_finishworkflow	dm_process	Finish Workflow	A workflow is terminated.
dm_pauseworkitem	dm_process	Pause Workitem	A work item is paused.

Event	Target object type	Event description	Trigger
dm_portselect	dm_process	Select Workflow Port	Selection of output ports by a user or Content Server upon completion of an activity. Note: This event is not triggered if the activity has a transition type of prescribed.
dm_pseudocompletedworkitem	dm_process	Pseudo_Complete Workitem	A work item is marked as pseudo-completed by Content Server
dm_removeattachment	dm_process	Remove Attachment	An attachment is removed from a workflow or work item
dm_removepackage	dm_process	Remove Workflow Package	A package is removed from a workflow
dm_repeatworkitem	dm_process	Repeat Workflow Work Item	A work item is repeated.
dm_resumeworkitem	dm_process	Resume Workitem	A work item is resumed.
dm_save_workqueue	Not applicable	Not applicable	Generated when a workqueue object is saved by the Workqueue SBO. It is not possible to register for this event.
dm_save_workqueuepolicy	Not applicable	Not applicable	Generated when a workqueue policy object is saved by the Workqueue SBO. It is not possible to register for this event.
dm_selectedworkitem	dm_process	Select Workitem	A work item is acquired.
dm_startworkflow	dm_process	Start Workflow	A workflow is started.

Event	Target object type	Event description	Trigger
dm_startedworkitem	dm_process	Start Workitem	A work item is generated.
dm_suspended-workqueueetask	dm_process	Suspend Workqueue Tasks	A task on a workqueue is suspended.
dm_unsuspended-workqueueetask	dm_process	Unsuspend Workqueue Tasks	A task on a workqueue is unsuspended.
dm_wf_autodelegate_failure	dm_process	Auto Delegation Failed	Automatic delegation of an activity failed.

Lifecycle events

Table 98, page 618 lists the events specific to lifecycles.

Table 98. Lifecycle events

Event	Target object type	Event description	Trigger
dm_bp_attach	dm_sysobject dm_retainer	Attach Lifecycle	Attaching a lifecycle to an object
dm_bp_demote	dm_sysobject dm_retainer	Demote from Lifecycle State	Demoting an object from a lifecycle state
dm_bp_promote	dm_sysobject dm_retainer	Promote to Lifecycle State	Promoting an object to a lifecycle state
dm_bp_resume	dm_sysobject dm_retainer	Resume Lifecycle	Resuming an object's suspended lifecycle
dm_bp_suspend	dm_sysobject dm_retainer	Suspend Lifecycle	Suspending an object's lifecycle

Events sent to the fulltext user

The following events are generated automatically for the `dm_fulltext_user` user:

- `dm_checkin`
- `dm_destroy`
- `dm_move_content`
- `dm_readonlysave`
- `dm_save`

Plug-in Library Functions for External Stores

This appendix describes the C functions that you must implement to support Content Server operations through the plug-in. The following functions are described:

- [General recommendations](#) , page 621
- [dm_close_all](#), page 622
- [dm_close_content](#), page 622
- [dm_deinit_content](#), page 622
- [dm_init_content](#), page 623
- [dm_open_content](#), page 623
- [dm_plugin_version](#), page 625
- [dm_read_content](#), page 625

General recommendations

Call `dm_init_content` once when the plug-in is loaded. Call `dm_plugin_version` once after the plug-in is loaded.

Use `dm_open_content` once for each `getFile` or `getContent` operation. Use `dm_read_content` multiple times to read the content in 16k blocks.

Use `dm_close_content` once for each `dm_open_content` call.

Use `dm_close_all` once in a session, and call `dm_deinit_content` once before the plug-in is unloaded.

You can find sample code for a plug-in the unsupported directory.

dm_close_all

The `dm_close_all` function is called by the plug-in when a session is terminated. The function is called to let the plug-in library cleanup any internal data structure(s) for the specified session.

The function definition is:

```
void dm_close_all (long session)
```

[Table 99, page 622](#), lists its arguments.

Table 99. dm_close_all arguments

Argument	Description
<i>session</i>	Identifies the terminated session.

dm_close_content

The `dm_close_content` function is called by the plug-in to perform internal cleanup. This function is called after the read operation for the supplied handle is completed or if the read operation is interrupted. The function returns a boolean value.

The function definition is:

```
BOOL dm_close_content (long handle)
```

[Table 100, page 622](#), describes the argument for the function.

Table 100. dm_close_content arguments

Argument	Description
<i>handle</i>	Identifies the read request.

dm_deinit_content

The `dm_deinit_content` function performs global internal cleanup operations. The function is called just before the server or client unloads the plug-in library, to let the plug-in perform any global internal cleanup operations.

The function definition is:

```
void dm_deinit_content(void)
```

dm_init_content

The `dm_init_content` function initializes internal data structures. This is the first function called by Content Server after the plug-in library is loaded.

The function definition is:

```
BOOL dm_init_content (long maxsession,
int mode)
```

[Table 101, page 623](#), describes the arguments to this function.

Table 101. dm_init_content arguments

Argument	Description
<i>maxsession</i>	Contains the maximum number of concurrent sessions.
<i>mode</i>	Indicates who is invoking the plug-in. The only valid value is 0, indicating the server.

The function returns a positive value for successful initialization. A negative return value forces the server to unload the plug-in library. This function should return a positive value when called multiple times within the same address space.

dm_open_content

The `dm_open_content` function retrieves content.

The function definition is:

```
BOOL dm_open_content ( long session, char *other_args,
char *token, char *store_object_id, void *callback,
long *handle, long errcode)
```

[Table 102, page 623](#), describes the arguments for the function.

Table 102. dm_open_content arguments

Argument	Description
<i>session</i>	Indicates the session that needs to retrieve the content.
<i>other_args</i>	Indicates the <i>other_args</i> supplied when executing a Mount method. NULL for the <code>dm_extern_url</code> , <code>dm_extern_free</code> storage types and when the 'token' specified in a Setpath operation is an absolute path.
<i>token</i>	Is the path-translated token for which to retrieve the content.

Argument	Description
<i>store_object_id</i>	Indicates the external storage object ID.
<i>callback</i>	Is a function pointer that can be called by the plug-in library to retrieve an attribute value for the supplied external storage object ID.
<i>handle</i>	Identifies the read request. Filled in on initialization and passed for subsequent read operations.
<i>errcode</i>	Contains error code in case of failure.

The plug-in DLL or shared library returns a positive value for successful initialization and fills in a value for the handle. For subsequent read operations for this token, the handle value is passed. In case of failure, the plug-in fills in an error code in *errcode*.

This function is called when the server or client needs to retrieve the content for the token.

The handle enables the plug-in to be a multi-threaded application with each thread servicing a particular read request, with a dispatching mechanism based on the handle value. For example, for *dm_extern_file* store objects, *other_args* is the root path.

For client side plug-in configurations, if the Mount method has not been issued, the *other_args* parameter is a pointer to the directory location represented by the *def_client_root* attribute.

For server side plug-in configurations, *other_args* points to the directory represented by the *a_location* value for the current sever configuration. If no *a_location* is configured for the current server configuration, it points to the directory represented by the *def_server_root* attribute.

The call back function (which is part of server and client) is of the form:

```
char *dmPluginCallback (long session, char *store_object_id, char *attr_name, int position)
```

The call back function returns an attribute value in string form. The value for the position parameter should be zero when requesting an attribute value for an single-valued attribute and should be zero or greater for multi-valued attributes.

When this callback function is called for *DM_TIME* datatype attribute values, the returned string format is *mm/dd/yyyy hh:mi:ss*.

Plug-in libraries can define the function pointer type as follows:

```
typedef char * (*DCTMPLUGINCALLBACK) (long, char *, char *, int)
```

Cast the callback parameter to *DCTMPLUGINCALLBACK* before calling by reference.

Advanced plug-ins may start performing the actual read asynchronously and start caching the content for performance reasons.

dm_plugin_version

The `dm_plugin_version` function enables backward compatibility for enhancement in future releases. This function is called once immediately after the plug-in is loaded into the process address space. The plug-in protocol version is 1.0. Therefore, the plug-in must set 'major' to 1 and 'minor' to 0.

The definition of this function is:

```
void dm_plugin_version(unsigned int *major, unsigned int *minor)
```

[Table 103, page 625](#), describes the arguments to the function.

Table 103. dm_plugin_verson arguments

Argument	Description
<i>major</i>	Set to 1.
<i>minor</i>	Set to 0.

dm_read_content

The `dm_read_content` function requires the plug-in to return the content data into the location pointed to by buffer supplied.

The definition of this function is:

```
long dm_read_content ( long handle, char *buffer,  
long buffer_size, long *more_data, long *errcode)
```

[Table 104, page 625](#), describes the arguments to this function.

Table 104. dm_read_content arguments

Argument	Description
<i>handle</i>	Identifies the read request.
<i>buffer</i>	Contains the location to return the content data; filled with zeros when there is no more content to be read or end-of-file is reached.
<i>buffer_size</i>	Contains the buffer size (16k).
<i>more_data</i>	When positive, indicates more reading is required.
<i>errcode</i>	Contains error code in case of failure.

The function returns the number of bytes read and filled into buffer.

The plug-in must maintain its own internal bookkeeping to start reading from the next byte after the previous read.

A

- a_archive property, 280
- a_content_attr_name
 - dm_storage_strategy setting, 253
- a_content_static property, 245
- a_current_status property, 144
- a_default_retention_date property, 247, 257
- a_last_return_code property, 144
- a_next_invocation property, 151
- a_retention_attr_name property, 248, 257
- a_retention_attr_required property, 247, 257
- a_silent_login property, 351
- a_storage_params property, 224
- a_storage_type property, 230
 - dm_turbo_store value, 224
- accepting server broadcasts, 193
- access control entries
 - described, 369
- access control lists, *see* ACLs
- access permission ACL entries, 370
- access restriction ACL entries, 370
- acl_class property, 379
- acl_domain property, 381
- acl_name property, 381
- acl_update_threshold (server.ini key), 98
- ACLs
 - ACL object overview, 369
 - acl_domain property, 381
 - acl_name property, 381
 - acl_update_threshold (server.ini key), 98
 - adding entries, 384
 - aliases in, 379
 - audit trail entries, interpreting, 413
 - caching, 100
 - changing
 - object name, 379
 - owners, 379
 - type default, 74
 - consistency checks, 553
 - constraint on aliases, 380
 - creating, 380
 - default, 383
 - default, assigning, 383
 - default, defining, 381
 - deleting
 - external, 385
 - from repository, 385
 - internal, 76, 385
 - described, 360
 - entries, described, 369
 - entry evaluation, 374
 - evaluation for object owners, 375
 - external, 378
 - folder default, 382
 - internal, 378
 - macl_security_disabled property, 377
 - modifying, 384
 - multiple entries for one user, 376
 - naming conventions, 379
 - object name, 381
 - object types, assigning to, 382
 - orphaned, 76
 - owner of, 381
 - owner_xpermit_default (server.ini key), 101
 - private, 379
 - public, defined, 379
 - removing entries, 384
 - removing orphans, 482
 - specifying server-level default, 383
 - system, defined, 379
 - TCS license and evaluation, 374
 - templates, 379
 - Trusted Content Services license and, 380, 384

- type default, 382
 - user default, 382
 - using dmclean to delete, 385
- ACS servers, 86
- activating
 - administration jobs, 453
 - jobs after loading, 60
- Active Directory
 - mode requirements, 342
 - user_name mapping requirement, 336
- addDigitalSignature method
 - auditing, 391
- addDynamicGroup method, 311
- addESignature method
 - tracing, 427
- Addesignature method
 - auditing, 391
 - parameters passed by, 425
- admingroup group, 302, 310
- Administration Encryption Key, *see* AEK (Administration Encryption Key)
- administration methods
 - MIGRATE_CONTENT, 259
 - PURGE_AUDIT, 416
 - RECOVER_AUTO_TASKS, 125
- administration, server
 - Content Server Manager, 32
 - Documentum Administrator, 32
 - DQL, 32
 - DQL statements, 33
 - privileges required, 31
 - tool suite, 33
 - tools, 446
- AEK (Administration Encryption Key)
 - audit trail entries and, 394
 - backing up, 431
 - described, 430
 - determining existence, 433
 - passphrase, changing, 434
 - putting in shared memory, 432
 - sharing among products, 430
 - single-repository distributed configurations, 431
- agent exec process, 148
 - behavior modification, 154
 - database_refresh_interval (server.ini key) and, 154
 - described, 85, 449
 - disabling all jobs, 153
 - jobs in polling cycle, setting maximum, 154
 - log file location, 155
 - max_concurrent_jobs, 154
 - override_sleep_duration, 154
 - polling interval, default, 86, 154
 - removing log files, 499
 - tracing, turning on, 155
- agent_exec_method method, 85
- agent_launcher property, 85, 153
- alias sets
 - creating, 72
 - deleting, 72
 - described, 71
 - modifying, 72
- aliases
 - ACL template use, 379
 - dm_dbo, 73
 - mounted directories, 66
 - repository owner, 73
 - use constraint in ACLs, 380
- _all_users_names (computed property), 311
- allow_trusted_login (dfc.properties key), 166
- ALTER TYPE statement for modifying types, 74
- annotations
 - dmclean argument for, 483
 - removing orphans, 482
- annotations, removing orphans, 76
- API (Application Program Interface)
 - IAPI utility, 568
- append_to_body property, 424
- application access control tokens
 - dfc.machine.id key (dfc.properties file), 439
 - enabling use, 439
 - retrieval from storage, enabling, 440
 - storage location, 443
 - token files, 441
 - troubleshooting, 444
- application codes, 364
- application events, 388, 393
 - audit trails, viewing, 396
- application permission ACL entries, 371
- application restriction ACL entries, 372
- application server, *see* JBoss application server
- application_access_control property, 439

- applications
 - application events, auditing, 393
 - application permits in ACLs, 371
 - application restrictions in ACLs, 372
 - archive method, 278
 - authentication plug-ins,
 - specifying, 347
 - connection brokers and, 199
 - enabling/disabling client caching, 178
 - file extensions, 258
 - firewalls and, 194
 - is_private property, usage, 308
 - Mount method, 243
 - objects, controlling, 361
 - plugin authentication, using, 347
 - private and public groups, 307
 - restore method, 278
 - shutting down servers with, 119
 - URLs, using, 234
- approved_clients_only property, 169
- archive directory, 282
- archive tool
 - archive directory, choosing, 285
 - described, 280
 - generated dump record, 281
 - generated load record, 282
 - repository operator, 284
 - verbose option, 281
- Archive tool, 454
- archiving
 - archive directory, choosing, 285
 - archive tool, 280
 - content files, moving, 262
 - content used in many documents, 283
 - generated dump record, 281
 - generated load record, 282
 - overview, 277
 - post_archive procedure, 283
 - post_restore procedure, 283
 - pre_restore procedure, 283
 - re-archiving restored documents, 286
 - repository operator, 284
 - setting up, 285
 - strategies, 283
- arguments passed to methods, 151
- ascii property, 236
- ascii property for blob store type, 217
- aspects
 - dump operations and, 51
- asset_class property, 70
- assume user program, 315
- assume_user_location property, 315
- attachments
 - package control and, 80
- Audit Management tool, 456
 - privileges to run, 416
- Audit method, 394
 - auditing execution of, 391
- audit trail
 - defined, 388
- audit trail entries
 - lifecycle, 404
 - querying/retrieving, 397
 - removal, auditing of, 391
 - removing, 456
 - signatures, verifying, 415
 - signing, 394
- audit trails
 - common use properties, 397
 - generic properties, 397
 - interpreting, 397
 - removing, 416
 - viewing, 396
- audit trial entries
 - ACLs, 413
 - groups, 413
 - workflow, 405
- audit_old_values property, 388
- auditing, 387
 - addDigitalSignature method, 391
 - Addesignature method, 391
 - application events, 393
 - Audit Management tool, 456
 - Audit method auditing, 391
 - audit trail, 388
 - audit trail entries
 - querying/retrieving, 397
 - audit trail entries, signing, 394
 - audit trails
 - interpreting, 397
 - removing, 416
 - audit trails, viewing, 396
 - common use properties, 397
 - compound documents, 417
 - conflicting registrations,
 - resolving, 395
 - default auditing, 391
 - distributed configurations, 416
 - dm_default_set event, 392
 - dm_purgeaudit events, 391

- dmi_registry objects and, 393
 - events, 387
 - failed logins, 392
 - IDfAuditTrailManager interface, 396
 - privileges to stop, 396
 - property values, 388
 - signatures, verifying, 415
 - Signoff method, 391
 - system events, 393
 - Unaudit method auditing, 391
 - audittrail attrs objects
 - removing, 416
 - auth config objects, 322
 - auth_protocol property, 318
 - unix_domain_used setting, 322
 - authentication, 328
 - See also* user authentication
 - auth_protocol property, 318
 - authentication plug-ins, 314
 - CA SiteMinder Policy Server, 315
 - dm_check_password file, 316
 - domain-required mode, 318
 - domains, affect of, 317
 - failed attempts, limiting, 356
 - failure auditing, 392
 - no-domain required mode, 318
 - password checking program,
 - creating, 319
 - plug-in modules
 - described, 346
 - server domain, 108
 - trusted logins, 351
 - unified login, 351
 - UNIX default mechanism, 316
 - UNIX users in domains, 320
 - user names, 108
 - user, options, 313
 - user_login_name property, 290
 - Windows default mechanism, 317
 - authentication modules
 - plug-in identifiers, 347
 - authentication
 - in-line passwords, using, 351
 - auto_request_forward (dfc.properties key), 191
 - automatic activities
 - execution
 - configuring, 123
 - disabling, 124
 - recovering work items on Content Server failure, 125
 - automatic operations
 - auditing failed authentication, 392
 - file clean up, 176
 - file extensions, 258
- ## B
- backslashes, in repository connection
 - file, 156
 - base_url property, 238
 - basic user privileges, 291
 - bind_type property, 334
 - binding modes for LDAP, 334
 - blob storage areas
 - ascii property, 217, 236
 - content location, identifying, 224
 - described, 217
 - maximum content size, 217
 - naming rules, 236
 - setting up, 236
 - bp_schedule_*.log files, 499
 - bp_transition_*.log files, 499
 - broadcasts, connection brokers
 - accepting, 193
- ## C
- C-clip buffer size, for Centera stores, 214
 - C-clip buffer, setting size, 252
 - CA certificates, installing, 340
 - CA SiteMinder
 - plug-in authentication module, 348
 - CA SiteMinder Policy Server, 315
 - ca store object type, 212
 - cabinets
 - consistency checks, 555
 - Create Cabinet user privilege, 292
 - creating, 43
 - default ACL, 383
 - defined, 42
 - editing, 43
 - home, 43
 - permissions to modify/delete, 44
 - private, 43
 - public, 43
 - user default, 298
 - cache config objects
 - automating validation, 182

- client check interval, setting, 181
 - creating, 179
 - server check interval, setting, 180
- cache_element_queries property,
 - setting, 179
- cache_element_types property,
 - setting, 179
- caches
 - database_refresh_interval server.ini
 - key, 104
 - dump process, 55
 - persistent client, 177
- CD-ROM files, external storage and, 220
- certdb_location property, 335
- certificate authorities
 - obtaining, 340
 - obtaining list of, 345
- certificate database
 - contents, listing, 345
 - installing, 340
 - location of, 335
- certutil utility, 340
- change password program, 316
- change_location permission, 366
- change_owner permission, 366
- change_password_location property, 316
- change_permit permission, 367
- change_state permission, 367
- checkpoint interval
 - described, 114
 - setting, 116
- checkst_client_version property, 168
- cleaning up file storage areas, 266
- client caches
 - flushing, 181
- client caching
 - enabling/disabling, 178
- client capability setting, 299
- client configuration
 - connection pooling, 172
 - local areas
 - disk space limit, 176
 - file clean up, 176
 - file naming conventions, 177
 - purgeLocalFiles method, 177
 - search_order (dfc.properties key), 191
 - short date formats, 175
 - short date formats, defining, 175
- client local area
 - changing location, 176
- client local areas
 - disk space limit, setting, 176
 - file clean up, 176
 - file naming conventions, 177
 - local_purge_on_diskfull
 - (dfc.properties key), 176
 - purgeLocalFiles method, 177
- client platform
 - defined, 162
- client rights objects
 - creating, 168
- client_check_interval property, 181
- client_pcaching_change property, 182
- client_pcaching_disabled property, 178
- client_session_timeout (server.ini
 - key), 103
- clients, 201
 - client_session_timeout (server.ini
 - key), 103
 - connection brokers and, 190
 - defined, 162
 - finding connection brokers, 200
 - getDocbrokerMap method, 200
- cluster environments
 - upd_last_chg_time_from_db
 - (server.ini key), 102
- code page requirements
 - user names, 290
- code pages
 - authentication modules, use in, 350
 - default settings, 87
 - dump and load requirements, 48
- collections
 - Kill method and, 120
- commas, in repository connection file, 156
- comments
 - IAPI format, 573
 - IDQL format, 567
- commit_read_operations (server.ini
 - key), 99
- common area
 - removing files from, 122
- compound documents
 - auditing, 417
- computed properties
 - _all_users_names, 311
- concurrent_sessions
 - login tickets and, 173
- concurrent_sessions, setting value in
 - server.ini file, 103

- Config Audit privileges, 396
- Config Audit user privileges, 293
- configuration
 - changing server configuration, 109
 - DocBrokers, 196
 - objects, 29
 - overview, 27, 29
 - session config objects, 174
- connect_retry_interval (dfc.properties), 167
- connected users, viewing, 34
- connection broker
 - monitoring status, script for, 590
- connection broker locator object
 - described, 189
- connection brokers
 - adding additional, 196
 - api config properties for, 201
 - auto_request_forward (dfc.properties key), 191
 - clients and, 190
 - communicating with, 114
 - configuration options, 189, 191
 - defining targets in server config, 115
 - deleting server information, 201
 - failover, 164, 191
 - finding, 200
 - firewalls and, 194
 - Getdocbrokermap method, 189
 - getDocbaseMap method, 200
 - getServermap method, 200
 - information about, 200
 - initialization file, 192
 - invoking initialization file, 193
 - load balancing, 164, 191
 - log files, 79
 - number of, 188
 - overview, 187
 - projection targets, 114
 - removing log files, 499
 - requesting information from, 199
 - restricting server access, 193
 - security, 192, 198
 - server shutdown notification, 120
 - servers and, 85, 189
 - setting checkpoint interval for servers, 116
 - setting keep_entry interval for servers, 116
 - shutdown security, 193
 - specifying for sessions, 163
 - specifying IP address for, 192
 - starting, 195
 - stopping, 197
 - stopping multiple, 199
 - translating IP addresses, 194
- connection mode
 - server-side, setting, 110
- connection pooling
 - dmbasic method server, 130
 - enabling, 172
- connection requests
 - gethostbyaddr (server.ini key), 100
- connection strings
 - DB2, 97
 - MS SQL Server, 97
 - Oracle, 96
 - Sybase, 97
- connection strings for EMC Centera clusters, 249
- connection strings, for EMC Centera store storage areas, 248
- connections
 - concurrent_sessions setting, 103
- Consistency Checker tool
 - described, 459
 - report sample, 460
- consistency checking rules
 - overriding, 183
- consistency_checker.ebs, 460
- content assignment policies
 - administering, 229
 - behavior on error, configuring, 227, 276
 - creating, 225
 - described, 225
 - DFC cache for, 227, 277
 - enabling/disabling, 276
 - enforcement, 226
 - logging use, 275
 - repository representation, 227
 - storage algorithm, 228
- content compression, 206
 - constraint on use in distributed stores, 207
 - dump files, for, 55
- content directories
 - creating, 65
- content duplication checking
 - behavior, 209

- constraint on use in distributed stores, 209
 - filestores, in, 209
 - hash format, 210
 - Macintosh constraint, 209
 - supporting properties, 209
- content files
 - accessing in load operations, 55
 - archiving content used in many documents, 283
 - archiving/restoring overview, 277
 - auditing movement, 263
 - automatic file extensions, 258
 - clean up in local areas, 176
 - clean up in repository, 76
 - compression in EMC Centera storage areas, 213
 - compression in NetApp SnapLock storage areas, 217
 - compression in storage, 206
 - content assignment policies, 225
 - content migration policies, 260
 - Content Warning tool, 468
 - default storage algorithm, 229
 - deleting from EMC Centera storage area, 265
 - digital shredding, 211
 - dos_extension property and, 68
 - dump files and, 54
 - dumping encrypted, 54
 - file extensions in storage, 237
 - File Report tool, 493
 - hash values, 210
 - moving, 259
 - naming convention in local areas, 177
 - naming conventions in file stores, 233
 - r_content_hash property, 210
 - removing from EMC Centera storage, 268
 - removing from server common area, 122
 - removing orphan files, 266, 487
 - retention of, 231
 - storage algorithm
 - primary, 229
 - renditions, 231
 - storage options, 204
 - storing in turbo store, 257
 - use_extensions property, 234, 258
- content migration policies
 - configurable arguments, 261
 - described, 260
 - log file location, 262
- Content Migration Thread option, 259
- content objects
 - consistency checks, 557
 - defined, 223
 - i_content_id property and, 266
 - is_archived property, 280
 - is_offline property, 280
 - r_content_hash property, 210
 - removing orphans, 266, 482
 - set_file property, 280
 - turbo storage and, 218
- Content Replication tool
 - arguments, 466
 - described, 465
 - report sample, 467
- Content Server
 - agent exec process, 85, 449
 - audit trails
 - interpreting, 397
 - auditing
 - application events, 393
 - privileges to stop, 396
 - auditing, default, 391
 - changing configuration, 109
 - code page, 87
 - common area, removing file from, 88
 - common area, removing files from, 122
 - communicating with connection brokers, 189
 - configuration, 84
 - configuration requirements for additional, 112
 - connecting to by host name, 165
 - connecting to by name, 165
 - connecting to specific server on specific host, 165
 - connection brokers and, 85
 - connection strings
 - DB2, 97
 - MS SQL Server, 97
 - Oracle, 96
 - Sybase, 97
 - data dictionary
 - populating, 593
 - database connection string, 96

- database_refresh_interval (server.ini key), 104
- default_acl, setting, 383
- defining projection targets, 114, 116
- deleting from connection broker
 - list, 201
- described, 84
- distinct_query_results (server.ini key), 104
- failover, 121
- getServerMap method, 200
- historical session cutoff, 100
- history_cutoff (server.ini key), 100
- history_sessions (server.ini key), 100
- index_store (server.ini key), 99
- internationalization, 87
- java.ini file, 134
- LDAP connection credentials, 334
- LDAP directory servers, multiple, 342
- listener queue length,
 - configuring, 118
- load balancing, 121
- max_ftacl_cache_size (server.ini key), 100
- monitoring status, script for, 590
- moving executable (UNIX), 109
- notifying connection broker of shutdown, 120
- parent and session servers, 84
- proximity values, defining, 117
- removing server logs, 499
- removing session logs, 499
- restarting, 111
- server.ini file, 88
- server_startup_sleep_time key, 106
- sessions, shutting down, 120
- setting checkpoint interval, 116
- setting keep_entry interval, 116
- shutting down, 118
- specifying listening address, 98
- standard arguments passed to
 - jobs, 449
- start_index_agents (server.ini key), 107
- starting, 88
- starting additional, 112
- threads, 84
- tracing options, 532
- version number, 514
 - viewing historical sessions, 100
- Content Server Manager, 32
- Content Servers
 - secure connection mode, setting, 110
- content storage areas
 - moving files, 259
- Content Storage Services license, 225
- content type
 - parent_id property, 223
 - storage_id property, 223
- Content Warning tool
 - described, 468
 - percent_full argument, 469
 - report sample, 469
- content_dupl_pref property, 210
- content-file servers
 - failover limitation, 122
- content_hash_mode property, 210
- ContentServerStatus script, 590
- copying
 - objects, folder security and, 363
- Create Cabinet user privilege, 292
- Create full-text events tool
 - report sample, 474
- Create Group user privilege, 292
- Create method, 65
- Create Type user privilege, 292
- CREATE...GROUP (statement), 308
- CREATE...OBJECT (statement)
 - creating format objects, 69
 - creating jobs with, 149
 - creating location objects, 65
- CREATE...TYPE (statement), 73
- createAudit() method, 394
- creating
 - blob storage areas, 236
 - default ACLs, 383
 - directories, 65
 - dump record object, 50
 - format objects, 69
 - jobs, 149
 - linked storage areas, 239
 - location objects, 64
 - method objects, 143
 - object types, 72
 - objects, folder security and, 363
 - repositories, 39
- creating full-text events, 470
- custom ACLs. See internal ACLs, 378

D

- data dictionary
 - consistency checks, 559
 - Data Dictionary Publisher job, 607
 - Data Dictionary Publisher tool, 475
 - 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 properties with population file, 597
 - setting foreign keys, 601
 - populating Japanese/Korean locales, 605
 - populating using DQL, 606
 - PUBLISH keyword, 608
 - publishDataDictionary method, 607
 - publishing, 606
- data dictionary data files
 - described, 595
 - examples, 602
 - settable properties, 597
 - structure of, 595
 - using multiple, 595
- Data Dictionary Publisher, 607
- Data Dictionary Publisher tool, 475
- data ticket number, 234
- data_dictionary.ini, 606
- data_store (server.ini key), 99
- database connection strings
 - DB2, 97
 - described, 96
 - MS SQL Server, 97
 - Oracle, 96
 - Sybase, 97
- Database Space Warning tool, 476
- database_conn server.ini key, 96
- database_name server.ini key, 97
- database_owner server.ini key, 96
- database_refresh_interval (server.ini key), 154
- date formats, short, 175
- dates
 - storage, configuring, 45
 - time zone adjustment, 45
- DB2
 - database connection strings, 97
 - object type table location, 45
 - object type tables
 - device, defining, 94
 - uninstalling repository, affects of, 100
- dcf_edit utility, 156
- dd_locales
 - setting, 44
- dd_populate.ebs, 606
 - described, 594
 - executing, 605
- deactivating
 - jobs, 153
 - jobs during load, 60
- default ACLs
 - assigning to objects, 383
 - cabinet, 383
 - changing type default, 74
 - creating, 383
 - default_acl property and, 383
 - described, 381
 - folder, 382
 - specifying at server level, 383
 - type default, 382
 - user, 382
- default storage algorithm for content, 229
- default values
 - changing default group, 304
 - public and private groups, 307
 - storage area for types, 74
- default_acl property, 383
- default_app_permit property, 364
- default_folder property, 298
- default_retention_days property, 247, 257
- default_storage property
 - format object property, 230
 - type info object property, 230
- default_storage property, for formats, 70
- deferred object update queue, setting
 - size, 107
- deferred update process, 107
- deferred_update_queue_size (server.ini key), 107
- delete_object permission, 367
- deleting
 - ACLs, 385
 - dequeued items, 503
 - format objects, 71
 - groups, 309

- log files, 499
- objects, folder security and, 363
- properties from types, 75
- renditions, 509
- users, 304
- delimiters
 - signature page defaults, changing, 421
- dequeued items, deleting, 78, 503
- Destroy method
 - automating with version management tool, 525
 - folder security and, 363
- DFC (Documentum Foundation Classes), 537
 - See also* DFC tracing
 - trace file location, default, 538
 - trace file size, default, 538
- DFC (Documentum Foundation Classes), 168, 536
 - See also* DFC tracing; privileged DFC
 - content assignment policy enforcement, 226
 - determining tracing status, 536
 - dump and load interfaces, 49
 - log files, 536
 - trace file naming, 539
 - tracing, 536
 - user privileges, setting, 303
- DFC events
 - audit trail entries, 398
- DFC tracing
 - dates, defining alternate format, 542
 - dates, defining column width, 542
 - default file location, 538
 - default file size, 538
 - determining activation status, 536
 - dfc.tracing.file_creation_mode, 544
 - dfc.tracing.thread_name_filter, 545
 - enabling, 537
 - exception stacks, printing, 546
 - file_creation_mode, defining, 539
 - log4j.properties and, 537
 - logger, 537
 - logging appender, 537
 - method calls, 542
 - method entry and exit, 543
 - RPC calls, tracing, 546
 - session information, including, 545
 - specifying package, class, method to trace, 543
 - timestamp format, setting, 540
 - trace file format, 548
 - trace file names, 539
 - users, filtering by name, 544
 - verbosity level, 546
- dfc.config.check_interval key, 162
- dfc.data.umask (dfc.properties), 175
- dfc.keystore file, 171
- dfc.machine.id key, 439
- dfc.privilege.enable key, 170
- dfc.properties
 - connect_retry_interval, 167
 - dfc.session.max_count key, 167
 - dfc.tracing.print_exception_stack, 546
 - directory specifications, 163
 - max_connect_retries, 167
 - modifying, 163
 - secure_connect_default, 166
- dfc.properties file
 - connection brokers, specifying, 163
 - described, 162
 - dfc.config.check_interval key, 162
 - dfc.data.umask, 175
 - dfc.session.pool.enable key, 172
 - dfc.storagepolicy.ignore_rule_error key, 276
 - dfc.tokenstorage.dir key, 440, 443
 - dfc.tokenstorage.enable key, 440
 - dfc.tracing.date_column_width, 542
 - dfc.tracing.date_format, 542
 - dfc.tracing.enable, 537
 - dfc.tracing.file_creation_mode, 539, 544
 - dfc.tracing.include_rpc_count, 546
 - dfc.tracing.include_rpcs, 546
 - dfc.tracing.include_session_id, 545
 - dfc.tracing.method_name_filter, 543
 - dfc.tracing.mode, 543
 - dfc.tracing.thread_name_filter, 545
 - dfc.tracing.timing_style, 540
 - dfc.tracing.user_name_filter, 544
 - dfc.tracing.verbosity, 546
- dfcfull.properties file, 162
- enable_persistence d key, 178
- extern_store_content_static key, 245
- force_coherency_check key, 183
- keu format, 162

- local_clean_on_init key, 177
- local_diskfull_limit key, 176
- local_purge_on_diskfull key, 176
- max_stack_depth, 542
- search_order key, 191
- write_interval key, 183
- .write_max_attempts key, for Centera stores, 253
- write_max_attempts key, for EMC Centera stores, 253
- dfc.session.pool.enable (dfc.properties), 172
- dfc.tokenstorage.dir key, 440, 443
- dfc.tokenstorage.enable key, 440
- dfc.tracing.date_column_width, 542
- dfc.tracing.date_format, 542
- dfc.tracing.enable, 537
- dfc.tracing.file_creation_mode, 539
- dfc.tracing.include_rpc_count, 546
- dfc.tracing.include_rpcs, 546
- dfc.tracing.include_session_id, 545
- dfc.tracing.timing_style, 540
- dfcfull.properties file, 162
- digital shredding
 - described, 211
- digital signatures, 427
- dir_user_sync_on_demand property, 328
- directories
 - archive, 282
 - creating, 65
 - dfc.data.umask (dfc.properties), 175
 - format objects and, 39
 - fulltext index objects and, 39
 - location objects and, 38
 - mount point objects and, 39
 - mounted drive aliases, 66
 - storage objects and, 39
 - temp, 78
 - umask server.ini key, 101, 109
- directory paths
 - archive dump file, 280
 - content in file stores, 223, 233
 - dump file, 59
 - State of the Repository Report tool, 514
- directory specifications, in
 - dfc.properties, 163
- disabling
 - content assignment policies, 276
 - policy engine (DFC), 276
- disk space management
 - Content Warning tool, 468
 - Database Space Warning tool, 476
 - Swap Info tool, 518
- distinct_query_results
 - server.ini key, 104
- DistOperations job, 447
- distributed content
 - dmclean tool and, 485
 - dmfilesca tool and, 490
- distributed environment and site-specific tool reports, 451
- distributed environments
 - signatures, verifying, 415
- distributed repositories
 - auditing in, 416
- distributed storage areas
 - creating, 236
 - described, 218
 - load operations and, 60
 - valid component types, 218
- distributed store storage type
 - r_component property, 224
- distributed stores
 - components, replacing, 274
- DM_ARCHIVE event, 278
- dm_Archive job, 454
- dm_AuditMgt job, 456
- dm_audittrail_acl objects, 392
- dm_audittrail_group objects, 392
- dm_auth_config objects, 322
- dm_autorender_mac user, 41
- dm_autorender_win31 user, 41
- dm_bpm_inbound_user user, 41
- dm_browse_all group, 294
- dm_browse_all_dynamic group, 294
- dm_check_password file, 316
- dm_CheckCacheConfig method, 182
- dm_ConsistencyChecker, 459
- dm_ContentReplication job, 465
- dm_ContentWarning job, 468
- dm_create_cabinet group, 295
- dm_create_group group, 295
- dm_create_type group, 295
- dm_create_user group, 295
- dm_crypto_boot utility, 432
- dm_crypto_change_passphrase utility, 434
- dm_crypto_create
 - troubleshooting AEK with, 433

- DM_CRYPTO_FILE environment
 - variable, 430
- dm_dbo alias, 73
- dm_DBWarning job, 476
- DM_DEBUG_ESIGN_METHOD environment variable, 427
- dm_default_set event, auditing, 392
- dm_DMclean job, 482
- dm_DMFilescan job, 487
- dm_domain_conv script, 319
- DM_ENCR_PASS prefix, 353
- dm_encrypt_password utility, 354
- dm_error utility, 34
- dm_FileReport job, 493
- dm_FTCreateEvents, 470
- dm_fulltext_index_user, 41
- DM_GROUP_LIST_LIMIT environment variable, setting, 79
- dm_GroupRename tool, 498
- DM_HOME_CURRENT environment variable, 88, 109
- dm_job_sequence objects, 147
- dm_launch_docbroker, 195
- dm_LDAPSynchronization job, 479
 - activating, 341
 - behavior on first run, 344
 - choosing LDAP servers, 327
 - described, 325
 - indexed LDAP properties, use of, 334
 - LDAPSynchronizationDoc.txt file, 337
 - on-demand execution, 328
 - properties set, 326 to 327
 - trace files, 345
- dm_LDAPSynchronization tool, 479
- dm_LogPurge job, 499
- dm_mediaserver user, 41
- dm_move_content event, 263
- dm_MoveContent
 - configurable arguments, 261
- dm_MoveContent job
 - described, 260
- dm_netegrity, 349
- DM_PLUGIN prefix, 347
- dm_QueueMgt job, 503
- dm_relation_ssa_policy objects, 227
- dm_RenditionMgt job, 509
- DM_RESTORE event, 278
- dm_retention_managers group, 294
- dm_retention_users group, 294
- dm_run_dependent_jobs
 - behavior, 147
- dm_run_dependent_jobs method
 - arguments, 159
 - introduced, 146
 - return values, 147
- dm_save events
 - load operations and, 60
- DM_SERVER trace facility, 211
- dm_shutdown_repository script
 - creating server_specific, 113
 - shutting down servers with, 119
- dm_ssa_policy objects, 227
- dm_start_repositoryname script
 - oclean argument, 88
 - starting servers with, 88
- dm_StateofDocbase job, 514
- dm_stop_docbroker utility, 198
- dm_storage_strategy, a_content_attr_name value, 253
- dm_store type
 - r_status property, 264
- dm_superuser group, 294
- dm_superuser_dynamic group, 294
- dm_SwapInfo job, 518
- dm_sysadmin group, 295
- dm_UpdateStats job, 519
- dm_UserChgHomeDb job, 522
- dm_UserRename job, 523
- dm_VersionMgt job, 525
- dmathplug.h header file, 350
- dmathplug.lib (dmathplug.a) file, 350
- dmbasic executables, 138
- dmbasic method server
 - connection pooling, 130
 - described, 130
 - enabling, 135
 - log files, removing, 499
 - worker threads, configuring, 136
- dmclean utility
 - arguments, 267
 - deleting internal ACLs, 385
 - described, 266 to 267
 - Dmclean tool, 482
 - EMC Centera storage areas and, 268
 - EXECUTE syntax, 268
 - executing generated script, 269
 - external storage and, 219
 - generated script sample, 269
 - standalone syntax, 269

- dmdocbroker.exe, 195
- dmfilesan utility
 - arguments, 270
 - described, 266, 270
 - Dmfilesan automated tool, 487
 - EXECUTE syntax, 273
 - executing generated script, 274
 - external storage and, 219
 - force_delete argument, 272
 - generated script example, 273
 - index use, 272
 - standalone syntax, 273
 - subdirectories to scan,
 - identifying, 271
- dmi_audittrail_attrs
 - privileges to query, 397
- dmi_queue_item objects, removing, 78
- dmqdocbroker script, 590
- dmtkgen utility
 - arguments, 441
 - output file, 441
 - output file, naming, 443
- DO_METHOD procedures, 129
 - See also* methods
 - creating method objects, 143
 - Java method server and, 131
 - post_archive, 283
 - post_restore, 283
 - pre_restore, 283
 - tracing, 133
- do_method servlet, 131
- DocApps
 - dump operations and, 64
 - dumping limitation, 51
- docbase config object
 - dd_locales, setting, 44
 - described, 38
 - wf_package_control_enabled
 - property, 80
- docbase config objects
 - check_client_version property, 168
 - macl_security_disabled property, 377
 - oldest_client_version property, 168
 - security_mode property, 362
- docbase configuration
 - trust_by_default property, 437
 - trusted_docbases property, 437
- docbase_id server.ini key, 96
- docbase_name server.ini key, 96
- Docbasic
 - java guidelines, 137
 - tool suite implementation, 449
- Docbasic argument length, maximum, 137
- docbroker locator object
 - getDocbrokerMap method, 201
- DOCBROKER_CONFIGURATION
 - section, 196
- DocBrokers
 - configuration, 196
- docu group, 61
- documents
 - a_archive property, 280
 - archiving/restoring overview, 277
 - consistency checks, 556
 - defining signature page templates
 - for, 423
 - re-archiving restored documents, 286
 - restoring, 495
 - restoring from archive, 286
 - restoring from archive with surrogate
 - get, 287
 - retention of, 231
 - setting up archiving, 285
- Documentum
 - creating repositories, 39
 - repository security levels, 362
 - security overview, 359
 - tool suite, 446
 - user privileges, 291
- domain controller maps, 322
- domain-required mode
 - converting to, 319
 - described, 318
- domains
 - auth_protocol property, 318
 - authenticating in, 317
 - authenticating UNIX users in, 314, 320
 - authentication, 108
 - dm_domain_conv script, 319
 - proximity values and, 118
- dos_extension property, 68
- DQL (Document Query Language)
 - auditing and, 393
 - creating method objects, 144
 - distinct_query_results (server.ini key), 104
 - format object deletion, 71
 - format object modification, 70

- NOFTDQL hint and distinct_query_
 - results server.ini key, 104
 - type modification procedures, 74
 - DROP TYPE (statement), 76
 - dump and load
 - table permits, affect on, 387
 - dump files
 - code pages and, 48
 - content files and, 54
 - dump object record object, 49
 - dump record objects
 - described, 49
 - dum_operaton property, 51
 - include_content property, 55
 - specifying
 - predicate property, 53
 - predicate2 property, 53
 - type property, 52
 - dumping
 - compressing content, 55
 - content files, 55
 - DocApps limitation, 51
 - dump_operation property, 51
 - external storage, 219
 - full_docbase_dump, 51
 - job objects, 60
 - non-restartable, 56
 - objects under retention, 50
 - objects with aspects, 51
 - partial, 51
 - repositories, 48, 64
 - sample script, 57
 - script, sample for full dump, 57
 - script, use of, 56
 - selecting objects, 53
 - specifying object types, 52
 - troubleshooting, 59, 64
 - Web Publisher repositories, 48
 - duplicate rows in queries, 104
 - duplicating repositories with dump and load, 48
 - dynamic groups, 361
 - dm_browse_all_dynamic, 294
 - dm_superuser_dynamic, 294
 - is_dynamic_default property, 308
 - membership default, setting, 310
- E**
- electronic mail notifications
 - disabling, 105
 - electronic signatures
 - append_to_body property, 424
 - customizing, 417
 - max_signatures property, 424
 - signature creation methods,
 - creating, 425
 - signature page templates,
 - creating, 427
 - tracing, 427
 - electronic signoff
 - registering for notification, 429
 - embedded blobs, configuring, 251
 - embedded blobs, in EMC Centera storage
 - described, 214
 - EMC Centera clusters, connection strings
 - for, 249
 - EMC Centera host machines, clock requirements, 255
 - EMC Centera profiles
 - required permissions, 249
 - EMC Centera storage areas
 - a_storage_params property, 224
 - C-clip buffer size, 214
 - C-clip buffer, setting size, 252
 - ca store object type, 212
 - Centera profiles, permissions
 - required, 249
 - configuration options, 212
 - connection string, setting, 248
 - content assignment policies, 225
 - content compression, 213
 - content, removing, 268
 - deleting content from, 265
 - dfc.content.castore.write_max_attempts key, 253
 - dfc.content.castore.write_sleep_interval key, 253
 - embedded blob use, configuring, 251
 - embedding or linking content, 214
 - expired objects, removing, 506
 - Linux platform, 212
 - machine clock requirements, 255
 - Macintosh computers, 212
 - memory map interface use, 215
 - memory map interface, configuring
 - use, 254
 - retention periods, 213
 - retention policies and, 248

- retention requirements,
 - configuring, 247
- setting up, 245
- single-instancing, overriding, 215, 253
- socket connections, configuring
 - maximum, 215, 254
- write attempts, configuring, 214, 253
- EMC Centera storage hosts
 - FP_OPTION_BUFFERSIZE
 - configuration parameter, 253
- EMC Centerastorage areas
 - Centera clusters, supporting, 249
- EMC RSA Plug-in for Documentum, 315
- enable_persistence key, 178
- encrypted content files
 - dumping, 54
- encrypted passwords, 352
- encrypted storage areas, 206
- encryption
 - DM_ENCR_PASS prefix, 353
 - dm_encrypt_password utility, 354
 - keys, described, 429
 - passwords, 352
 - passwords, discontinuing for, 353
 - resolving compromised keys, 275
 - utilities, described, 432
- encryptPassword method, 353
- enforce_four_digit_year key, 104
- ENV_CONNECT_DOCBASE (iapi/idql argument), 565, 570
- ENV_CONNECT_PASSWORD (iapi/idql argument), 565, 569
- ENV_CONNECT_USER_NAME (iapi/idql argument), 564
- environment variables
 - DM_CRYPTO_FILE, 430
 - DM_DEBUG_ESIGN_METHOD, 427
 - DM_GROUP_LIST_LIMIT, 79
 - DM_HOME_CURRENT, 88, 109
- error messages
 - dm_error utility, 34
- evaluation, of ACL entries, 374
- event notifications
 - use_group_address (server.ini key), 102
- events
 - application events, auditing, 393
 - auditing, 387
 - defined, 387
 - dm_signoff notification, 429
 - full-text indexing, used by, 619
 - kinds of, 388
 - notification, disabling, 105
 - system events, 393
 - system-defined, 609
 - user-defined, 387
- exceptions, tracing in DFC, 546
- execute_proc permission, 367
- execution agents
 - choosing, 132
 - default, 132
 - described, 130
 - performance, 132
 - security issues, 133
- expiration date of jobs, 151
- expiration_date property, 151
- explicit transactions
 - caching objects and, 184
- EXPORT_TICKET_KEY administration
 - method, 436
- exportTicketKey (DFC method), 436
- extended characters, in LDIF files, 301
- extended object-level permissions
 - described, 366
 - viewing, 367
- extended restriction ACL entries, 371
- extended user privileges, 293
 - permissions to grant/revoke, 302
- extern file storage areas
 - described, 220
- extern_store_content_static (dfc.properties key), 245
- external ACLs
 - defined, 378
 - deleting, 385
 - name format, 381
- external applications
 - dedicated servers, 38
 - Docbasic argument length,
 - maximum, 137
 - jobs and, 149
 - URLs, using, 234
- external file store storage areas
 - example of use, 242
- external free storage areas
 - creating, 244
 - defined, 220
- external password checking, with
 - LDAP, 330
- external storage

- a_content_static property, 245
 - CD-ROM files, 220
 - content dump not allowed, 219
 - content load not allowed, 219
 - extern_store_content_static (dfc.properties key), 245
 - external free stores, 220
 - external URL stores, 220
 - optimizing retrieval performance, 245
 - renditions and, 220
 - replication constraint, 219
 - external storage areas
 - described, 219
 - setting up, 242
 - types of, 219
 - external URL storage areas
 - creating, 244
 - defined, 220
 - extended permissions
 - owner_xpermit_default (server.ini key), 101
 - extra_directory_config_id property, 327, 329
- F**
- fail-over mechanisms
 - multiple service names in server.ini file, 98
 - failed_auth_attempt property, 306, 356
 - failover
 - connection brokers, 164, 191
 - Content Server, 121
 - content-file server limitation, 122
 - failover_ldap_config_ids property, 331
 - failover_use_interval property, 331
 - federations
 - LDAP authentication and, 328
 - LDAP directory servers and, 325
 - file extensions
 - implementing automatic use, 258
 - in storage areas, 237
 - file formats, 67
 - See also* formats
 - format objects and, 67
 - file naming conventions in client local areas, 177
 - File Report tool, 493
 - file store keys
 - resolving compromised, 275
 - file store storage areas
 - automatic file extensions, 258
 - base_url property, 238
 - content assignment policies, 225
 - content compression, 206
 - content duplication checking, 209
 - creating, 237
 - described, 205
 - digital shredding, 211
 - encrypted, 206
 - file store keys, resolving
 - compromised, 275
 - is_public property, 206
 - location_name property, 223
 - moving, 264
 - private access, 206
 - public access, 206
 - use_extensions property, 234, 258
 - file systems
 - linked stores and, 221
 - Windows NT and, 238
 - file systems, external storage and, 220
 - file_system_path property
 - character constraint, 65
 - filename_modifier property, 70
 - files
 - dfc.data.umask (dfc.properties), 175
 - dump, moving, 59
 - removing from server common area, 88
 - temporary, 78
 - umask server.ini key, 101, 109
 - filestore_01 storage area, 233
 - filters, for LDAP, 334
 - firewalls, 194
 - folder default ACL, 382
 - folder security
 - changing, 364
 - defined, 362 to 363
 - folder_security property, 364
 - folders
 - consistency checks, 555
 - creating, 43
 - defined, 42
 - editing, 43
 - permissions to modify/delete, 44
 - force_coherency_check key, 183
 - foreign keys
 - setting, 601
 - format objects

- format_class property, 68
 - format objects
 - adding, 69
 - creating, 69
 - default_storage property, 230
 - deleting, 71
 - described, 67
 - dos_extension property, 68
 - modifying, 70
 - obtaining list of, 69
 - repository configuration, 39
 - rich media formats, 69
 - format_class property, 68
 - formats
 - formats.csv file, 68
 - internal ACL names, 379
 - obtaining list of, 69
 - rich media, 69
 - formats.csv, 68
 - fragmentation, table, 476
 - FTDQL
 - distinct_query_results server.ini key, 104
 - full-text indexing
 - ACL caching, 100
 - creating events, 470
 - event generation, stopping during load operations, 61
 - events used by, 619
 - format_class property, 68
 - job reports, 451
 - load operations and, 60
 - reindexing a repository, 470
 - starting index agents, 107
 - use_estimate_search (server.ini key), 108
 - fulltext index objects
 - repository configuration, 39
 - FUNCTION_EXTENT_SIZE server.ini file section, 95
 - FUNCTION_SPECIFIC_STORAGE server.ini section, 94
- G**
- generate_event load operations
 - parameter, 61
 - getDocbaseMap method, 200
 - getDocbrokerMap method, 201
 - gethostbyaddr (server.ini key), 100
 - getServerMap method, 200
 - GRANT (statement), 302
 - Group Rename tool, 498
 - group_admin property, 309
 - group_global_unique_id property, 327
 - group_table_permit property, 386
 - groups
 - adding, 307
 - audit trail entries, interpreting, 413
 - changing default, 304
 - consistency checks, 552
 - Create Group user privilege, 292
 - deactivated users, adding, 309
 - deleting, 309
 - docu, 61
 - dynamic, 361
 - email notifications for,
 - configuring, 102
 - group_global_unique_id property, 327
 - is_dynamic_default property, 361
 - is_private property, 308
 - LDAP synchronization, 325
 - membership setting, for dynamic groups, 308
 - modifying, 309
 - naming restrictions, 308
 - obtaining count of members, 311
 - obtaining list of, 311
 - obtaining list of members, 311
 - ownership, assigning, 307
 - ownership, changing, 309
 - private, 307
 - privileged, 294
 - properties
 - mapping LDAP attributes to, 335
 - required LDAP mappings, 336
 - properties set by synchronization, 327
 - public, 307
 - querying, 311
 - renaming, 498
 - required group entries in ACLs, 372
 - required group set ACL entries, 373
 - Saveasnew defaults, 101
 - system-created, 41
- H**
- hierarchy, repository, 42
 - history_cutoff (server.ini key), 100

- history_sessions (server.ini key), 100
 - home cabinet, 43
 - home repositories, changing, 522
 - host (server.ini key), 98
 - host machine
 - connecting to specific, 165
 - dm_crypto_boot, running after restart, 432
 - TCP/IP service name in server.ini file, 98
 - version number, 514
 - HTTP_POST administration method
 - adding servlet for, 123
- I**
- i_contents property, 212
 - i_contents_id property, 266
 - i_crypto_key property, 431
 - IAPI (utility)
 - commands for, 571
 - comments, entering, 573
 - executing methods, 572
 - exiting, 573
 - identifiers, 347
 - See also* plug-in identifiers
 - IDfAuditTrailManager interface, 396
 - IDfDumpRecord interface, 49
 - IDfLoadRecord interface, 49
 - IDmMethod interface, 142
 - IDQL (utility)
 - buffer, clearing, 567
 - commands, 566
 - idql command syntax, 563
 - queries, entering, 567
 - quitting, 568
 - starting, 563
 - ignore_client_domain key, 104
 - ignore_rule_errors (dfc.properties key), 276
 - IMPORT_TICKET_KEY administration method, 436
 - importTicketKey (DFC method), 436
 - inactivating
 - administration jobs, 453
 - jobs, 153
 - inboxes
 - archive and restore requests, 280
 - deleting dequeued items from repository, 503
 - index agent
 - monitoring status, script for, 591
 - index positions, 54
 - index_store (server.ini key), 99
 - IndexAgentCtrl script, 591
 - indexes (type), specifying location, 99
 - indexes, for dmfilesca, 272
 - information
 - connection broker, 199
 - format objects, 67
 - server startup file, 38
 - State of the Repository Report tool, 514
 - tool reports, 530
 - initialization files
 - connection broker
 - format, 192
 - invoking, 193
 - server.ini, 88
 - installations
 - described, 28
 - repository security levels, 362
 - security overview, 359
 - user privileges, 291
 - internal ACLs
 - deleting, 385
 - described, 378
 - names, 379
 - object names and, 381
 - internationalization
 - Content Server, 87
 - dump and load requirements, 48
 - group names, 308
 - plug-in authentication module code page, 350
 - short date formats, 175
 - short date formats, defining, 175
 - user names, 290, 296
 - IP addresses
 - for connection brokers, 192
 - specifying in server.ini, 98
 - translating, 194
 - is_archived property, 280
 - is_dynamic property, 361
 - is_dynamic_default property, 308, 310, 361
 - is_inactive property, 153
 - is_offline property, 280
 - is_private property, 308
 - is_public property, 206

J

- Japanese locales
 - populating on Korean host, 605
- Java method server, 131
 - described, 131
 - implementing methods for, 141
 - method object requirements, 143
 - monitoring status, script for, 590
 - servlets, adding, 123
 - shutdown.sh script, 131
 - starting/stopping, 131
 - storage location for methods, 143
- java methods
 - execution agents for, 132
 - guidelines, 137
- Java methods
 - implementing, 141
 - storage location, 143
- Java security policy files, modifying, 170
- java.ini file, 134
- java.ini file, location, 138
- JBoss, 131
- JBoss application server, 86
 - binaries, location of, 126
 - starting and stopping, 126
- job reports, full-text indexing and, 451
- job sequences
 - candidates for inclusion, 150
 - creating, 150
 - dcf_edit utility, 156
 - described, 146
 - dm_run_dependent_jobs method
 - execution, 159
 - execution, 147
 - failure recovery, 158
 - name of, 147
 - repository connection file, 148, 155
 - repository representation, 147
 - return codes, defining, 144
 - return values, 147
 - status report, described, 158
 - tool suite and, 453
- jobs, 146, 446
 - See also* job sequences; tool suite
 - a_next_invocation, 151
 - activating/inactivating, 153
 - admingroup group and, 302
 - agent exec process, 85, 148, 154, 449
 - content migration policies, 260
 - creating, 149
 - creating full-text events, 470
 - Data Dictionary Publisher, 607
 - described, 146
 - disabling all, 153
 - dm_Archive, 454
 - dm_AuditMgt, 456
 - dm_ConsistencyChecker, 459
 - dm_ContentReplication, 465
 - dm_ContentWarning, 468
 - dm_DBWarning, 476
 - dm_DMClean, 482
 - dm_DMFilescan, 487
 - dm_FileReport, 493
 - dm_FTCreateEvents, 470
 - dm_GroupRename, 498
 - dm_LDAPsynchronization, 341, 479
 - dm_LogPurge, 499
 - dm_MoveContent, 260
 - dm_QueueMgt, 503
 - dm_RenditionMgt, 509
 - dm_StateofRepository, 514
 - dm_SwapInfo, 518
 - dm_UpdateStats, 519
 - dm_UserChgHomeDb, 522
 - dm_UserRename, 523
 - dm_VersionMgt, 525
 - dumping/loading, 60
 - expiration_date property, 151
 - is_inactive property, 153
 - job reports, full-text indexing
 - and, 451
 - maximum concurrent, setting, 154
 - maximum executions, 151
 - passing standard arguments, 151
 - queueperson argument for tools, 450
 - records migration, 262
 - removing log files, 499
 - reports, viewing, 451
 - return codes for jobs in sequence, 144
 - run_interval property, 151
 - run_mode property, 151
 - run_now property, 152
 - scheduling, 150
 - server time, use in scheduling, 151
 - standard arguments passed, 449
 - start_date property, 151
 - tool suite implementation, 449
 - tool suite log files, 452
 - validating cached data, 182

K

- keep_entry_interval, 116
- keepSLabel argument, 77
- keys
 - server.ini file, 89
- keystore file
 - configuring shared file, 172
- keytool utility
 - passwords, 171
- keywords
 - PUBLISH, 608
- Kill method, 120
- Korean locales
 - populating on Japanese host, 605

L

- LDAP attribute mappings
 - benefits, 324
 - defining, 335
 - examples, 338
 - repository storage of, 337
- LDAP authentication
 - password checking program,
 - creating, 341
- ldap config objects
 - bind_type property, 334
 - deleting, 342
 - federations and, 328
 - mapping storage properties, 337
 - multiple, using, 332
 - retry_count property, 330
 - retry_interval property, 330
 - retry_interval, configuring, 343
 - set-up values, defining, 333
- LDAP directory server
 - deleting users, effect of, 306
 - dm_LDAPSynchronization tool, 479
- LDAP directory servers
 - Active Directory
 - mode requirements, 342
 - user_name mapping, 336
 - attributes, mapping to properties, 335
 - authentication, *see* user authentication
 - certificate authorities, 340
 - certutil utility, 340
 - Changepassword constraint, 324
 - choosing for authentication, 329
 - choosing for synchronization, 327
 - connection retry configuration, 330
 - deleting from repository, 342
 - dm_LDAPSynchronization job, 325
 - federations and synchronization, 325
 - implementing, 332
 - integrating entries with
 - repository, 324
 - LDAP_RECONNECT_INCREMENT_SECOND environment variable, 343
 - LDAP_RECONNECT_TIME_SECONDS environment variable, 343
 - ldapconfig.cnt file, 334
 - LDAPSynchronizationDoc.txt file, 337
 - multiple config objects, using, 332
 - multiple Content Servers, use
 - with, 342
 - multiple, using, 325
 - overview, 314
 - property mapping
 - guidelines, 337
 - required mappings, 336
 - rules for, 336
 - retry_interval, configuring, 343
 - search bases/filters, 334
 - secondary servers, 330
 - secure connections
 - configuring, 335
 - described, 330
 - set-up values, defining, 333
 - synchronization
 - activating job, 341
 - behavior on first run, 344
 - groups, 325
 - on-demand, described, 328
 - users, 325
 - usage
 - benefits, 324
 - constraints, 324
 - use with repository, 323
- LDAP synchronization, *see* LDAP directory servers
- LDAP synchronization on demand
 - described, 328
- LDAP users
 - authentication overview, 329
 - changing LDAP directory server, 344
 - on-demand synchronization, 328
 - property mappings, required, 336

- user_global_unique_id property, 327
 - user_login_domain property, 327
 - user_source property, 327
- LDAP, secure
 - certificate authorities, obtaining, 340
 - certutil utility, 340
- ldap_config_id property, 327, 329
- LDAP_RECONNECT_INCREMENT_SECONDS environment variable, 343
- LDAP_RECONNECT_TIME_SECONDS environment variable, 343
- ldapcertdb_loc location object, 335
- ldapconfig.cnt file, 334
- ldapsearch utility, 346
- LDIF files
 - described, 300
 - extended characters and, 301
- legacy files, external storage for, 220
- lifecycles
 - audit trail entries, interpreting, 404
 - consistency checks, 560
 - log files, removing, 499
- limits
 - keep entry interval, 116
 - RDBMS timeout, 106
 - table fragmentation, 476
- link_location property, 224
- linked storage areas
 - content location, identifying, 224
 - creating, 239
 - described, 221
- linking, folder security and, 363
- Linux platform, 212
- listener_queue_length (server.ini key), 118
- load balancing, 38, 121
 - connection brokers, 164, 191
- load object record objects, 49
- load operations
 - full-text indexing and, 60
 - generate_event parameter, 61
 - indexing-related events, turning off, 61
- load record objects, 49
- loading
 - accessing content files, 55
 - code pages and, 48
 - content not allowed into external storage, 219
 - DocApps and, 64
 - job objects, 60
 - new repositories, 61 to 62
 - preload utility, 62
 - registered tables, 61
 - repositories, 48, 64
 - troubleshooting, 64
 - Web Publisher repositories, 48
- local_clean_on_init (dfc.properties key), 177
- local_dir, changing, 176
- local_diskfull_limit (dfc.properties key), 176
- local_purge_on_diskfull (dfc.properties key), 176
- locales
 - adding, 593
 - dd_locales, setting, 44
- locales, supported, 87
- location objects
 - creating, 64 to 65
 - repository configuration, 38
 - security_type property, 65
 - State of the Repository Report tool, 514
- location_name property, 223
- log files
 - connection broker, 79
 - content assignment policy use, for, 275
 - Content Server, 79
 - DFC (Documentum Foundation Classes), 536
 - removing for repository cleanup, 78
 - server, 121
 - session, 78
- Log Purge tool, 499
- logger, for DFC tracing, 537
 - logging appender, configuring, 537
- logging appender, 537
- logging appender, configuring, 537
- login failure, auditing, 391
- login ticket key
 - exporting and importing, 436
 - resetting, 436
- login tickets, 107
 - backwards compatibility, 173
 - restricting use of global, 438
 - revoking in repository, 438
 - server cache size, setting, 173
 - troubleshooting, 438

- validity period, defining, 437
 - validity period, setting, 173
- login_ticket_timeout property, 173, 437
- logs
 - lifecycle, 499
 - removing files, 499
 - tool job files, 452
- M**
- Macintosh
 - external storage and, 219
 - Windows NT servers and, 238
- Macintosh computers, 212
- Macintosh files
 - storage constraint, 209
- mail_notification (server.ini key), 105
- map_attr property, 337
- map_attr_type property, 338
- map_rejection property, 338
- map_val property, 338
- map_val_type property, 338
- mapping rules for LDAP property
 - mappings, 336
- max_auth_attempt property, 306, 356
- max_connect_retries (dfc.properties), 167
- max_count (dfc.properties key), 167
- max_ftacl_cache_size (server.ini key), 100
- max_iterations property, 151
- max_login_ticket_timeout property, 437
- max_nqa_string (server.ini key), 101
- max_sessions_heap_size (server.ini key), 101
- max_signatures property, 424
- max_stack_depth, 542
- max_storage_info_count (server.ini key), 105
- Maximum Content Migration Threads
 - value, 260
- maximums
 - age of renditions, 510
 - age of versions, 527
 - blob storage size, 217
 - concurrent jobs, 154
 - concurrent sessions, 103
 - connection broker projection
 - targets, 116
 - dump cache size, 55
 - failed authentication attempts, 356
 - history sessions, 100
 - job executions, 151
 - predicate property, 53
- Media Server
 - new repositories and, 40
- Media Transformation Services
 - new repositories and, 40
- memory
 - dump cache, 56
 - login tickets, 107
 - max_sessions_heap_size (server.ini key), 101
- memory map interface, using, 254
- messages
 - dump and load, 64
 - from tools, 452
 - session log file, 532
- method execution queue, 130
- method objects
 - consistency checks, 561
 - creating, 143
 - jobs passing standard arguments, 151
- method server
 - java.ini file, 134
 - method object requirements, 140
 - method_server_enabled (server.ini key), 105
 - method_server_threads (server.ini key), 105
- method_server_enabled (server.ini key), 105
- method_server_threads (server.ini key), 105
- methods
 - defined, 129
 - executing automatically, 146
 - executing on demand, 145
 - execution agents
 - choosing, 132
 - described, 130
 - performance, 132
 - execution queue, 130
 - IAPI execution, 572
 - IDmMethod interface, 142
 - implementing, 136
 - output, recording, 140
 - property settings for Content Server, 140
 - property settings for dmbasic method server, 140

- property settings for Java method
 - server execution, 143
 - removing results files, 499
 - security issues, 133
 - storing content, 129
 - success_return_codes, setting, 144
 - success_status, setting, 144
 - tracing, 133
 - Microsoft Performance Monitor tool, 33
 - MIGRATE_CONTENT administration
 - method, 259
 - auditing, 263
 - memory map interface use, 215
 - minimum values
 - age of audit trails, 458
 - age of dequeued items, 504
 - modifying
 - ACLs, 384
 - format objects, 70
 - groups, 309
 - server config objects, 109
 - types, 75
 - users, 303
 - monitoring scripts, 589
 - Mount method, 243
 - mount point objects
 - alias drive letters and, 66
 - repository configuration, 39
 - mount points
 - creating, 66
 - moving
 - content through archive, 280
 - contents through restoring, 281
 - dump files, 59, 282
 - objects, folder security and, 363
 - repositories, 48
 - MS SQL Server
 - database connection strings, 97
 - database_name server.ini key, 97
- N**
- naming conventions
 - ACLs, 379
 - archive dump files, 281
 - blob storage areas, 236
 - data ticket values, 234
 - file extensions, 237
 - files in client local areas, 177
 - log purge reports, 499
 - storage area subdirectories, 272
 - storage objects, 63
 - user objects, 290
 - win_preferred_alias, 67
 - native and secure connection mode
 - setting, 111
 - native connection mode setting, 111
 - NetApp SnapLock storage areas
 - a_storage_params property, 224
 - content assignment policies, 225
 - content compression, 217
 - retention periods, 216
 - retention policies and, 257
 - retention requirements,
 - configuring, 256
 - setting up, 255
 - network service names, 112
 - no-domain required mode, 318
 - NOFTDQL hint, 104
 - None user privilege, 292
 - notifications
 - disabling, 105
 - number (quantity of)
 - concurrent sessions, 103
 - connection brokers, 188
 - historical sessions, 100
 - login tickets, 107
 - numbers
 - data ticket, 234
 - docbase_id, 96
 - port, for connection brokers, 187
 - proximity values, 117
- O**
- object replication
 - external store constraint, 219
 - object type indexes
 - consistency checks, 561
 - object type tables
 - alternate locations for (Oracle or DB2), 45
 - object types
 - ACLs, assigning, 382
 - consistency checks, 558
 - default_storage property, 230
 - dm_create_cabinet group, 295
 - dm_create_group group, 295
 - dm_create_type group, 295
 - dropping from repository, 76

- modifying, 73
 - privileges to modify/create, 72
 - object-level permissions
 - basic permissions, 365
 - described, 364
 - extended permissions, 366
 - table permits and, 387
 - objects
 - application control of, 364
 - application-level control of, 361
 - creating and folder security, 363
 - deferred object update, queue size, 107
 - deleting and folder security, 363
 - dump object record, 49
 - dump record, 49
 - format, 39
 - fulltext index, 39
 - load object record, 49
 - load record, 49
 - location, 38
 - mount point, 39
 - moving and folder security, 363
 - owner access evaluation, 375
 - storage, 39
 - superuser access to, 376
 - oldest_client_version property, 168
 - operator_name property
 - described for archive tool, 284
 - tool messages and, 450
 - Oracle
 - database connection strings, 96
 - database_name server.ini key, 97
 - dm_dbo alias, 73
 - object type table location, 45
 - object type tables
 - size, defining, 95
 - tablespace, defining, 94
 - Oracle Intranet Directory
 - properties, indexing, 334
 - orphan content files, 76
 - orphaned annotations, 76
 - owner_table_permit property, 386
 - owner_xpermit_default (server.ini key), 101
- P**
- package_control property, 80
 - packages
 - component names, controlling use of, 80
 - parallel content migration, 259
 - parent server
 - described, 84
 - parent_id property, 223
 - pass_standard_argument property, 151
 - passphrases, changing, 434
 - password checking program
 - creating, 319
 - requirements, 320
 - password checking program, creating for LDAP, 341
 - passwords
 - change constraint with LDAP, 324
 - changing encrypted, 354
 - connection broker shutdown, 193
 - encrypted, 352
 - server authentication, 108
 - stopping encrypted use, 353
 - performance
 - auditing and, 456
 - blob storage areas, 217
 - dmclean and dmfilescan, 266
 - dump operations, 55
 - inboxes and, 503
 - large archive files, 283
 - load balancing, 38
 - turbo storage, 218
 - type indexes, 44
 - performance of queries, 79
 - permissions
 - base object-level, 364
 - extended object level, 366
 - multiple entries for single user, resolving, 376
 - required for Centera profile, 249
 - permits, registered tables, 362
 - persistent client caches
 - cache config objects, creating, 179
 - cached objects backup, 183
 - client_pcaching_change property, 182
 - client_pcaching_disabled property, 178
 - described, 177
 - forcing currency checks, 183
 - troubleshooting, 184
 - persistent client caching
 - client check interval, setting, 181
 - enabling/disabling, 178

- flushing a cache, 181
- server check interval, setting, 180
- persistent object cache
 - automatic back up, 183
- PKI credentials
 - accessing, 171
 - location, 171
- plug-in authentication module
 - implementing custom, 349
- plug-in authentication modules
 - CA SiteMinder, 348
 - code page, defining, 350
 - described, 346
 - dmathplug.h file, 350
 - dmathplug.lib file, 350
 - identifiers, 347
 - identifying to server, 347
 - RSA, 348
 - trace files, 351
- plug-in identifiers
 - CA SiteMinder, 349
 - defining, 347
- policy engine (DFC)
 - disabling, 276
- policy engine, content assignment
 - policies, 226
- port number for connection brokers, 187
- post_archive (DO_METHOD procedure), 283
- post_restore (DO_METHOD procedure), 283
- pre_restore (DO_METHOD procedure), 283
- precedence, LDAP directory server, 324
- predicate property, 53
- predicate2 property, 53
- preload utility, 62
- preserve_existing_types (server.ini key), 106
- primary content
 - storage algorithm, 229
- private
 - ACLs
 - creating, 380
 - ACLs, defined, 379
 - cabinets, 43
 - file store storage areas, 205
 - groups, 307
 - private file stores, 206
 - privileged delete
 - required EMC Centera profile
 - permission, 249
- privileged DFC
 - alias, defining, 172
 - approved_clients_only property, 169
 - client rights objects, 168
 - dfc.keystore file, 171
 - disabling, 170
 - enabling, 169
 - Java security policy files and, 169 to 170
 - role groups used by, 295
- privileged groups, 294
 - role groups used by DFC, 295
- privileges, *see* user privileges
- programs
 - shutting down servers with, 119
- projection properties for connection broker, 115
- projection targets, 114
- properties
 - adding to types, 74
 - auditing
 - initiating, 388
 - changing on signature page
 - template, 420
 - default access permissions, 73
 - deleting, 74
 - deleting from types, 75
 - dump record, 49
 - index position, 54
 - lengthening string, 75
 - load object, 49
 - mapping to LDAP attributes, 324
 - max_nqa_string (server.ini key), 101
 - object and ACL connection, 380
 - permissions and, 365
 - values in State of the Repository Report tool, 514
- property mappings for LDAP,
 - examples, 338
- proximity values
 - defining, 117
 - projection_proxval property, 115
- Prune method
 - automating with version management
 - tool, 525
 - folder security and, 363
 - repository cleanup, 77
- public

- cabinets, 43
- file store areas, 205
- groups, 307
- public ACLs
 - defined, 379
- PUBLISH keyword, 608
- publishDataDictionary method, 607
- publishing
 - data dictionary, 607
- Purge Audit privileges, 416
- Purge Audit user privileges, 293
- PURGE_AUDIT administration
 - method, 416
 - auditing, 391
- purgeLocalFiles method, 177

Q

- queries
 - distinct_query_results (server.ini key), 104
 - duplicate rows in results, 104
- query performance, 79
- queue items, deleting unwanted, 78
- Queue Management tool, 503
- queueperson
 - tool argument, 450
 - tool errors, 530

R

- r_component property, 224
- r_content_hash property, 210
- r_gen source property, 394
- r_is_public property, acl_update_threshold and, 98
- r_normal_tz property, 45
- r_status property, 264
- RDBMS
 - connection timeout, 106
 - Database Space Warning tool, 476
 - finding fragmented tables, 476
 - locks, releasing, 99
 - login name, 290
 - setting table permits, 386
 - table permits, 386
 - updating table statistics, 519
 - version number, 514
- rdbms_connect_retry_timeout key, 106
- records migration jobs, creating, 262

- RECOVER_AUTO_TASKS administration
 - method, 125
- registered tables
 - affects of loading operations, 61
 - setting table permits, 386
 - table permits, 386
 - user_db_name and, 290
- reindexing repositories, 470
- rejecting server broadcasts, 193
- Relate permit level and relationships, 366
- relation ssa policy objects, 227
- relationships
 - permissions for, 366
- Remove Expired Retention Objects
 - tool, 506
- removeDynamicGroup method, 311
- removing
 - aborted workflows, 77
 - dequeued items, 503
 - log files, 499
 - renditions, 509
- Rendition Management tool, 509
- renditions
 - external storage and, 220
 - removing old, 77, 509
 - storage algorithm, 231
 - turbo storage and, 218
- replica_filestore_01 storage area, 233
- replicate_temp_store storage area, 233
- reports (tool suite), 451
- repositories
 - activating LDAP synchronization job, 341
 - adding
 - format objects, 69
 - groups, 307
 - users, 295
 - AEK described, 430
 - AEK passphrase, changing, 434
 - AEK, backing up, 431
 - application access control tokens,
 - configuring, 439
 - approved_clients_only property, 169
 - audit trails
 - interpreting, 397
 - removing, 416
 - auditing, 387
 - application events, 393
 - auditing, default, 391
 - changing user home, 522

- cleaning up, 76
- client_pcaching_disabled
 - property, 178
- connecting to, 165
- content storage options, 204
- creating, 39
- creating cabinets/folders, 43
- database_owner in server.ini, 96
- dd_locales, setting, 44
- default_app_permit, 364
- defined, 37
- deleting
 - ACLs, 385
 - dequeued items, 503
 - format objects, 71
 - groups, 309
 - orphaned objects, 76
 - users, 304
- dmbasic method server,
 - described, 130
- docbase config object, 38
- dumping and loading, 48, 64
- dumping complete, 51
- dumping partial, 51
- enabling dmbasic method
 - server, 135
- encryption key, 431
- getDocbaseMap method, 200
- groups, system-created, 41
- Java method server, described, 131
- LDAP directory server, adding, 332
- LDAP directory servers and, 323, 325
- limiting access using client
 - version, 168
- listing format objects, 69
- loading, 59
- loading from dump file, 62
- loading new, 61
- locales, adding, 593
- login ticket keys, managing, 436
- login tickets, configuring, 437
- macl_security_disabled property, 377
- method execution agents, 130
- methods, implementing, 136
- modifying
 - format objects, 70
 - groups, 309
 - users, 303
- moving, 48
- multiple Content Servers, 85
- name in server.ini, 96
- object defining configuration, 38
- object types
 - modifying, 73
- objects in new repository, 41
- obtaining list of groups in, 311
- organization, 42
- properties supporting LDAP
 - mappings, 337
- query performance, 79
- repository identifier in server.ini, 96
- restoring documents, 495
- sample dump script, 57
- security
 - folder security setting, 364
- starting connection brokers, 196
- State of the Repository tool, 514
- storage areas, configuration
 - options, 222
- synchronizing with directory
 - servers, 325
- trust mode, configuring, 437
- uninstalling, 33
- uninstalling DB2, affects of, 100
- user privileges
 - basic, 291
 - extended, 293
- users
 - renaming, 304
 - users, adding multiple, 299
 - users, system-created, 41
 - Web Publisher, 48
 - workflow package control,
 - configuring, 80
- repository configuration
 - audit_old_values property, 388
 - login_ticket_cutoff property, 438
- repository connection file
 - creating and maintaining, 155
- repository connection file, for job
 - sequences, 148
- repository encryption key, 431
- repository name, changing, 523
- repository operator
 - archive tool and, 284
 - tool messages and, 450
- repository sessions
 - changing configuration, 174
 - connection brokers, specifying, 163
 - connection pooling, configuring, 172

- dfc.session.pool.enable (dfc.properties), 172
 - max_count (dfc.properties key), 167
 - session config objects, 174
 - SSL default, setting, 166
 - UNIX constraint on number of, 168
 - require_ticket property, 234
 - required group ACL entries
 - aliases, constraint on use, 380
 - described, 372
 - required group set ACL entries
 - aliases, constraint on use, 380
 - described, 373
 - RESET_TICKET_KEY administration
 - method, 436
 - resetTicketKey (DFC method), 436
 - respository connection file
 - commas and backslashes, escaping, 156
 - dcf_edit utility, 156
 - respository sessions
 - requests for, configuring, 167
 - restoring content files through archive and restore, 277
 - restrict_su_ticket_login property, 438
 - restricted_folder_ids property, 298
 - results log files, deleting, 499
 - retainer objects
 - dump operations and, 50
 - retention storage areas
 - EMC Centera, 212
 - retention periods
 - defined in EMC Centera storage areas, 213
 - defined in NetApp SnapLock storage areas, 216
 - retention policies, 231
 - dm_retention_managers group, 294
 - dm_retention_users group, 294
 - dump operations and, 50
 - EMC Centera store retention periods and, 248
 - NetApp SnapLock store retention periods and, 257
 - retention requirements
 - configuring for EMC Centera stores, 247
 - configuring for NetApp SnapLock stores, 256
 - retry_count property, 330
 - retry_interval property, 330
 - return codes, for jobs in sequence, 144
 - REVOKE (statement), 302
 - rich media formats
 - format objects, 69
 - richmedia_enabled property, 70
 - roles, for privileged DFC, 295
 - root certificate authority
 - obtaining, 340
 - RPC calls, tracing, 546
 - RSA Access Manager, *see* EMC RSA Plug-in for Documentum
 - RSA plug-in modules, 348
 - RSA plugin, *see* EMC RSA Plug-in for Documentum
 - run_interval property, 151
 - run_mode property, 151
 - run_now property, 152
- ## S
- Saveasnew method, 363
 - default group assignment and, 101
 - saveasnew_retain_source_group
 - server.ini key, 101
 - schedules, changing tool, 453
 - scripts
 - archive, 282
 - ContentServerStatus, 590
 - dmclean, 266
 - dmfilesca, 270
 - dmqdocbroker, 590
 - IndexAgentCtrl, 591
 - preload utility, 62
 - sample repository dump, 57
 - TestConnection, 590
 - search bases, for LDAP, 334
 - search_order (dfc.properties key), 191
 - secondary LDAP servers, 330
 - secure connection default, setting, 166
 - secure connection mode
 - resetting, 110
 - secure connection mode setting, 111
 - secure connections
 - LDAP and, 330
 - secure LDAP
 - certicate database location, 335
 - setting up, 335
 - secure_connect_default (dfc.properties), 166

- secure_connect_property, 111
- security
 - ACL caching, 100
 - ACL naming conventions, 379
 - acl objects, 369
 - ACLs, creating, 380
 - application access control tokens,
 - enabling use, 439
 - application control of SysObjects, 364
 - auditing, 387
 - changing
 - ACL object name, 379
 - ACL owner name, 379
 - connection broker shutdown, 193
 - connection brokers and, 198
 - Create Cabinet (user privilege), 292
 - Create Type (user privilege), 292
 - default ACLs, 381
 - deleting ACLs, 385
 - digital signatures, 427
 - electronic signatures,
 - customizing, 417
 - execution agents, for methods, 133
 - folder, 362 to 363
 - granting/revoking user
 - privileges, 301
 - modifying ACLs, 384
 - overview, 359
 - repository security levels, 362
 - secure connections with LDAP, 330
 - security argument, 88
 - signing audit trail entries, 394
 - Superuser (user privilege), 292
 - Sysadmin (user privilege), 292
 - table permits, 386
 - user privileges, 291
- security_mode property, 362
- security_type property, 65
- SELECT (statement)
 - finding users with, 308
- server common area
 - removing files from, 88
- server config objects
 - changing, 109
 - connection broker projection
 - properties, 115
 - described, 85
 - modifying, 110
 - operator_name property and archive
 - tool, 284
 - setting checkpoint_interval
 - property, 116
 - setting keep_entry_interval
 - property, 116
- server configuration
 - application access control tokens,
 - enabling, 439
 - global login tickets, restricting
 - superuser use, 438
 - login_ticket_timeout property, 437
 - max_login_ticket_timeout
 - property, 437
- server log files
 - deleting, 499
 - described, 121
 - trace messages, 532
- server. ini file
 - start_index_agents key, 107
- server_check_interval property, 180
- server_config_name key, 106
- server.ini
 - method_server_enabled, 105
- server.ini file
 - acl_update_threshold, 98
 - changing, 110
 - comment character, 89
 - commit_read_operations key, 99
 - connection broker entries, 93
 - data_store key, 99
 - database_conn key, 96
 - database_name key, 97
 - database_owner key, 96
 - database_refresh_interval, 104
 - defaulted keys, 103
 - deferred_update_queue_size, 107
 - defining projection targets, 116
 - described, 84, 88
 - docbase_id key, 96
 - docbase_name key, 96
 - enforce_four_digit_year key, 104
 - gethostbyaddr key, 100
 - history_cutoff key, 100
 - history_sessions key, 100
 - ignore_client_domain, 104
 - index_store key, 99
 - mail_notification key, 105
 - max_sessions_heap_size, 101
 - max_storage_info_count key, 105
 - preserve_existing_types, 106

- saveasnew_retain_source_group
 - key, 101
- server_config_name key, 106
- service key, 98
- State of the Repository Report
 - tool, 514
- upd_last_chg_time_from_db key, 102
- use_estimate_search, 108
- use_group_address key, 102
- validate_database_user key, 102
- server_login_ticket_version (server.ini key), 174
- SERVER_STARTUP server.ini file
 - section, 89
- server_startup_sleep_time key, 106
- service (server.ini key), 98
- service name
 - connection broker, 197
 - server, 112
- services file, 196
- servlets
 - adding to Java method server, 123
- session config objects
 - described, 174
- session log files
 - deleting, 499
 - described, 532
 - dump and load messages, 64
- session servers, 84
- session threads, 84
- sessions
 - client_session_timeout (server.ini key), 103
 - dfc.properties file, 162
 - max_sessions_heap_size (server.ini key), 101
 - modifying configuration, 174
 - session config object, 174
 - trusted logins, disabling, 166
 - viewing historical, 100
- set_file property, 280
- SET_OPTIONS
 - trace_method_server flag, 133
- Setup program, 33
- shared libraries
 - plug-ins, 220
- shared memory, login tickets, 107
- Shift_JIS
 - NEC extensions, 88
- short date formats, 175
 - described, 175
 - UNIX defaults, 175
 - Windows default, 175
- shutdown method, 120
- shutting down
 - connection broker, requiring
 - passwords for, 193
 - connection brokers, 197
 - servers, 118
 - sessions, 120
- signature creation method
 - custom, creating, 425
 - parameters passed to, 425
 - tracing, 427
- signature page template
 - adding/removing properties, 420
 - appearance, modifying, 422
 - append_to_body property, 424
 - changing delimiters, 421
 - default, described, 418
 - document types, defining for, 423
 - number of signatures,
 - configuring, 424
- signature page templates
 - creating, 427
- signature requirement support
 - simple signoffs, customizing, 428
- signature validation programs
 - customizing, 428
- signature_chk_loc property, 429
- signatures
 - audit trail entries, on, 394
 - digital, 427
 - verifying in distributed
 - environment, 415
- Signoff method
 - auditing, 391
- signoffs, simple
 - customizing, 428
 - signature_chk_loc property, 429
 - validation program, customizing, 428
- single-repository configurations
 - signatures, verifying, 415
- single-repository distributed configurations
 - AEK requirements, 431
- size
 - blob storage areas, 217
- sleep interval (workflow agent)
 - changing, 124

- sorting query results, 104
- space, reclaiming
 - in repositories, 77
- ssa policy objects, 227
- ssl_mode property, 335
- ssl_port property, 335
- standard arguments for jobs, 151, 449
- start_date property, 151
- start_index_agents (server.ini key), 107
- starting
 - additional Content Servers, 112
 - connection brokers, 195
 - Content Servers, 88, 111
- startup.sh script, 131
- State of the Repository tool, 514
- statistics
 - updating, 519
- statistics, updating, 79
- stopping
 - connection brokers, 197
 - jobs, 153
 - servers, 118
- storage algorithm
 - primary content, 229
 - renditions, 231
- storage algorithms
 - DFC policy engine, 228
- storage areas
 - blob set up, 236
 - cleaning up, 266
 - configuration options, summary, 222
 - default, 232
 - determining state, 264
 - distributed store components,
 - replacing, 274
 - EMC Centera, 212, 245
 - file stores, creating, 237
 - file stores, moving, 264
 - filestore_01, 233
 - full-text indexing and, 223
 - moving files, 259
 - naming conventions in file stores, 233
 - naming conventions,
 - subdirectories, 272
 - NetApp SnapLock, 255
 - removing orphan files, 487
 - replica_filestore_01, 233
 - replicate_temp_store, 233
 - setting offline/online, 263
 - shared memory, 105
 - streaming_storage_01, 233
 - thumbnail_storage_01, 233
 - types, 74
 - types of, 204
 - use_extensions property, 237
- storage types
 - blob store, 217
 - distributed store, 218
 - file store, 205
 - linked store, 221
 - turbo storage, 218
- storage_id property, 223
- storage objects
 - repository configuration, 39
- streaming content files
 - storage area default, 233
 - URLs, format, 234
- streaming_storage_01 storage area, 233
- string properties, lengthening, 75
- subcontent objects, 218
- success_return_codes property, 144
- success_status property, 144
- Superuser privilege
 - access permissions for, default, 293
- Superuser privileges
 - dm_superuser group, 294
 - dm_superuser_dynamic group, 294
- Superuser user privilege
 - admingroup group and, 302
 - dumping repositories, 48
 - granting and revoking, 302
 - loading repositories, 48
- Superuser user privileges, 292
- superusers
 - ACL entry evaluation and, 376
 - global login tickets, restricting use of, 438
- Swap Info tool, 518
- Sybase
 - database connection strings, 97
 - database_name server.ini key, 97
 - dm_dbo alias, 73
- Sysadmin privileges
 - dm_sysadmin group, 295
- Sysadmin user privilege, 292
- SysObjects
 - a_storage_type property, 230
 - access permission levels, 364
 - application-level control, 364
 - consistency checks, 554

- determining associated ACL, 381
 - i_contents_id property, 266
 - owner access evaluation, 375
 - superuser access to, 376
 - system ACLs
 - creating, 380
 - defined, 379
 - system administration
 - tool suite, 33
 - System cabinet
 - load operation and, 61
 - location objects in, 64
 - mount point objects in, 64
 - tool reports in, 530
 - system events, 387 to 388, 393
 - audit trails, viewing, 396
 - system-defined events, 609
- T**
- table permits
 - affects of loading operation, 61
 - described, 362, 386
 - dump and load, effect of, 387
 - object-level permissions and, 387
 - setting, 386
 - tables
 - finding fragmented, 476
 - updating statistics, 519
 - tablespaces
 - identifying, 99
 - targets, connection broker projection, 114
 - TCP/IP service name in server.ini file, 98
 - Temp cabinet
 - load operations and, 61
 - tool trace logs, 530
 - template ACLs, 379
 - creating, 380
 - TestConnection script, 590
 - threads, session and Content Server, 84
 - threads, tracing, 545
 - thumbnail files
 - URLs, format, 234
 - Thumbnail Server
 - new repositories and, 40
 - thumbnail_storage_01 storage area, 233
 - ticket_multiplier server.ini key, 107, 173
 - tickets, number in shared memory, 107
 - time zones, date storage and, 45
 - timeouts, 437
 - See also* validity periods
 - connection to RDBMS, 106
 - tool suite
 - activating/inactivating a tool, 453
 - agent exec process and, 449
 - Archive, 454
 - Audit Management, 456
 - Consistency Checker, 459
 - Content Replication, 465
 - Content Warning, 468
 - create full-text events, 470
 - Data Dictionary Publisher, 475
 - Database Space Warning, 476
 - dm_LDAPSynchronization, 479
 - Dmclean, 482
 - Dmfilescan tool, 487
 - email messages, 452
 - File Report, 493
 - Group Rename, 498
 - implementation, 449
 - inactivating a tool, 453
 - job log files, 452
 - job reports, viewing, 451
 - job sequences, using, 453
 - Log Purge, 499
 - overview, 33, 446
 - Queue Management, 503
 - queueperson argument, 450
 - Remove Expired Retention Objects, 506
 - Rendition Management, 509
 - reports, 451
 - running on demand, 454
 - schedule recovery, 450
 - scheduling window, 450
 - State of the Repository, 514
 - Swap Info, 518
 - tool schedules, setting, 453
 - ToolSetup (install utility), 519
 - Update Statistics tool, 519
 - UserChgHomeDb, 522
 - UserRename, 523
 - Version Management, 525
 - window_interval argument, 450
 - ToolSetup tool, 519
 - trace files, 537
 - See also* DFC tracing
 - LDAP synchronization jobs, 345
 - plug-in authentication modules, 351
 - trace_method_server flag, 133

- tracing
 - content duplication checking and prevention, 211
 - Content Server options, 532
 - DFC operations, 536
 - DFC trace file format, 548
 - job execution, 155
 - LDAP authentication failover, 332
 - method execution, 133
 - rejected LDAP entries, 337
 - workflow agent, 124
 - transactions
 - Kill method and, 120
 - loading and, 60
 - server shutdown and, 120
 - session server and, 120
 - translating IP addresses for connection brokers, 194
 - troubleshooting
 - Content Server Manager, 33
 - dump and load, 64
 - persistent client caches, 184
 - process monitoring scripts, 589
 - repository dump operation, 59
 - trust_by_default property, 437
 - Trusted Content Services
 - ACL evaluation and, 374
 - ACLs, creating, 380
 - ACLs, modifying, 384
 - trusted login, disabling, 166
 - trusted logins, 351
 - trusted logons, 318
 - trusted_docbases property, 437
 - turbo storage
 - described, 218
 - renditions of content, 218
 - setting up, 257
 - size constraints, 218
 - turbo storage areas
 - content location, identifying, 224
 - type default ACL, 382
 - type indexes
 - creating, 44
 - type info object type
 - default_storage property, 230
 - TYPE_EXTENT_SIZE server.ini file
 - section, 95
 - TYPE_SPECIFIC_STORAGE server.ini
 - section, 94
 - types, 72
 - See also* user-defined types
 - adding properties, 74
 - changing, 74
 - Create Type user privilege, 292
 - creating, 73
 - default ACL, 382
 - default ACL, changing, 74
 - dropping properties, 75
 - lengthening string properties, 75
 - privileges to modify/create, 72
 - removing from repository, 76
 - specifying index location, 99
- ## U
- umask, *see* dfc.data.umask (dfc.properties)
 - umask server.ini key, 101, 109
 - Unaudit method
 - auditing execution of, 391
 - unified login, enabling, 351
 - uninstalling a repository, 33
 - UNIX
 - constraint on number of sessions, 168
 - UNIX clients
 - short date format, 175
 - unlinking, folder security and, 363
 - upd_last_chg_time_from_db (server.ini key), 102
 - Update Statistics administration tool, 79
 - Update Statistics tool, 79, 519
 - update_access_date server.ini key, 107
 - updating table statistics, 519
 - URLs
 - base_url property, 238
 - format, 234
 - use_estimate_search (server.ini key), 108
 - use_extensions property
 - appending extension, 234
 - file extensions and, 237
 - setting up storage area, 258
 - use_group_address (server.ini key), 102
 - Useacl method, 383
 - user authentication
 - auth_protocol property, 318
 - authentication name, 296
 - authentication plug-ins, 314
 - CA SiteMinder, 315
 - certutil utility, 340
 - dm_check_password file, 316
 - domain-required mode, 318

- domains, affect of, 317
- failed attempts, limiting, 356
- failover, 331
- failover for LDAP, 330
- failover tracing, 332
- LDAP and federations, 328
- LDAP directory servers, 314
- LDAP options, 330
- LDAP synchronization job, 341
- no-domain required mode, 318
- options, 313
- password checking program,
 - creating, 319
- plug-in modules
 - described, 346
- secure LDAP
 - setting up, 335
- trusted logins, 351
- unified login, 351
- UNIX default mechanism, 316
- UNIX users against domain, 314
- UNIX users in domains, 320
- user_login_namee property, 290
- user_os_domain property, 318
- user_os_name property, 290
- Windows default mechanism, 317
- user default ACL, 382
- user names
 - code page requirements, 290, 296
 - overview, 290
 - user_db_name property, 290
 - user_login_name property, 290
 - user_name property, 290
 - user_os_namee property, 290
- user privileges
 - audit records, viewing and, 396
 - basic, list of, 291
 - Create Cabinet, 292
 - Create Group, 292
 - Create Type, 292
 - described, 291, 362
 - dm_create_cabinet group, 295
 - dm_create_group group, 295
 - dm_create_type group, 295
 - dm_create_user group, 295
 - extended, list of, 293
 - GRANT statement, 302
 - None, 292
 - permissions to grant/revoke, 301
 - REVOKE statement, 302
 - setting, 297, 302
 - User Renametool, 304
 - user type default ACL, 382
 - user_address property, 296
 - user_auth_case key, 108
 - user_auth_target key, 108
 - user_db_name property, 290, 298
 - user-defined properties
 - adding, 74
 - dropping, 75
 - user-defined types
 - adding properties, 74
 - allowed supertypes, 72
 - changing, 74
 - changing default acl, 74
 - creating, 72
 - lengthening string properties, 75
 - removing from repository, 76
 - user_global_unique_id property, 327
 - user_login_doamin property, 327
 - user_login_name property, 290, 296, 329
 - user_name property, 290, 296
 - user_os_domain property, 318
 - user_os_name property, 290
 - user_password property, 351
 - user_privileges property, 302
 - user_source property, 323, 327
 - user_state property, 305 to 306
 - user_validation_location property, 316
 - user_xprivileges property, 302
 - UserChgHomeDb tool, 522
 - UserRename tool, 523
 - users, 327, 336
 - See also* LDAP users
 - ACL entry evaluation and, 374
 - activating, 306
 - adding, 295
 - adding multiple, 299
 - authenticating, 108
 - client capability, 299
 - configuring for Windows NT domain
 - authentication, 323
 - consistency checks, 552
 - deactivated, adding to groups, 309
 - deactivating or locking, 305
 - default ACL, defining, 297
 - default folder, defining, 298
 - deleting, 304
 - dm_create_user group, 295
 - email address, 296

- extended privileges, 293
 - granting/revoking privileges, 301
 - home repository, changing, 522
 - in-line password authentication, 351
 - inactivating automatically, 306
 - LDAP authentication, 328, 330
 - LDAP authentication failover, 330
 - LDAP synchronization, 325
 - login failure, auditing, 391
 - modifying, 303
 - name properties, 290
 - privileges, 291, 362
 - properties
 - mapping LDAP attributes to, 335
 - properties set by synchronization, 327
 - renaming, 304, 523
 - restricting folder access, 298
 - system-created, 41
 - tracing in DFC, 544
 - user_db_name, setting, 298
 - viewing connected, 34
- utilities
- certutil, for ldap, 340
 - dm_crypto_boot, 432
 - dm_crypto_change_passphrase, 434
 - dm_crypto_create, 433
 - dm_encrypt_password, 354
 - dm_error, 34
 - dm_launch_docbroker, 195
 - dm_stop_docbroker, 198
 - dmclean, 266, 269
 - dmdocbroker.exe, 195
 - dmfilescan, 266
 - dmtkgen, 441
 - preload, 62
- V**
- validate_database_user server.ini key, 102
 - validate_user location object, 316
 - .validation.interval (dfc.properties key), 277
 - validity periods, for login tickets
 - defining, 437
 - verbosity, in DFC tracing, 546
 - verifyAudit method, 415
 - verifyESignature method
 - tracing, 427
 - Version Management tool, 525
 - version numbers in State of the Repository
 - Report tool, 514
 - versions
 - removing outdated, 77
 - Version Management tool, 525
 - View Audit user privileges, 293
- W**
- wait_for_connect_timeout server.ini key, 108
 - Web Publisher repositories
 - dump and load operations not supported, 48
 - web.xml file, trace parameter, 134
 - wf_agent_worker_threads property, 124
 - wf_package_control_enabled property, 80
 - wf_sleep_interval property, 124
 - window_interval argument, 450
 - Windows clients
 - short date format, 175
 - Windows domain
 - authenticating in, 317
 - Windows NT
 - file systems, 238
 - Macintoshes and, 238
 - work items
 - email notification, disabling, 105
 - recovering claimed work items, 125
 - worker sessions (workflow agent)
 - changing number of, 124
 - workflow agent
 - configuring, 123
 - disabling, 124
 - sleep interval, changing, 124
 - tracing, 124
 - worker sessions, changing
 - number, 124
 - workflow definitions
 - package_control property, 80
 - workflow methods, return values, 137, 141
 - workflows
 - audit trail entries, interpreting, 405
 - consistency checks, 557
 - dmclean and, 267
 - package name control, configuring, 80
 - removing aborted workflows, 77

use_group_address (server.ini
key), 102
workflow agent, 123
world_table_permit property, 386
write_interval key, 183

write_max_attempts (dfc.properties
key), 253
write_sleep_interval (dfc.properties
key), 253