# Content Server API Reference Manual


**Version 5.3 SP1**
**September 2005**

# Table of Contents

# List of Figures

# List of Tables

# Preface

This manual is the reference manual for the DMCL API of Content Server. It is intended as a companion to *Content Server DQL Reference Manual*, *EMC Documentum Object Reference Manual*, *Content Server Fundamentals*, *Content Server Administrator's Guide*, and *Documentum Distributed Configuration Guide*.

## Intended audience

This manual is written for application developers and system administrators and any others who want to build a content or work-group management application using the Documentum Application Programming Interface. It assumes that you are familiar with the concepts of document processing, object-oriented programming, and client-server applications. It also assumes working knowledge of SQL.

## Conventions

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

#### Syntax conventions

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

## Terminology changes

Two common terms are changed in 5.3 and later documentation:

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

# Revision history

The following changes have been made to this document.

**Revision history**

| Revision date | Description |
|---|---|
| September 2005 | Initial publication |

# Chapter 1

# Server Methods by Task

This chapter summarizes the API methods. The API methods are organized by the tasks they can be used to accomplish. The summary includes a brief description of each method and its syntax. For a full description of a method, refer to Chapter 2, API Server Methods .

This chapter sorts the methods by the following categories:



## Methods for communicating with the server

The following methods communicate with the server to disconnect, retrieve messages, or manage explicit transactions.

**Table 1-1. Methods for communicating with Content Server**

| Method | Purpose and syntax |
|---|---|
| Abort, page 40 | Cancels all changes made in an open explicit database transaction.<br>`dmAPIExec("abort,session")` |
| Assume, page 106 | Gives ownership of an existing primary repository session to a new user.<br>`dmAPIExec("assume,session,user_auth_name, password\|login_ticket [,domain] [,user_arg]")` |
| Begintran, page 127 | Opens an explicit database transaction.<br>`dmAPIExec("begintran,session")` |
| Connect, page 155 | Establishes a repository session.<br>`dmAPIGet("connect,repository,user_auth_name,password\|ticket[,domain][,user_arg][,secure_mode][,application_token][,application_id][,force_authenticate]")` |
| Commit, page 151 | Saves changes made during an explicit database transaction and closes the transaction.<br>`dmAPIExec("commit,session")` |
| Disconnect, page 192 | Disconnects a session.<br>`dmAPIExec("disconnect,session")` |
| Encryptpass, page 201 | Encrypts a password.<br>`dmAPIGet("encryptpass,session,password")` |
| Getlogin, page 247 | Obtains a ticket that an application can substitute for the current user's password as an argument to the Connect method.<br>`dmAPIGet("getlogin,session[,user]")` |
| Getmessage, page 251 | Retrieves all error and informational messages from a session.<br>`dmAPIGet("getmessage[,session] [,severity_level]")` |
| Initcrypto, page 269 | Initializes the AEK.<br>`dmAPIExec("initcrypto,apisession[,location] [,passphrase]")` |

| Method | Purpose and syntax |
|---|---|
| Kill, page 295 | Shuts down a particular Content Server session.<br>`dmAPIExec("kill,session,session_to_kill [,immediacy_level][,message]")` |
| Listmessage, page 302 | Returns a collection comprising all messages at or above a particular severity level for a session.<br>`dmAPIGet("listmessage,session[,severity_level]")` |
| Setbatchhint, page 422 | Defines the maximum number of rows an RDBMS call can return.<br>`dmAPIExec("setbatchhint,session,size")` |
| Shutdown, page 447 | Shuts down the Content Server connected to a particular session.<br>`dmAPIExec("shutdown,session[,vengeance_level] [,delete_entry][,message][,timeout]")` |
| Trace, page 456 | Returns trace information for the server for the current session.<br>`dmAPIExec("trace,session,severity_level [,apilogfile]")` |

# Methods for system administration

The following methods administer the system.

**Table 1-2. Methods for system administration**

| Method | Purpose and syntax |
|---|---|
| Apply, page 92 | Executes an administrative function.<br>`dmAPIGet("apply,session,object_id, function_name {,argument,datatype,value}")` |
| Archive, page 98 | Sends a DM_ARCHIVE event to the repository operator.<br>`dmAPIGet("archive,session,predicate [,operator][,send_mail][,due_date] [,priority]")` |

| Method | Purpose and syntax |
|---|---|
| Audit, page 115 | Enables auditing of the specified event.<br>`dmAPIExec("audit,session[,object_id],event_name[,audit_subtypes][,controlling_app] [,policy_id[,state_name]][,sign_audit][,authentication][,event_description][,attribute_list][,esignature_required]")` |
| Authenticate, page 123 | Validates a user name and password.<br>`dmAPIExec("authenticate,session,user_auth_name,password\|ticket [,domain][,user_arg]")` |
| Changepassword, page 135 | Changes a user's password.<br>`dmAPIGet("changepassword,session, old_password,new_password")` |
| Createaudit, page 169 | Creates an audit trail entry.<br>`dmAPIGet("createaudit,session, audited_obj_id,event_name[,string_1] [,string_2][,string_3][,string_4] [,string_5][,id_1][,id_2][,id_3][,id_4][,id_5]")` |
| Dumpconnection, page 196 | Returns a list of subconnections and the repositories to which they are connected.<br>`dmAPIGet("dumpconnection[,session]")` |
| Dumploginticket, page 198 | Returns a login ticket or application access control token in a readable text format.<br>`dmAPIGet("dumploginticket,session,login_ticket\|aac_token")` |
| Flush, page 213 | Removes query caches, frees up memory for the server, or removes data dictionary information.<br>`dmAPIExec("flush,session,cache_keyword,[object_type][,reset_change_bit][,include_subtypes]")` |
| Flushcache, page 217 | Removes objects in server and client caches.<br>`dmAPIExec("flushcache,session [,discard_changed]")` |
| Flushconnectpool, page 219 | Disconnects all free connections in the connection pool and removes them from the pool.<br>`dmAPIExec("flushconnectpool")` |

| Method | Purpose and syntax |
|---|---|
| Getconnection, page 229 | Returns the subconnection identifier for an existing connection or establishes a new subconnection.<br>`dmAPIGet("getconnection,session, scoping_value[,connect_flag]")` |
| Getdocbasemap, page 235 | Returns information about all repositories known to a connection broker.<br>`dmAPIGet("getdocbasemap, session\|apisession[,protocol,host_name,port_number]")` |
| Getdocbrokermap, page 237 | Returns information about the connection brokers in the installation.<br>`dmAPIGet("getdocbrokermap,session\|apisession")` |
| Getservermap, page 256 | Returns information about the servers known to a connection broker.<br>`dmAPIGet("getservermap,session\|apisession, repository[,protocol][,host_name] [,port_number]")` |
| Iscached, page 293 | Indicates whether an object type is in the DMCL type cache.<br>`dmAPIGet("iscached,session,typecache, cache_key")` |
| Listconnection, page 300 | Returns the subconnections and the repositories associated with them for a specified session.<br>`dmAPIGet("listconnection,session")` |
| Publish_dd, page 329 | Publishes data dictionary information.<br>`dmAPIExec("publish_dd,session[,locale] [,type][,attribute][,force_publish]")` |
| Purgelocal, page 332 | Removes files copied to the local common area during a session.<br>`dmAPIExec("purgelocal,session")` |
| Reinit, page 349 | Reinitializes the main server thread (root server process) .<br>`dmAPIExec("reinit,session[,server_config_name] [,send_to_docbroker]")` |

| Method | Purpose and syntax |
|---|---|
| Restart, page 386 | Reinitializes a session's server process.<br>`dmAPIExec("restart,session [,server_config_name] [,reset_client]")` |
| Restore, page 389 | Queues a restore from archives request to the repository operator.<br>`dmAPIGet("restore,session[,predicate] [,dump_file][,operator][,send_mail] [,due_date][,priority]")` |
| Unaudit, page 466 | Stops auditing of an event.<br>`dmAPIExec("unaudit,session [,object_id] ,event_name[,controlling_app] [,policy_id ][,state_name]]")` |
| Verifyaudit, page 510 | `dmAPIExec("verifyaudit,session,object_id")` |
| Verifyesignature, page 512 | `dmAPIExec("verifyesignature,session,object_id")` |

# Methods for handling objects

The following methods manipulate objects.

**Table 1-3. Methods for handling objects**

| Method | Purpose and syntax |
|---|---|
| Adddigsignature, page 46 | Creates an audit trail entry that records the signing of a SysObject by a user.<br>`dmAPIGet("adddigsignature,session,object_id[,user_name][,reason]")` |
| Addesignature, page 48 | Creates a copy of the object's content that includes a signature.<br>`dmAPIGet("addesignature,session,object_id[,user_auth_name],password[,justification][,format_to_sign][,hash_algorithm][,presignature_hash], sig_method_name[,app_properties][,passthrough_arg1][,passthrough_arg2]")` |

| Method | Purpose and syntax |
|---|---|
| Addnote, page 54 | Attaches a dm_note (annotation) to a document.<br>`dmAPIExec("addnote,session,note_id, document_id[,keep_permanent]")` |
| Attach, page 111 | Changes the lifecycle of an object.<br>`dmAPIExec("attach,session,sysobject_id [,default\|policy_id[,position\|state_name]] [,alias_set_position\|alias_set_name]")` |
| Branch, page 131 | Simultaneously checks out a document and creates a new version branch.<br>`dmAPIGet("branch,session,object_id [,version_label]")` |
| Checkin, page 138 | Creates and saves a new version of a checked-out object and removes the lock set on the previous version by the Checkout method.<br>`dmAPIGet("checkin,session,object_id [,save_lock]{,version_label}")` |
| Checkinapp, page 141 | Creates and saves a new version of a checked-out object and removes the lock set on the previous version by the Checkout method. This method is intended for use internally and by user-written applications. It differs from Checkin by providing arguments that set the a_special_app and a_compound_architecture attributes.<br>`dmAPIGet("checkin,session,object_id [,save_lock][,version_label] [,old_compound_arch_value] [,old_special_app_value]`<br>`[,new_compound_arch_value] [,new_special_app_value]")` |
| Checkout, page 146 | Checks an object out of the repository and places a lock on the object.<br>`dmAPIGet("checkout,session,object_id[,version_label][,compound_arch_value] [,special_app_value]")` |
| Create, page 167 | Creates an object.<br>`dmAPIGet("create,session,object_type")` |

| Method | Purpose and syntax |
|---|---|
| Demote, page 176 | Demotes an object from its current normal lifecycle state to the previous normal state.<br>`dmAPIExec("demote,session,sysobject_id [,to_state\|to_position,return_to_base_flag]")`<br>`dmAPIExec("demote,session,sysobject_id ,scheduled_date,[pattern],to_state\|to_ position\|cancel[,return_to_base]")` |
| Dereference, page 181 | Obtains the object ID of a remote object pointed to by a local mirror object or replica.<br>`dmAPIGet("dereference,session,object_id")` |
| Destroy, page 186 | Removes an object from the repository.<br>`dmAPIExec("destroy,session,object_id [,force_flag]")` |
| Dump, page 194 | Returns all attribute names and values or data dictionary information for an object.<br>`dmAPIGet("dump,session,object_id\|type_ identifier")` |
| Fetch, page 209 | Retrieves an object from the repository but does not place a lock on the object.<br>`dmAPIExec("fetch,session,object_id[,type] [,persistent_cache][,consistency_check_value]")` |
| Freeze, page 221 | Marks an object as unchangeable.<br>`dmAPIExec("freeze,session,object_id [,freeze_ components]")` |
| Grant, page 258 | Creates an access control entry in an ACL object.<br>`dmAPIExec("grant,session,object_id[,accessor_ name\|alias] [,permit_type][,unused][,basic_ permission\|extended_permit][,application_ permit]")` |
| Link, page 297 | Links an object to a particular folder or cabinet.<br>`dmAPIExec("link,session,object_id, folder_ specification\|alias")` |
| Lock, page 306 | Places a database lock on an object.<br>`dmAPIExec("lock,session,object_id[,validate_ stamp]")` |

| Method | Purpose and syntax |
|---|---|
| Locate, page 304 | Returns the integer index for a particular repeating attribute value.<br>`dmAPIGet("locate,session,object_id,attribute,value")` |
| Mark, page 310 | Assigns one or more symbolic labels to an object.<br>`dmAPIExec("mark,session,object_id,label{,label}")` |
| Offset, page 318 | Returns the integer index for an attribute relative to the number of attributes in a particular object.<br>`dmAPIGet("offset,session,object_id,attribute")` |
| Promote, page 323 | Promotes an object from its current normal lifecycle state to the next normal state.<br>`dmAPIExec("promote,session,sysobject_id[,to_state \|to_position,test_only_flag,override_flag]")`<br>`dmAPIExec("promote,session,sysobject_id,scheduled_date[,pattern] ,to_state\|to_position\|cancel[,override_flag]")` |
| Prune, page 326 | Removes versions of an object.<br>`dmAPIExec("prune,session,object_id[,keepSLabel]")` |
| Refresh, page 344 | Refreshes the attribute values of a mirror object or replica.<br>`dmAPIExec("refresh,session,object_id")` |
| Removenote, page 358 | Detaches an annotation from an object.<br>`dmAPIExec("removenote,session,note_id,document_id")` |
| Reset, page 380 | Resets an object's error status.<br>`dmAPIExec("reset,session,object_id")` |
| Resolvealias, page 382 | Resolves an alias, returning a user name, group, or path specification.<br>`dmAPIGet("resolvealias,session[,object_id],alias")` |

| Method | Purpose and syntax |
|---|---|
| Resume, page 392 | Restarts a halted workflow, paused activity, or paused work item, or resumes the lifecycle of an object.<br>`dmAPIExec("resume,session,object_id [,activity_seq_no]")`<br>`dmAPIExec("resume,session,sysobject_id [,from_state\|from_position,return_to_base_flag ,test_only_flag,override_flag]")`<br>`dmAPIExec("resume,session,sysobject_id ,scheduled_date,[pattern],[from_state] \|from_position\|cancel ,return_to_base_flag ,override_flag]")` |
| Revert, page 398 | Re-fetches an object from the repository.<br>`dmAPIExec("revert,session,object_id[,acl_only]")` |
| Revoke, page 400 | Removes the access control entries or an extended permission for a user or group.<br>`dmAPIExec("revoke,session,object_id,accessor_name\|alias [,permit_type][,unused] [,basic_permit\|extended_permit\|application_permit]")` |
| Save, page 405 | Saves an object, overwriting the object currently in the repository.<br>`dmAPIExec("save,session,object_id[,save_lock] {,version_label}")` |
| Saveasnew, page 409 | Creates a copy of an object.<br>`dmAPIGet("saveasnew,session,object_id ,share_content[,keep_storage_areas]")` |
| Signoff, page 449 | Creates an audit trail entry containing signoff information for an object.<br>`dmAPIExec("signoff,session,object_id [,user_auth_name],password[,reason]")` |

| Method | Purpose and syntax |
|---|---|
| Suspend, page 453 | Suspends the lifecycle of an object by moving the object to the exception state for its current state.<br>`dmAPIExec("suspend,session,sysobject_id [,from_state\|from_position,test_only_flag ,override_flag]")`<br>`dmAPIExec("suspend,session,sysobject_id ,scheduled_date,[pattern],from_state \|from_position\|cancel[,override_flag]")` |
| Type, page 464 | Returns the object describing a particular object type or attribute.<br>`dmAPIGet("type,session,object_type [,attribute][,business_policy[,state]]")` |
| Unfreeze, page 469 | Unfreezes a frozen object.<br>`dmAPIExec("unfreeze,session,object_id [,thaw_components]")` |
| Unlink, page 474 | Unlinks an object from a folder or cabinet.<br>`dmAPIExec("unlink,session,object_id, folder_specification\|alias")` |
| Unlock, page 477 | Removes an intention lock from an object.<br>`dmAPIExec("unlock,session,object_id [,send_ mail][,compound_arch_value] [,special_app_ value]")` |
| Unmark, page 480 | Removes one or more symbolic labels from an object.<br>`dmAPIExec("unmark,session,object_id,label {,label}")` |
| Useacl, page 491 | Assigns a default ACL to an object.<br>`dmAPIExec("useacl,session,object_id, folder\|user\|type\|none")` |

# Methods for retrieving and setting attributes

The following methods manipulate object attributes.

**Table 1-4. Methods for retrieving and setting attributes**

| Method | Purpose and syntax |
|---|---|
| Append, page 77 | Adds a new value to a repeating attribute.<br>`dmAPISet("append,session,object_id, attribute[,pattern]","value")` |
| Count, page 164 | Counts the number of attributes for an object.<br>`dmAPIGet("count,session,object_id")` |
| Datatype, page 171 | Returns the datatype of an attribute.<br>`dmAPIGet("datatype,session,object_id, attribute")` |
| Describe, page 184 | Returns a description of the attributes for an object type or registered table.<br>`dmAPIGet("describe,session,type, object_type")`<br>`dmAPIGet("describe,session,table,table_name")` |
| Get, page 224 | Returns the value of an attribute.<br>`dmAPIGet("get,session,object_id\|type_ identifier,attribute [[index]][,pattern]")` |
| Insert, page 271 | Inserts a value into a repeating attribute.<br>`dmAPISet("insert,session,object_id, attribute[[index]][,pattern]","value")` |
| Locate, page 304 | Returns the integer index for a repeating attribute of a given value.<br>`dmAPIGet("locate,session,object_id, attribute,value")` |
| Remove, page 351 | Removes a value from a repeating attribute.<br>`dmAPIExec("remove,session,object_id, attribute[index]")` |
| Repeating, page 377 | Determines whether an attribute is repeating or single-valued.<br>`dmAPIGet("repeating,session,object_id, attribute")` |

| Method | Purpose and syntax |
|---|---|
| Set, page 416 | Sets the value of an attribute.<br>`dmAPISet("set,session,object_id,attribute [[index]][,pattern]","value\|alias")`<br>**Note:** *alias* is only allowed when *attribute* is one of: owner_name, acl_name, or acl_domain. |
| Truncate, page 462 | Removes all values for a repeating attribute.<br>`dmAPIExec("truncate,session,object_id, attribute")` |
| Values, page 497 | Returns the number of values for an attribute.<br>`dmAPIGet("values,session,object_id, attribute")` |

# Methods for searching the repository

The following methods execute, process, and close DQL queries.

**Table 1-5. Methods for searching the repository**

| Method | Purpose and syntax |
|---|---|
| Cachequery, page 133 | Executes a DQL query, returns the identifier for the resulting collection, and caches the results across sessions.<br>`dmAPIGet("cachequery,session,dql_query")` |
| Close, page 149 | Closes a query.<br>`dmAPIExec("close,session, collection_ identifier")` |
| Execquery, page 203 | Executes a DQL statement longer than 255 characters.<br>`dmAPIExec("execquery,session, [readquery_ flag],dql_query")` |
| Execsql, page 205 | Executes any SQL statement except a SELECT statement.<br>`dmAPIExec("execsql,session,sql_statement")` |
| Getlastcoll, page 245 | Returns the ID of the most recent collection generated by the server.<br>`dmAPIGet("getlastcoll,session")` |

| Method | Purpose and syntax |
|---|---|
| Id, page 267 | Returns the ID of the object that satisfies a particular search condition.<br>`dmAPIGet("id,session,search_condition")` |
| Next, page 316 | Returns an item from a collection.<br>`dmAPIExec("next,session, collection_identifier")` |
| Query, page 337 | Executes a DQL query and returns the identifier for the resulting collection. The method allows you to modify data in the repository during query result processing.<br>`dmAPIGet("query,session,dql_query")` |
| Query_cmd, page 334 | Executes a DQL query and allows you to mark the query results for persistent client caching.<br>`dmAPIGet("query_cmd,session[,read_only][,persistent_cache] [,consistency_check_value][,unused][,unused][,unused] ,dql_query")` |
| Readquery, page 342 | Executes a DQL query and returns the identifier for the resulting collection. The method does not allow you to modify the repository during query result processing.<br>`dmAPIGet("readquery,session,dql_query")` |
| Retrieve, page 396 | Fetches the object that satisfies a search condition and returns the object's ID.<br>`dmAPIGet("retrieve,session,search_condition")` |

# Methods for handling content objects

The following methods manipulate content objects.

**Table 1-6. Methods fo handling content oObjects**

| Method | Purpose and sSyntax |
|---|---|
| Addrendition, page 66 | Adds a rendition to a document. (Rendition information is stored with content objects.)<br>`dmAPIExec("addrendition,session,object_id, file_name,format[,page_number][,atomic] [,keep_flag][,page_modifier] [,batch_ flag][,other_file]")` |
| Appendcontent, page 80 | Appends information in working memory as the last content file for a document. (Used in applications to append content to multipaged documents.)<br>`dmAPISet("appendcontent,session,object_id [,length][,set_resource]","content")` |
| Appendfile, page 82 | Appends a content file to the list of files for a document.<br>`dmAPIExec("appendfile,session,object_id, file_name[,other_file]")` |
| Bindfile, page 129 | Binds a content object belonging to one document to another so the content belongs to both.<br>`dmAPIExec("bindfile,session,object_id ,page_number,src_id,src_page_number")` |
| Getcontent, page 231 | Retrieves a content file from the repository and places it in working memory.<br>`dmAPIGet("getcontent,session,object_id [,format][,page_number][,page_ modifier][,get_resource]")` |
| Getfile, page 241 | Returns the name of the file containing content in a particular format.<br>`dmAPIGet("getfile,session,object_id [,file_name][,format][,page_number] [,get_resource][,page_modifier]")` |
| Getpath, page 253 | Returns the internal path specification to a content file.<br>`dmAPIGet("getpath,session,object_ id[,page][,format][,page_modifier][,get_ resource]")` |

| Method | Purpose and sSyntax |
|---|---|
| Insertcontent, page 274 | Inserts information in working memory as content in a document. (Used by applications to insert data into multipaged documents.)<br>`dmAPISet("insertcontent,session,object_id[,page_number][,length][,set_resource]","content")` |
| Insertfile, page 277 | Inserts content in a document. (Used with multipaged documents.)<br>`dmAPIExec("insertfile,session,object_id,file_name[,page_number][,other_file]")` |
| Mount, page 312 | Sets the content location for an external store object.<br>`dmAPIExec("mount,session,object_id,path")` |
| Removecontent, page 355 | Removes content from an object.<br>`dmAPIExec("removecontent,session,object_id[,page_number]")` |
| Removerendition, page 368 | Removes a rendition from a document.<br>`dmAPIExec("removerendition,session,object_id,format[,page_number][,atomic] [,page_modifier]")` |
| Seek, page 413 | Sets the next position (byte range) for reading a content collection.<br>`dmAPIExec("seek,session,collection,position[,direction]")` |
| Setcontent, page 424 | Sets data in working memory as new content or uses it to replace content. (Used in applications to set the content of an object.)<br>`dmAPISet("setcontent,session,object_id][,format[,page_number][,length][,set_resource]","content")` |
| Setcontentattrs, page 427 | `dmAPIExec("setcontentattrs,session,object_id,format[,page_number][,page_modifier],parameters")` |

| Method | Purpose and sSyntax |
|---|---|
| Setfile, page 434 | Adds a file as primary content to a document.<br>`dmAPIExec("setfile,session,object_id, file_name[,page_num][,format] [,other_file]")` |
| Setpath, page 439 | Adds a file in external storage as primary content to an document.<br>`dmAPIExec("setpath,session,object_id, file_name[,page_number\|format] [,other_file]")` |

# Methods for printing documents

The following methods perform print operations.

**Table 1-7. Methods for printing documents**

| Method | Purpose and syntax |
|---|---|
| Lpq, page 308 | Shows the print queue for a printer.<br>`dmAPIGet("lpq,session[,printer]")` |
| Print, page 321 | Sends a document to a printer.<br>`dmAPIGet("print,session,object_id[,printer][,print_cover][,save_output][,num_copies][,starting_content_page] [,ending_content_page]")` |
| Unprint, page 482 | Removes a print job from a print queue.<br>`dmAPIExec("unprint,session[,printer_name],job_id")` |

# Methods for handling virtual documents

The following methods manipulate the components of virtual documents.

**Table 1-8. Methods for handling virtual documents**

| Method | Purpose and syntax |
|---|---|
| Appendpart, page 84 | Adds a new component to the end of a document's list of components.<br>`dmAPIGet("appendpart,session,document_id, component_id[,version_label] [,use_node_ ver_label][,follow_assembly] [,copy_child] [,containment_type,containment_desc]")` |
| Assemble, page 100 | Creates an assembly for a virtual document.<br>`dmAPIExec("assemble,session,document_id [,virtual_doc_id][,interrupt_freq] [,qualification][,nodesort_list]")` |
| Disassemble, page 190 | Destroys an assembly for a virtual document.<br>`dmAPIGet("disassemble,session,document_id")` |
| Freeze, page 221 | Marks the components of a virtual document's assembly and the virtual document itself as unchangeable.<br>`dmAPIExec("freeze,session,object_id [,freeze_ components]")` |
| Insertpart, page 280 | Inserts a component at a particular position in a virtual document's list of components.<br>`dmAPIGet("insertpart,session,document_id, component_id[,version_label], containment_ id\|order_no,orderno_flag] [[,use_node_ver_ label][,follow_assembly] [,copy_child] [,containment_type,containment_desc]")` |
| Removepart, page 364 | Removes a component from a virtual document.<br>`dmAPIExec("removepart,session,document_id ,containment_id\|order_no[,orderno_flag]")` |
| Setdoc, page 432 | Indicates whether an object is a virtual document.<br>`dmAPIExec("setdoc,session,object_id, is_virtual_doc")` |
| Unfreeze, page 469 | Unfreezes a frozen assembly and the associated virtual document.<br>`dmAPIExec("unfreeze,session,object_id [,thaw_components]")` |

| Method | Purpose and syntax |
|---|---|
| Updatepart, page 486 | Modifies a component of a virtual document.<br>`dmAPIExec("updatepart,session,virtdoc_id, component_id[,version_label][,order_no] [,use_ node_ver_label][,follow_assembly] [,copy_child] [,containment_type,containment_desc]")` |
| Vdmpath, page 501 | Returns paths to a component in one or more virtual documents. This method provides faster performance than Vdmpathdql but less flexibility in choosing the paths.<br>`dmAPIGet("vdmpath,session,component_id [,root_id][,shortest_path]{,version_list}")` |
| Vdmpathdql, page 506 | Returns paths to a component in one or more virtual documents. This method provides more flexibility in choosing the paths but slower performance.<br>`dmAPIGet("vdmpathdql,session,component_id [,root_id][,shortest_path][,type] [,binding_ condition][,nodesort_by]")` |

# Methods for handling lifecycles

The following methods manage lifecycles.

**Table 1-9. Methods for handling lifecycles**

| Method | Purpose and syntax |
|---|---|
| Appendstate, page 90 | Appends a state to a lifecycle.<br>`dmAPIGet("appendstate,session,object_id")` |
| Insertstate, page 287 | Adds a state within a lifecycle.<br>`DmAPIExec("insertstate,session,policy_id [,position\|state_name]")` |
| Install, page 289 | Installs a lifecycle.<br>`dmAPIExec("install,session,object_id [,notify_flag]")` |

| Method | Purpose and syntax |
|---|---|
| Movestate, page 314 | Rearranges the state definitions within a lifecycle definition<br>`dmAPIExec("movestate,session,object_id, old_position,new_position")` |
| Removestate, page 373 | Removes a state from a lifecycle.<br>`dmAPIexec("removestate,session,object_id [,position\|state_name[,force_flag]]")` |
| Uninstall, page 472 | Moves a lifecycle from the installed state to the validated state.<br>`dmAPIExec("uninstall,session,object_id [,notify_flag]")` |
| Validate, page 494 | Determines whether a lifecycle is valid.<br>`dmAPIExec("validate,session,object_id")` |

# Methods for handling workflows

The following methods manage running workflows.

**Table 1-10. Methods for handling workflows**

| Method | Purpose and syntax |
|---|---|
| Abort, page 40 | Aborts a workflow.<br>`dmAPIExec("abort,session,workflow_id")` |
| Addpackage, page 56 | Attaches a package to the start activity of a workflow.<br>`dmAPIGet("addpackage,session,object_id, start_activity_name,input_port_name, package_name,component_ID {,component_ID}")` |
| Execute, page 207 | Starts a workflow.<br>`dmAPIExec("execute,session,workflow_id")` |
| Halt, page 265 | Halts a workflow or activity.<br>`dmAPIExec("halt,session,workflow_id [,activity_ sequence_number[,suspend_interval]]")` |
| Invalidate, page 291 | Sets a validated workflow or activity definition to draft state.<br>`dmAPIExec("invalidate,session,object_id")` |

| Method | Purpose and syntax |
|---|---|
| Queue, page 339 | Posts an event to a workflow.<br>`dmAPIGet("queue,session,workflow_id, [supervisor] ,event_type,[priority], [send_mail],[due_date] [,message]")` |
| Removepackage, page 360 | Removes a package from a start activity of a workflow.<br>`dmAPIExec("removepackage,session,object_id, port_name,package_name")` |
| Restart, page 386 | Restarts a halted workflow or activity.<br>`dmAPIExec("restart,session,workflow_id [,activity_sequence_number]")` |
| Resume, page 392 | Resumes a halted workflow or a paused activity.<br>`dmAPIExec("resume,session,workflow_id [,activity_sequence_number]")` |
| Setoutput, page 437 | Defines the output port for manual transition of a work item.<br>`dmAPIExec("setoutput,session,workitem_id {,output_port_name}")` |
| Setperformers, page 441 | Adds performers chosen from a group or all users at runtime to an activity.<br>`dmAPIExec("setperformers,session,object_id, activity_name [,performer_list]")` |
| Setsupervisor, page 445 | Changes the supervisor of a workflow.<br>`dmAPIExec("setsupervisor,session,workflow_ id,new_supervisor")` |

# Methods for handling workflow definitions

The following methods manage workflow definitions.

**Table 1-11. Methods for handling workflow definitions**

| Method | Purpose and syntax |
|---|---|
| Addactivity, page 44 | Adds an activity to a process definition.<br>`dmAPIExec("addactivity,session,object_id ,activity_identifier,activity_definition_id [,activity_type][,activity_priority]")` |
| Addlink, page 52 | Creates a link connecting an output port of a source activity to an input port of a destination activity.<br>`dmAPIExec("addlink,session,object_id ,link_identifier,source_activity,source_port ,destination_activity,destination_port")` |
| Install, page 289 | Installs a validated process definition.<br>`dmAPIExec("install,session,object_id [,install_activity][,resume]")` |
| Removeactivity, page 353 | Removes an activity from a process definition.<br>`dmAPIExec("removeactivity,session,object_id ,activity_identifier")` |
| Removelink, page 357 | Removes a link.<br>`dmAPIExec("removelink,session,object_id ,link_identifier")` |
| Uninstall, page 472 | Uninstalls a validated process definition.<br>`dmAPIExec("uninstall,session,object_id")` |
| Validate, page 494 | Validates a process definition.<br>`dmAPIExec("validate,session,object_id [,check_activity]')` |

# Methods for handling activity definitions

The following methods manage activity definitions.

**Table 1-12. Methods for handling activity definitions**

| Method | Purpose and syntax |
|---|---|
| Addpackageinfo, page 60 | Adds a package to an activity port definition.<br>`dmAPIExec("addpackageinfo,session,activity_id,port_name ,package_name,package_type[,package_id[,package_label]][,package_operation][,r_package_flag]")` |
| Addport, page 64 | Adds a port to an activity definition.<br>`dmAPIExec("addport,session,activity_id,port_name,port_type")` |
| Addroutecase, page 72 | Adds a route case to an activity definition.<br>`dmAPIExec("addroutecase,session,object_id,route_case_identifier,routing_condition{,output_port}")`<br>`dmAPIExec("addroutecase,session,object_id ,EXCEPTIONAL{,output_port}")` |
| Install, page 289 | Installs a validated activity definition.<br>`dmAPIExec("install,session,object_id[,install_activity][,resume]")` |
| Removepackageinfo, page 362 | Removes a package from a port.<br>`dmAPIExec("removepackageinfo,session,object_id,port_name,package_name")` |
| Removeport, page 367 | Removes a port added with the Addport method.<br>`dmAPIExec("removeport,session,object_id,port_name")` |
| Removeroutecase, page 371 | Removes a route case from a condition list.<br>`dmAPIExec("removeroutecase,session,object_id,route_case_identifier")` |
| Uninstall, page 472 | Uninstalls a validated activity definition.<br>`dmAPIExec("uninstall,session,object_id")` |
| Validate, page 494 | Validates an activity definition.<br>`dmAPIExec("validate,session,object_id[,check_activity]")` |

# Methods for handling work items

The following methods manage work items.

**Table 1-13. Methods for handling work items**

| Method | Purpose and syntax |
|---|---|
| Acquire, page 42 | Acquires a task for a user.<br>`dmAPIExec("acquire,session,workitem_id")` |
| Addpackage, page 56 | Attaches a package to a work item.<br>`dmAPIGet("addpackage,session,object_id, start_activity_name,input_port_name, package_name,component_ID{,component_ID}")` |
| Complete, page 153 | Marks a work item as finished.<br>`dmAPIExec("complete,session,workitem_ id[,[return_value] ,[os_error],[result_ id][,user_time][,user_cost]")` |
| Delegate, page 174 | Reassigns a work item.<br>`dmAPIExec("delegate,session,workitem_id, user_name")` |
| Pause, page 320 | Suspends a work item.<br>`dmAPIExec("pause,session,workitem_id")` |
| Removepackage, page 360 | Detaches a package from a start activity.<br>`dmAPIExec("removepackagesession,object_id, port_name,package_name")` |
| Repeat, page 375 | Allows a user to repeat a work item.<br>`dmAPIGet("repeat,session,workitem_id {,user_name\|group_name}")` |
| Resume, page 392 | Restarts a halted workflow, paused activity, or paused task.<br>`dmAPIExec("resume,session,workitem_id")` |

# Methods for managing inboxes

The following methods manage inboxes (user queues).

**Table 1-14. Methods for managing inboxes**

| Method | Purpose and syntax |
|---|---|
| Anyevents, page 75 | Determines whether there are any new items in the user's queue.<br>`dmAPIExec("anyevents,session")` |
| Dequeue, page 179 | Removes an item from a user's queue that was placed in the queue using the Queue method.<br>`dmAPIExec("dequeue,session,stamp")` |
| Getevents, page 239 | Returns all items in the user's queue.<br>`dmAPIGet("getevents,session")` |
| Queue, page 339 | Places an item in a user's queue.<br>`dmAPIGet("queue,session,object_id, queue_owner,event,priority[,send_mail] [,due_date],message")` |
| Register, page 346 | Registers a user to receive event notification for a specific event.<br>`dmAPISet("register,session,object_id,event [,priority][,send_mail]","message")` |
| Unregister, page 484 | Removes an event registration.<br>`dmAPIExec("unregister,session,object_id, event")` |

# Chapter 2

# API Server Methods

This chapter contains full descriptions of the Content Server DMCL API methods. The description of each method includes:

- Purpose
- Syntax
- Argument descriptions
- Return value
- Usage notes
- Example

The method descriptions are ordered alphabetically.

**General Note on Syntax:** The arguments to all methods are positional. That is, if an argument is optional and not the final argument in the argument list, you must include the comma that is its place-holder in the list whether you include the argument or not. If the argument is the final argument, it is not necessary to include the comma if you do not include the argument.

# Abort

## Purpose

Cancels an explicit transaction or terminates a workflow.

## Syntax

```
dmAPIExec("abort,session[,workflow_id]")
```

## Arguments

**Table 2-1. Abort method arguments**

| Argument | Description |
|---|---|
| *session* | Identifies an open repository session. |
| *workflow_id* | Identifies the workflow to abort. Use the workflow's object ID. |

## Return value

The Abort method returns TRUE if successful or FALSE if unsuccessful.

## General notes

Use the Abort method when you want to close an explicit database transaction and cancel any changes that were made during the transaction or to terminate a workflow.

### Aborting transactions

An explicit database transaction is a transaction that was opened by executing the Begintran method.

If the Abort method returns FALSE, it indicates a failure to accomplish the rollback in the underlying database. This may result in an inconsistent database.

### Aborting workflows

You cannot abort a workflow that has active automatic work items. Only the workflow supervisor or a user with Superuser or Sysadmin privileges can abort a workflow.

The Abort method sets the workflow's r_runtime_state to terminated.

Using Abort does not delete the workflow instance or associated work items and packages. You must use a Destroy method to remove the workflow object and the work items and packages associated with the workflows.

## Related methods

For transactions:

Begintran, page 127
Commit, page 151

For workflows:

Halt, page 265

## Example

The following example begins an explicit transaction, creates a document within the transaction, and aborts the transaction.

```
status = dmAPIExec("begintran,s0")

    doc_id = dmAPIGet("create,s0,dm_document");
       if (doc_id == NULL)
       {  err_mess = dmAPIGet("getmessage,s0")
          print err_mess
          exit
       }

    status = dmAPISet("set,s0,last,object_name", "my_document")

    status = dmAPIExec("save,s0,last")
      if (status != 1)
      {  err_mess = dmAPIGet("getmessage,s0")
         print err_mess
         exit
      }

status = dmAPIExec("abort,s0")
 if (status != 1)
 {  err_mess = dmAPIGet("getmessage,s0")
    print err_mess
    exit
 }
```

The following example uses the Abort method to abort a workflow.

```
status = dmAPIExec("abort,s0," wflow_id)
```

# Acquire

## Purpose

Acquires a work item for the current user.

## Syntax

```
dmAPIExec("acquire,session,workitem_id")
```

## Arguments

**Table 2-2. Acquire method arguments**

| Argument | Description |
|---|---|
| *session* | Identifies an open repository session. |
| *workitem_id* | Identifies the work item. Use the work item's object ID or an indirect reference (*@object_id*) that points to the object. |

## Return value

The Acquire method returns TRUE if successful or FALSE if unsuccessful.

## Usage notes

Using the Acquire method has the following conditions:

- The workflow must be in the running state, the activity that created the work item must be in the installed state, and the specified work item must be in the dormant state.
- By default, you must be one of the following:
  - The work item's assigned performer, or if the assigned performer is a group, a member of that group
  - The workflow supervisor
  - A user with either Sysadmin or Superuser privileges

If workflow_security_disabled, a server.ini key, is set to T (TRUE), any user can issue an Acquire method against a particular work item. By default, the key set to F, which enables the restrictions listed above.

The Acquire method performs the following operations:

- Changes the state of the specified work item from dormant to acquired
- Sets the actual_start_date attribute of the dmi_queue_item object (This is the object that represents the queued inbox item.)
- Sets the work item's r_performer_name attribute to the current user.

If the task's assigned performer is a group, when a member of that group acquires the task, the task is removed from the group's queue and placed in the individual group member's queue.

## Related methods

Complete, page 153
Delegate, page 174

## Example

```
status = dmAPIExec("acquire,s0," rid)
   if (status != 1 )
      Print  "Error: The work item was not acquired"
else
   Print "The work item was successfully acquired"
```

# Addactivity

## Purpose

Adds a new activity to a process definition.

## Syntax

```
dmAPIExec("addactivity,session,process_id,
activity_identifier,activity_definition_id[,activity_
type[,activity_priority]]")
```

## Arguments

**Table 2-3. Addactivity method arguments**

| Argument | Description |
|---|---|
| *session* | Identifies an open repository session. |
| *process_id* | Identifies the process to which you want to add the activity. Use the object ID of the process. |
| *activity_identifier* | Identifies the activity. The name in *activity_identifier* must be unique within the process definition. |
| *activity_definition_id* | Refers to a dm_activity instance. Use the object ID of the activity. |
| *activity_type* | Identifies the type of the activity (step, begin, or end). The default value is step. |
| *activity_priority* | Priority of the activity. The value can be any integer value. |

## Return value

The Addactivity method returns TRUE if successful or FALSE if unsuccessful.

## Usage notes

Use the Addactivity method to add a new activity to a workflow process definition. After you use Addactivity, use the Addlink method to link the new activity's ports in the process definition object.

## Related methods

Addlink, page 52
Removeactivity, page 353

## Example

```
status = dmAPIExec("addactivity,"
   session ",4b00007b80000100,original_request,
   4c00007b80000100,begin,0")
```

# Adddigsignature

## Purpose

Creates an audit trail entry that records the signing of a SysObject by a user.

## Syntax

```
dmAPIGet("adddigsignature,session,object_id[,user_name][,reason]")
```

## Arguments

**Table 2-4. Adddigsignature method arguments**

| Argument | Description |
|---|---|
| *session* | Identifies an open repository session. |
| *object_id* | Object ID of the SysObject signed by the user. Use the object's ID or an indirect reference (*@object_id*) that points to the object. |
| *user_name* | Name of the user who signed the SysObject. If not included, the default is the currently logged-in user. |
| *reason* | User-defined explanation about why the object was signed. The explanation can be a maximum of 200 characters. |

## Return value

Adddigsignature returns the object ID of the audit trail entry.

## Usage notes

Before executing Adddigsignature, the object identified in object_id must be checked in. You cannot execute Adddigsignature on a checked out object.

The logged-in user executing this method must have at least Relate permission on the object.

Adddigsignature is intended for use in applications that use a digital signature format, such as PKCS #7, XML Signature, or PDF Signature, to sign documents. After the document is signed and checked in, the application calls Adddigsignature to record the signing in an audit trail entry.

Executing the method creates an audit trial entry with the event name dm_adddigsignature (Add Digital Signature). The attributes in the audit trail entry identify the user who signed the object, the date and time at which the object was signed (server time and UTC time), and a reason for signing if one was specified in the method.

**Note:** If you want the audit trail entry to be signed by the server, issue an Audit method that designates signing for the event. For example, the following Audit method registers all dm_adddigsignature events on SysObjects and SysObject subtypes for signature:

```
audit,s0,SysObject_type_object_id,dm_adddigsignature,T,,T
```

The first Boolean argument in the example (set to T) tells the server to include SysObject subtypes. The second tells the server to sign the audit trail entries.

## Related methods

Addesignature, page 48
Audit, page 115
Signoff, page 449

## Example

```
<extract signature from document and verify>
   <if verification succeeds>
   status=dmAPISet("set,S0,0900000183241ac3,a_is_signed",T)
<then check in the document>
new_doc_id=dmAPIGet("checkin,S0,0900000183241ac3")
<if doc was signed>
audit_id=dmAPIGet("adddigsignature,S0,0900000183241ac3")
```

# Addesignature

## Purpose

Adds a signature to an object's content and generates an audit trail entry recording the operation.

## Syntax

```
dmAPIGet("addesignature,session,object_id
[,user_auth_name],password[,justification][,format_to_sign]
[,hash_algorithm][,presignature_hash],
sig_method_name[,app_properties][,passthrough_arg1][,passthrough_arg2]")
```

## Arguments

**Table 2-5. Addesignature method arguments**

| Argument | Description |
|---|---|
| *session* | Identifies an open repository session. |
| *object_id* | Identifies the object to be signed. Use the object's object ID. |
| *user_auth_name* | Identifies the user signing the object. Use the value in the user's user_os_name attribute. If specified, the value must match that of the user for the current session.<br><br>The default is the the current user's user_os_name value. |
| *password* | Specifies the user's password. Valid password formats are:<br><br>*password*<br>DM_PLUGIN=*plugin_id/password*<br><br>where *plugin_id* is the plugin identifier for an authentication plugin.<br><br>If the password is encrypted, the format for the password argument is:<br><br>DM_ENCR_PASS=*encrypted_password*<br><br>where *encrypted_password* is one of the following in encrypted form:<br><br>*password*<br>DM_PLUGIN=*plugin_id/password* |

| Argument | Description |
|---|---|
| | Refer to the Usage notes for details.<br><br>A password may not exceed 8,192 characters in length. |
| *justification* | The justification text. This must be enclosed in single quotes. If the text includes single quotes, the quotes must be escaped with a single quote. |
| *format_to_sign* | The format of the content to be signed. Use the name of the format's format object.<br><br>The default is pdf. |
| *hash_algorithm* | Name of the hashing algorithm to use to generate the hash value for the audit trail entry. The only supported algorithm is SHA-1. The argument defaults to SHA-1 if unspecified. |
| *presignature_hash* | Hash value of the document prior to signing. The argument must be in the following format:<br><br>`'{algorithm_name}/{format_name}:hash_value_in_hex'`<br><br>Refer to the Usage notes for an expanded explanation and an example. |
| *sig_method_name* | Name of the method object representing the method used to generate the signed copy. The default value is esign_pdf. |
| *app_properties* | A comma-delimited list of attribute names and values to pass to the signature creation method. The list is used to populate user-defined fields on the signature page.<br><br>The list must be enclosed in single quotes. Attribute values in the list must be enclosed in escaped single quotes. For example:<br><br>`'Attribute1=''attr1_value'',Attribute2=''attr2_value'''` |
| *passthrough_arg1* and *passthrough_arg2* | These arguments pass data specific to the method identified in *sig_method_name*. |

## Return value

Addesignature returns the object ID of the audit trail object created by the method.

## Usage notes

You must have installed the server with a Trusted Content Services license to use this method.

You must have at least Browse permission on the object identified in *object_id* to sign the object.

You cannot execute this method against an object that is checked out of the repository.

If an object has multiple primary content files, only the first page is affected by the Addesignature method. The method does not operate on second or subsequent pages or any renditions associated with these pages.

*format_to_sign* identifies the format of the content to which the signature will actually be added. The format-to-be-signed is not necessarily the format of the document's primary content. The format-to-be-signed can be a rendition of the primary content. For example, if you want to sign a Word document and are using the default signature functionality, you must create a PDF rendition of the document and attach that rendition to page number zero in the document. PDF is the format to be signed. The default method attaches the signature page to the PDF rendition.

When you issue an Addesignature method, either the object's primary content page 0 or a rendition of that page must be in the format identified in *format_to_sign*. The page_modifier attribute for that content page or rendition must be blank.

After the signature page is generated, the method replaces the source content with the generated signature page. If *format_to_sign* identifies the primary content format, the signed content replaces that content in the object. If *format_to_sign* identifies a rendition format, the method replaces the rendition with the signed rendition.

**Note:** If the signature is the first signature on the version, the source content is appended back to the document as a rendition with the page modifier set to dm_sig_source.

For example, suppose the format of a document's primary content is Maker (for Framemaker files) and that its first primary page (page 0) has a PDF rendition. If *format_to_sign* is PDF, the method signs the PDF rendition and replaces the PDF rendition with the signed rendition.

Using the *presignature_hash* argument lets you or the application ensure that the object that the user is signing is the same as the object in the repository. Using that argument, you pass the name of the hashing algorithm, the format of the content to be signed, and the hash value of the source content. The application is responsible for generating the hash value passed in the argument. Content Server will use the specified hashing algorithm to hash the copy of the object in the repository and compare it to the passed-in hash value. If they are different, the method fails.

**Note:** Content Server currently supports only the SHA-1 hashing algorithm.

## Related methods

Adddigsignature, page 46
Signoff, page 449

## Example

```
auditID=dmAPIGet("addesignature,c,090000016473a21b,johndoe,
  <jdpasswrd>,'Approved for Release',,
  ,{SHA-1}/{pdf}:af7f76fb9960a05a1341c1777b48f1df,
  ,'approval_date=''<date>''")
```

# Addlink

## Purpose

Creates a link within a workflow definition that connects an output port of a source activity to an input port of a destination activity.

## Syntax

```
dmAPIExec("addlink,session,process_id,link_identifier,
source_activity,source_port,destination_activity
,destination_port")
```

## Arguments

**Table 2-6. Addlink method arguments**

| Argument | Description |
|---|---|
| *session* | Identifies an open repository session. |
| *process_id* | Identifies the process definition. Use the process' object ID. |
| *link_identifier* | Uniquely identifies the link within the process object. |
| *source_activity* | Identifies the source activity to link. Use the activity added with the Addactivity method. |
| *source_port* | Identifies the source port to link. |
| *destination_activity* | Identifies the destination activity to link. Use the activity added with the Addactivity method. |
| *destination_port* | Identifies the destination port to link. |

## Return value

The Addlink method returns TRUE if successful or FALSE if unsuccessful.

## Usage notes

For activities added to a process definition with the Addactivity method, the Addlink method creates a link that connects an output port of a source activity to an input port of a destination activity. The designer creates a unique identifier for each link.

## Related methods

Removelink, page 357

## Example

```
status = dmAPIExec("addlink,s0,4b00007b80000100,link_4,"
act_name3 ",port_o1," act_name4 ",port_i2")
```

# Addnote

## Purpose

Adds an annotation to a specified document (or other SysObject) or a workflow package.

## Syntax

```
dmAPIExec("addnote,session,annotation_id,object_id
[,keep_permanent]")
```

## Arguments

**Table 2-7. Addnote method arguments**

| Argument | Description |
|---|---|
| *session* | Identifies an open repository session. |
| *annotation_id* | Defines which annotation you want to attach to the specified document. Use the annotation's object ID or an indirect reference (*@object_id*) that points to the annotation. |
| *object_id* | Identifies document or package to which you are attaching the annotation. Use the object or package's ID. |
| *keep_permanent* | Indicates whether you want the annotation to remain attached to the document when users version the document or whether the note travels with the workflow package to all subsequent performers. The default is FALSE. |

## Return value

The Addnote method returns TRUE if successful or FALSE if unsuccessful.

## Usage notes

The Addnote method attaches an existing annotation (dm_note object) to a document (or other SysObject) or to a workflow package. It does not actually create the annotation. You must do that first with the Create method.

To use Addnote, the following conditions must be true:

- You must have at least RELATE permit for the document to which you are attaching the annotation or be a performer of the workflow to which the package belongs.
- You must be the owner of the annotation or have at least Write permission for the annotation.
- You must be a performer in the workflow to which the package belongs.

After issuing the Addnote method, you must issue a Save method to create the dm_relation object that links the annotation to the document.

You can attach multiple annotations to a single document. Conversely, you can attach one annotation to multiple documents.

## Related methods

Removenote, page 358

## Example

```
note_id = dmAPIGet("create,c,dm_note");
if (note_id == NULL)
print dmAPIGet("getmessage,s0");

status = dmAPIExec("setfile,c,l,/home/sofia/.cshrc,text")
if (status != 1)
print dmAPIGet("getmessage,s0");

status = dmAPIExec("addnote,c,l,09000bb2800012a5")
if (status != 1)
print dmAPIGet("getmessage,s0");

status = dmAPIExec("save,c,l")
if (status != 1)
print dmAPIGet("getmessage,s0");
```

# Addpackage

## Purpose

Attaches a new package to the start activity of a workflow or to a work item.

## Syntax

To add a package to a start activity:

```
dmAPIGet("addpackage,session,object_id
,start_activity_name,input_port_name,package_name
,package_type[,note_id[,keep_permanent]]{,component_
ID}{,component_name}")
```

To add a package to a work item:

```
dmAPIGet("addpackage,session,object_id,package_name
,package_type{,component_ID}{,component_name}")
```

## Arguments

**Table 2-8. Addpackage method arguments**

| Argument | Description |
|---|---|
| *session* | Identifies an open repository session. |
| *object_id* | Identifies the workflow or work item to which you want to add the package. Use the object ID of the workflow or work item or an indirect reference (*@object_id*) that points to the object. |
| *start_activity_name* | Specifies the start activity to which to attach the package. |
| *input_port_name* | Identifies the input port. |
| *package_name* | Identifies the package. |
| *package_type* | The type of the components (the real type or a supertype) |
| *note_id* | Specifies the object ID of a note object to be delivered with the package. |
| *keep_permanent* | Set to TRUE to deliver the note object to all subsequent recipients of the package. The default is FALSE, which delivers the note only to the recipient of the start activity tasks. |

| Argument | Description |
|---|---|
| *component_id* | Object IDs of the components of the package. If you include a mirror object ID, the ID is de-referenced and the resulting object is bound to the package.<br><br>Separate multiple object IDs with commas. You must enclose the argument value in single quotes. |
| *component_name* | The object names of the components identified in *component_id*. The order of the names should correspond to the order of the object IDs.<br><br>Separate multiple object names with commas. You must enclose the argument value in single quotes. |

## Return value

The Addpackage method returns the package object ID if successful or NULL if unsuccessful.

## Usage notes

To use Addpackage to add a package to a start activity, you must be the workflow supervisor, the workflow creator, or a user with at least Sysadmin privileges. To use Addpackage to add a package to a work item, you must be the work item's performer, the workflow supervisor, or a user with at least Sysadmin privileges.

Use Addpackage to add a package to a start activity only if the activity has input ports and a defined precondition. Adding a package to an input port is like hand delivering a package to a port. If the same package definition appears in multiple input ports, you may have to add similar packages to each one of them, depending on how the pre-condition is set.

You can only add packages to an active workflow or to an acquired work item.

If you want to add a note to a package added to a work item, you must call the Addnote method after you call Addpackage.

### Specifying component IDs

You can specify multiple object IDs in the *component_id* argument. If you include multiple object IDs, the values must be separated by commas. You must enclose the argument in single quotes. The object IDs are recorded in the r_component_id attribute of the package object.

If you do not include one or more component object IDs, Content Server creates an empty package. When task performers receive an empty package, they must add components to the package before Content Server will allow them to complete the task. To add components to the package, the performer must remove the empty package and issue an Addpackage method that specified the components.

### Specifying component names

If you use the *component_name* argument, include the object names in the same order in which the components are specified by object ID in the *component_id* argument. Separate multiple names with commas. You must enclose the argument in single quotes.

The names are recorded in the package object's r_component_name attribute if package control is turned off at the repository and workflow levels. At the repository level, setting the r_component_name attribute is controlled by the wf_package_control_enabled attribute of the docbase config object. At the workflow level, the behavior is controlled by the package_control object in the process object (the workflow definition). These attributes interact to determine whether Content Server records object names, when specified in the component_name argument, or blanks in the package object's r_component_name attribute. Table 2–9, page 58, describes the interaction.

**Table 2-9. Interaction of attributes determining package control behavior**

| package_control (process setting) | wf_package_control_enabled (docbase config setting) | |
|---|---|---|
| | T (on) | F (off) |
| **0 (off)** | Content Server records blanks in 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 |

## Related methods

Removepackageinfo, page 362

## Example

The following example uses the Addpackage method to add a package to a running workflow.

```
buff = dmAPIGet("addpackage,s0," wflow_id ",act_01_0,port_i1,
pkg_00_a0,dm_document," component_id)
```

# Addpackageinfo

## Purpose

Adds a package definition to an existing port definition.

## Syntax

```
dmAPIExec("addpackageinfo,session,activity_id,port_name
,package_name,package_type[,package_id[,package_label]]
[,package_operation][,r_package_flag]")
```

## Arguments

**Table 2-10. Addpackage method arguments**

| Argument | Description |
|---|---|
| *session* | Identifies an open repository session. |
| *activity_id* | Identifies the activity. Use the object ID of the activity definition. |
| *port_name* | Identifies the port. |
| *package_name* | Identifies the package; must be unique for this port. |
| *package_type* | Identifies the type of objects expected to be bound. |
| *package_id* | Identifies the package. This can be a local object ID, replica object ID to a foreign object, or a foreign object ID. |
| *package_label* | Identifies the version of the component to add. Used for late binding. |
| *package_operation* | Identifies instructions for applications, such as a specific user inbox. |
| *r_package_flag* | Indicates whether the package is visible or invisible to the activity and whether the package may be empty. Valid values are:<br>0, meaning the package is invisible but cannot be empty<br>1, meaning the package is visible and cannot be empty<br>2, meaning the package is invisible but may be empty<br>3, meaning the package is visible and may be empty |

The default is 1.

## Return value

The Addpackageinfo method returns TRUE if successful and FALSE if unsuccessful.

## Usage notes

The Addpackageinfo method adds a package definition to an existing port definition. The new package information is appended to the end of the port and package list.

The port name and package name pair must be unique within the activity.

If a package definition already exists for the port you specify, the method returns an error.

If you specify a foreign object ID for the package_id, the server connects to the remote repository to verify the object.

The server does not process or validate the package_operation instructions.

### Package compatibility

Package definitions for linked input and output ports must be compatible. Package compatibility, page 218, in Content Server Fundamentals provides a full description of how compatibility is evaluated.

If you are defining a package for an output port, keep in mind that an output port is allowed to send packages whose components are subtypes of the object type identified in the package_type argument. Consequently, if the package type specified for an output port is a supertype of the object type specified for its linked input port, the server will generate a warning when you validate the process, to alert you that a runtime error may occur on that link.

For example, suppose an output port's package definition specifies dm_sysobject in r_package_type and its linked input port's package definition specifies a dm_folder object. The link will pass validation because folders are a subtype of SysObject. However, at runtime, the output port may send a document over the link. This will cause an error because the input port was expecting a folder, and documents and folders have a peer relationship in the object hierarchy rather than one of subtype and supertype.

### Setting the r_package_flag argument

This argument controls whether or not the package is visible to the activity performer and whether or not the package may be empty. Making a package invisible has the following behavioral consequences for the package:

- Webtop and DTC Documentum clients at version levels 5.3 and higher will not display the package to the activity performer.

- Removepackage and Addnote methods are not supported against an invisible package.
- The GET_INBOX administration method does not return objects associated with invisible packages.
- In the text in a task_subject, package references are only resolved if the package is visible. References to invisible packages are not resolved.
- In email templates, package references are only resolved and displayed if the package is visible. References to invisible packages are not resolved.

Workflow Manager does not recognize the visibility setting. Consequently, making a package invisible by setting the *r_package_flag* argument in Addpackageinfo has no effect if users are managing the workflows using Workflow Manager. Similarly, the setting is not recognized by pre-5.3 client applications.

An empty package is a package that has no components. Empty packages are generated by calling an Addpackage method at runtime with no components specified.

## Related methods

Removepackageinfo, page 362

## Example

This example uses the Addpackageinfo method to add package definitions to ports in an activity definition.

```
Print "Create dm_activity"
act_id1 = dmAPIGet ("create," sess ",dm_activity")
status = dmAPISet("set,s0," act_id1 ",object_name",
act_name1)
status = dmAPISet("set,s0," act_id1 ",trigger_threshold", 1)
status = dmAPISet("set,s0," act_id1 ",transition_type", 2)
status = dmAPISet("set,s0," act_id1 ",exec_type", 1)
status = dmAPISet("set,s0," act_id1 ",performer_type", 0)
status = dmAPISet("set,s0," act_id1 ",exec_method_id",
method_id)
status = dmAPISet("set,s0," act_id1 ",exec_save_results",
"F")
status = dmAPISet("set,s0," act_id1 ",exec_time_out", "1")
status = dmAPISet("set,s0," act_id1 ",exec_err_handling", 1)
status = dmAPIExec ("addport,s0," act_id1 ",port_i1,I")
status = dmAPIExec ("addpackageinfo,s0," act_id1
",port_i1,pkg_00_a1,dm_document")
status = dmAPIExec ("addport,s0," act_id1 ",port_o1,O")
status = dmAPIExec ("addpackageinfo,s0," act_id1
",port_o1,pkg_22_a1,dm_document")
status = dmAPIExec ("addport,s0," act_id1 ",port_o2,O")
status = dmAPIExec ("addpackageinfo,s0," act_id1
",port_o2,pkg_33_a1,dm_document," & doc_id2)
```

```
status = dmAPIExec ("addroutecase,s0," act_id1
",route2,dm_workflow.r_runtime_state = 1, port_o2")
status = dmAPIExec ("addroutecase,s0," act_id1
",EXCEPTIONAL,port_o1")
status = dmAPIExec ("save,s0," act_id1)
```

# Addport

## Purpose

Adds a new port to an activity definition.

## Syntax

```
dmAPIExec("addport,session,activity_id,port_name,port_type")
```

## Arguments

**Table 2-11. Addport method arguments**

| Argument | Description |
|---|---|
| *session* | Identifies an open repository session. |
| *activity_id* | Identifies the activity definition to which you want to add the port. Use the object ID of the activity. |
| *port_name* | Identifies the name of the port. The designer must create unique port names for each activity |
| *port_type* | Identifies the type of the port: I(input), O (output) or R (revert). |

## Return value

The Addport method returns TRUE on success and FALSE if unsuccessful.

## Usage notes

Use the Addport method to add a new port to an activity, and then use Addpackageinfo method to add package information to the new port.

The Addport method returns an error if the port name has already been used in this activity.

## Related methods

Addpackageinfo, page 60
Removeport, page 367

## Example

This example uses the Addport method to add ports to an activity definition.

```
Print "Create dm_activity"

act_id1 = dmAPIGet ("create," sess ",dm_activity")
status = dmAPISet("set,s0," act_id1 ",object_name",
act_name1)
status = dmAPISet("set,s0," act_id1 ",trigger_threshold", 1)
status = dmAPISet("set,s0," act_id1 ",transition_type", 2)
status = dmAPISet("set,s0," act_id1 ",exec_type", 1)
status = dmAPISet("set,s0," act_id1 ",performer_type", 0)
status = dmAPISet("set,s0," act_id1 ",exec_method_id",
method_id)
status = dmAPISet("set,s0," act_id1 ",exec_save_results",
"F")
status = dmAPISet("set,s0," act_id1 ",exec_time_out", "1")
status = dmAPISet("set,s0," act_id1 ",exec_err_handling", 1)
status = dmAPIExec ("addport,s0," act_id1 ",port_i1,I")
status = dmAPIExec ("addpackageinfo,s0," act_id1
",port_i1,pkg_00_a1,dm_document")
status = dmAPIExec ("addport,s0," act_id1 ",port_o1,O")
status = dmAPIExec ("addpackageinfo,s0," act_id1
",port_o1,pkg_22_a1,dm_document")
status = dmAPIExec ("addport,s0," act_id1 ",port_o2,O")
status = dmAPIExec ("addpackageinfo,s0," act_id1
",port_o2,pkg_33_a1,dm_document," & doc_id2)
status = dmAPIExec ("addroutecase,s0," act_id1
",route2,dm_workflow.r_runtime_state = 1, port_o2")
status = dmAPIExec ("addroutecase,s0," act_id1
",EXCEPTIONAL,port_o1")
status = dmAPIExec ("save,s0," act_id1)
```

# Addrendition

## Purpose

Adds a new rendition to an object.

## Syntax

```
dmAPIExec("addrendition,session,object_id,file_name,format
[,page_number][,storage_name][,atomic][,keep_flag]
[,page_modifier][,batch_flag][,other_file]")
```

## Arguments

**Table 2-12. Addrendition method arguments**

| Argument | Description |
|---|---|
| *session* | Identifies an open repository session. |
| *object_id* | Identifies the object to which you want to add the rendition. Use the object's ID or an indirect reference (*@object_id*) that points to the object. |
| *file_name* | Specifies the location of the file that contains the rendition you want to add. This can be either a relative or absolute path specification. |
| *format* | Defines the format of the file. This must be an unquoted character string that specifies one of the acceptable file formats. If the format is the same as the format of the content file identified in page_number, you must also include the *page_modifier* argument. |
| *page_number* | Identifies the page number of the content file to which you are attaching the rendition. (Refer to Usage notes for an explanation of page numbers.) The default is 0. |
| *storage_name* | Identifies the storage area where you want to store the new rendition. Use the name of the storage area's storage object. If this is not specified, the rendition is placed in the same storage area as the document's content. |

| Argument | Description |
|---|---|
| *atomic* | Commits any content changes to the object identified in *object_id* immediately. Set this to TRUE to save the changes automatically or FALSE if you want to require an explicit Save operation. The default is FALSE. |
| *keep_flag* | Indicates whether to keep or remove the rendition when the content with which the rendition is associated is updated or removed from the document or repository. TRUE means to keep the rendition. FALSE means to remove it. The default is FALSE. |
| *page_modifier* | Identifies the rendition when the rendition has the same format as the content page with which it is associated. You must set this argument if the rendition has the same format as the content file identified in *page_number*. The page modifier can be any string of no more than 32 characters.<br><br>You cannot include a page modifier if the *batch_flag* is set to T (TRUE). |
| *batch_flag* | Indicates whether the file identified in file_name is a tarred batch file containing several renditions to be processed. TRUE means the file is a batch file. FALSE means that the file is a single file.<br><br>If *batch_flag* is TRUE and *atomic* is TRUE, then batch processing occurs immediately. If *batch_flag* is TRUE but *atomic* is FALSE, the batch processing occurs when the associated document is saved.<br><br>The default for *batch_flag* is FALSE. Refer to Batch processing, page 68 in the Usage notes for information about using the batch processing capabilities of Addrendition. |
| *other_file* | Specifies the file that contains the resource fork for a Macintosh document. You can use either an absolute or a relative path. If this is set, the path must be valid or Addrendition fails.<br><br>The other_file value is ignored when used with external stores. |

## Return value

The Addrendition method returns TRUE if successful or FALSE if unsuccessful.

## Usage notes

Executing Addrendition creates a content object for the rendition. The content object binds the rendition to the document and content from which the rendition is derived. This is done by setting the parent_id, rendition, and page attributes in the content object.

The parent_id attribute is set to the object ID of the document that contains the source content file.

The rendition attribute is set to 1, 2, or 3. A value of 1 indicates that the rendition was generated by Content Server. A value of 2 indicates a rendition created by a client. A value of 3 indicates that the keep_flag was set to T (TRUE) in the Addrendition method. Setting the keep_flag to T means that when the object containing the source content is versioned, the rendition will be kept as a rendition of the content in the new version.

The page attribute records the page number of the content file from which the rendition is derived. Each primary content assigned to a document has a page number. This is an integer value that defines the content's position within the ordered list of contents associated with the document. The first content that you associate with a document has the page number zero. The page numbers of subsequent contents increment by one and must be in sequence.

If the content file already has a rendition with the same format and page modifier as the rendition you are adding, the new rendition replaces the current rendition.

Although no special permission is needed to execute the Addrendition method, to save the changes to the object, you must have either Version permission (to check in the object) or Write permission (to save the object). Setting the atomic argument to T requires only Version permission.

### Using the *atomic* argument

The atomic argument controls whether content changes are saved immediately, when the method executes, or later, when an explicit Save or Checkin is issued. If the argument is T, all content-related changes to the object are saved immediately. For example, if you issued a Setfile to add a new primary content and then used Addrendition with atomic set to T to add a rendition, both the Setfile and Addrendition changes are committed.

### Batch processing

Addrendition allows you to add multiple renditions to a single content page in one operation. The rendition content files plus a text file listing the rendition properties must be tarred into one file, which is then included as the *file_name* argument in the Addrendition method. The following procedure describes the steps in the process:

#### To add multiple renditions using one Addrendition method call:

1. Create a file called property.txt.

   This file lists the file name, page modifier, and parameters for each rendition you want to add. Refer to Property.txt format, page 69 for the format of this file.

2. Create a source text file.

   You can give this file any name.

3. Edit the source text file to include the name of the property.txt file and the names of the content files representing the renditions.

   Each file name must be on a separate line. Include the rendition file names in the same order in which their entries occur in the property.txt file. Do not include the actual file, only the file name. For example:

   property.txt
   rend_file_1.txt
   rend_file_2.txt
   rend_file_3.txt

   You can use full or relative path specifications for the files. If the files are in the same directory as the tar executable, no path specification is needed.

4. Tar the source text file.

   You can use %DM_HOME%\bin\dmtar.exe on Windows or $DM_HOME/bin/tar on UNIX platforms.

   The tarred file must have a .tar extension.

5. Execute the Addrendition method including the tarred file resulting from Step 4 as the *file_name* argument.

   If the Addrendition is issued with the atomic flag set to T, the file is untarred and processed and changes saved to the repository immediately. If the atomic flag is F, the file is untarred and processed when the document is saved to the repository.

### Property.txt format

Each rendition that you add in the batch must have a three-line entry in the property.txt file in the following format:

```
file_name='name'
page_modifier='modifier'
parameters='parameter="value"{,parameter="value"}'
```

*name* is the name of the content file for the rendition. *modifier* is the page modifier for the rendition. If there is no modifier, specify an empty string. The page modifier dm_batch is reserved for use by Content Server.

The parameter values you can set are the same as those allowed in the SET_CONTENT_ATTRS administration method. (Refer to SET_CONTENT_ATTRS, page 307 of the *Content Server DQL Reference Manual* for details.)

Each line in the entries must begin in the first column of the page; no spaces are allowed before or after the equal signs; and the entries for individual renditions must be separated by one blank line.

For example, this excerpt from a property.txt file describes three renditions:

```
file_name='rend_file_1.txt'
page_modifier='first'
parameters='title="Rend_1",width=FLOAT(20),date=DATE(7/4/02)'

file_name='rend_file_2.txt'
page_modifier='second'
parameters='title="Rend_2",width=FLOAT(40),date=DATE(7/4/02)'

file_name='rend_file_3.txt'
page_modifier='third'
parameters='title="Rend_3",width=FLOAT(60),date=DATE(7/4/02)'
```

## Related methods

Removerendition, page 368

## Example

This example performs the following actions:

- Creates a document object
- Associates content of a particular format
- Associates a rendition of the content in a different format
- Saves the document object

```
doc_id = dmAPIGet("create," session ",dm_document")
if (doc_id == NULL) {
err_msg = dmAPIGet( "getmessage," session )
print "Error creating a document object"
print err_msg
exit
}

txt_file = "myfile.txt"
err_flag = dmAPIExec("setfile," session "," doc_id "," txt_file ",text")
if (err_flag == 0) {
err_msg = dmAPIGet("getmessage," session)
print "Error setting file contents"
print err_msg
exit
}

wp5_file = "myfile.wp5"
err_flag = dmAPIExec("addrendition," session ","
doc_id "," wp5_file ",wp5")
if (err_flag == 0) {
err_msg = dmAPIGet("getmessage," session)
print "Error adding a rendition of file contents"
print err_msg
exit
}

if (dmAPIExec("save," session "," doc_id) ==0) {
```

```
print "Error saving the document object'" doc_id"'"
print dmAPIGet("getmessage, " session)
}
```

# Addroutecase

## Purpose

Adds a route case to an activity definition.

## Syntax

```
dmAPIExec("addroutecase,session,activity_id,
route_case_identifier,routing_condition{,output_port}")

dmAPIExec("addroutecase,session,activity_id,
route_case_identifier[,,{output_port}]")

dmAPIExec("addroutecase,session,activity_id,EXCEPTIONAL
{,output_port}")
```

## Arguments

**Table 2-13. Addroutecase method arguments**

| Argument | Description |
|---|---|
| *session* | Identifies an open repository session. |
| *activity_id* | Identifies the activity definition to which you want to add a route case. Use the object ID of the activity. |
| *route_case_identifier* | Names the route case. The designer must create unique route case names. |
| *routing_condition* | The condition expression. |
| EXCEPTIONAL | A reserved word that creates an exceptional route case. This word is not case sensitive. |
| *output_port* | Identifies the port or ports associated with this route case. Use the ports' names. |

## Return value

The Addroutecase method returns TRUE if successful and FALSE if unsuccessful.

## Usage notes

The Addroutecase method appends the route case to the condition list. The first form adds a regular route case and the second form changes the ports of an existing route case. The third form adds an exceptional route case to an automatic transition condition.

Regular route cases are evaluated in the order in which they appear in the list, and the exceptional route case is always evaluated last.

You must assign a unique identifier to each regular route case. Identifiers that are longer than sixteen characters are automatically truncated.

### Specifying the routing condition

The routing condition can be any valid expression. (Refer to The WHERE clause, page 142 of the *Content Server DQL Reference Manual* for information about valid expressions.) If the expression references an attribute of the workflow, work item, or package object, you can use the tokens dm_workflow, dmi_workitem, or dmi_package to reference the object. At runtime, the server substitutes the correct object for the token.

For example, if you want the server to examine the value of the work item's return_value attribute, you could define the routing condition as:

```
dmi_workitem.return_value=0
```

At runtime, the server interprets the dmi_workitem to mean the work item referenced by the r_last_witem_id attribute of the workflow object.

If you use dmi_package, the server chooses the first component in the package (identified in r_package_name[0]) if there are multiple components in the package.

Additionally, when you use the package token, the condition may only reference attribute defined for the object type specified in the package's definition. For example, if the package definition specifies dm_sysobject and the object type of actual component in the package is my_subtype, only the attributes defined for dm_sysobject can be referenced in the routing condition.

## Related methods

Removeroutecase, page 371

## Example

This example uses the Addroutecase method to add routing information to an activity definition.

```
Print "Create dm_activity"

act_id1 = dmAPIGet ("create," sess ",dm_activity")
```

```
status = dmAPISet("set,s0," act_id1 ",object_name",
act_name1)
status = dmAPISet("set,s0," act_id1 ",trigger_threshold", 1)
status = dmAPISet("set,s0," act_id1 ",transition_type", 2)
status = dmAPISet("set,s0," act_id1 ",exec_type", 1)
status = dmAPISet("set,s0," act_id1 ",performer_type", 0)
status = dmAPISet("set,s0," act_id1 ",exec_method_id",
method_id)
status = dmAPISet("set,s0," act_id1 ",exec_save_results",
"F")
status = dmAPISet("set,s0," act_id1 ",exec_time_out", "1")
status = dmAPISet("set,s0," act_id1 ",exec_err_handling", 1)
status = dmAPIExec ("addport,s0," act_id1 ",port_i1,I")
status = dmAPIExec ("addpackageinfo,s0," act_id1
",port_i1,pkg_00_a1,dm_document")
status = dmAPIExec ("addport,s0," act_id1 ",port_o1,O")
status = dmAPIExec ("addpackageinfo,s0," act_id1
",port_o1,pkg_22_a1,dm_document")
status = dmAPIExec ("addport,s0," act_id1 ",port_o2,O")
status = dmAPIExec ("addpackageinfo,s0," act_id1
",port_o2,pkg_33_a1,dm_document," & doc_id2)
status = dmAPIExec ("addroutecase,s0," act_id1
",route2,dm_workflow.r_runtime_state = 1, port_o2")
status = dmAPIExec ("addroutecase,s0," act_id1
",EXCEPTIONAL,port_o1")
status = dmAPIExec ("save,s0," act_id1)
```

# Anyevents

## Purpose

Determines if there are any events waiting for the user.

## Syntax

```
dmAPIExec("anyevents,session")
```

## Arguments

**Table 2-14. Anyevent method arguments**

| Argument | Description |
|---|---|
| *session* | Identifies an open repository session. |

## Return value

The Anyevents method returns TRUE if there are events pending or FALSE if there are no events waiting.

## Usage notes

The Anyevents method looks at an internal flag to see if any items have been placed on the user's queue since the last time the Getevents method was called to retrieve inbox items for the user. This flag is set whenever an item is queued to the user and unset each time the user executes a Getevents method.

## Related methods

Getevents, page 239

## Example

This example demonstrates the use of Anyevents in a polling routine:

```
while (1)
{sleep (10);
```

```
if (dmAPIExec("anyevents," session))
  {coll = dmAPIGet ("getevents," session);
  while (dmAPIExec("next," session "," coll);
  {print"EVENT:" dmAPIGet("get,"session","coll",message")
   }
  }
 }
```

# Append

## Purpose

Adds a value to a repeating attribute.

## Syntax

```
dmAPISet("append,session,object_id,attribute
[,pattern]","value")
```

## Arguments

**Table 2-15. Append method arguments**

| Argument | Description |
|---|---|
| *session* | Identifies an open repository session. |
| *object_id* | Identifies the object that contains the attribute to which you want to add a value. Use the object's ID or an indirect reference (*@object_id*) that points to the object.<br><br>To specify the api config object, use the keyword apiconfig instead of the object ID. |
| *attribute* | Identifies the attribute to which you are adding a value. You must specify a repeating attribute that belongs to the specified object. |
| *pattern* | Valid only if the attribute is a date attribute, this argument defines a date format for the specified value. The pattern can be any valid input format for dates. Refer to the Usage notes, page 78 for an example and to Date literals, page 15 of the *Content Server DQL Reference Manual* for a list of all valid input formats. |
| *value* | Specifies the value to append. You can use either a variable or a character string to specify the value. |

## Return value

The Append method returns TRUE if successful or FALSE if unsuccessful.

## Usage notes

The Append method adds the value at the end of the list of values in the repeating attribute.

If the attribute contains date values, you may want to specify an input format for the value. This ensures that a specified value is not misinterpreted by the server. For example, 03/04/1995 could be interpreted as March 4, 1995 (*mm/dd/yyyy*) or as April 3, 1995 (*dd/mm/yyyy*). To ensure that the value is interpreted as you wish, specify the format's pattern in the Append method. For example:

```
dmAPISet("append,c,09000002648921e5,review_date,dd/mm/yyyy",
"08/03/1996")
```

## Related methods

Insert, page 271
Remove, page 351
Set, page 416

**Note:** You can also use the DQL UPDATE...OBJECT statement to add values to repeating attributes.

## Example

The following example performs the following operations:

- Creates a document object
- Associates content of the desired file
- Appends the authors' names
- Saves the document object

```
doc_id = dmAPIGet("create," session ",dm_document")
if (doc_id == NULL) {
err_msg = dmAPIGet("getmessage," session)
print "Error creating a document object"
print err_msg
exit
}

the_file = "chap_1.wp5"
err_flag = dmAPIExec("setfile," session "," doc_id "," the_file ",wp5)
if (err_flag == 0) {
err_msg = dmAPIGet("getmessage," session)
print "Error setting file contents"
print err_msg
exit
}

author_list = "W. Smith,J. Jones,R. Abnous,S. Ruppenthal"
count = split(author_list, author, ",")
```

```
for (i=1; i<=count; i++) {
err_flag = dmAPISet("append,"session","doc_id",authors",author[i])
if (err_flag == 0) {
err_msg = dmAPIGet("getmessage," session)
print "Error appending authors' name to the 'authors' attribute"
print err_msg
exit
}
}

err_flag = dmAPIExec("save," session "," doc_id)
if (err_flag == 0) {
err_msg = dmAPIGet("getmessage," session)
print "Error saving the document object"
print err_msg
exit
}
```

# Appendcontent

## Purpose

Appends a content file to an object.

## Syntax

```
dmAPISet("appendcontent,session,object_id[,length][,set_resource]",
"content")
```

## Arguments

**Table 2-16. Appendcontent method arguments**

| Argument | Description |
|---|---|
| *session* | Identifies an open repository session. |
| *object_id* | Identifies the object to which you are appending the content. Use the object's ID or an indirect reference (*@object_id*) that points to the object. |
| *length* | Length, in bytes, of the data to be appended as content. This is only required if the content is binary data. |
| *set_resource* | For files created on Macintosh systems, indicates whether you want the method to append the data fork (the file containing the content) or the resource fork. FALSE (the default) appends the data fork (the content file). TRUE appends the resource fork (the file specified in the other_ticket attribute of the content object). |
| *content* | The location in memory of the data to be appended. |

## Return value

The Appendcontent method returns TRUE if successful or FALSE if unsuccessful.

## Usage notes

Use Appendcontent in an application when you want to append data that resides in working memory. This method is not intended for direct use by a user. Instead, in an

application, the method should interact with the user's system, to take data residing in working memory and append it as content to a document. (Users wanting to add content to an object should use the Setfile, Appendfile, or Insertfile method.)

If you are appending content created on a Macintosh machine, you must issue the Appendcontent twice before saving the object. Issue the method once to append the data fork (set_resource=F) and then again to append the resource fork (set_resource=T). When you append the data fork, the content argument must identify the location in memory of the data fork. When you append the resource fork, the content argument must identify the memory location of the resource fork.

You cannot use the Appendcontent method with external storage areas.

## Related methods

Insertcontent, page 274
Removecontent, page 355
Setcontent, page 424

## Example

```
doc_id = dmget("create,s0,dm_document");
if (doc_id == NULL)
{ err_mess = dmget("getmessage,s0")
  print err_mess
  exit
 }
 status = dmset("set,s0,last,object_name","my_document")
 status = dmset("set,s0,last,a_content_type", "text")

content = "This manual is for use with the B1.1release."

 status = dmset("appendcontent,s0,last", content)
 if (status != 1)
 { err_mess = dmget("getmessage,s0")
   print err_mess
   exit
  }

 status = dmexec("save,s0,last")
 if (status != 1)
 { err_mess = dmget("getmessage,s0")
   print err_mess
   exit
 }
```

# Appendfile

## Purpose

Appends a content file to an object.

## Syntax

```
dmAPIExec("appendfile,session,object_id,file_name[,other_file]")
```

## Arguments

**Table 2-17. Appendfile method arguments**

| Argument | Description |
|---|---|
| *session* | Identifies an open repository session. |
| *object_id* | Identifies the document to which you are adding the content. Use the object's ID or an indirect reference (*@object_id*) that points to the object. |
| *file_name* | Specifies the file that you are adding as content to the object. Use either a relative or absolute path specification. |
| *other_file* | Specifies the file that contains the resource fork for a Macintosh document. You can use either an absolute or a relative path. If this is set, the path must be valid or the method fails.<br><br>The other_file value is ignored when used with external stores. |

## Return value

The Appendfile method returns TRUE if successful or FALSE if not successful.

## Usage notes

You must have Version permission for the document to add a new content to a document.

Note that if the content type of the object identified by *object_id* has not been set, you must do so before you execute the Appendfile method. You set the object's content type by setting its a_content_type attribute.

## Related methods

Insertfile, page 277
Setfile, page 434

## Example

This example performs the following operations:

- Creates a document object
- Appends files to the document object
- Saves the document object

```
doc_id = dmAPIGet("create," session ",dm_document")
if (doc_id == NULL) {
err_msg = dmAPIGet("getmessage," session)
print "Error creating a document object"
print err_msg
exit
}

cmd_str =
err_flag=dmAPISet("set," session "," doc_id ",a_content_type", "wp5")
if (err_flag == 0) {
err_msg = dmAPIGet("getmessage," session)
print "Error setting content type"
print err_msg
exit
}

file_list = "section1.wp5 section2.wp5 section3.wp5 section4.wp5"
count = split(file_list, file, ",")
for (i=1; i=count; i++) {
err_flag = dmAPIExec("appendfile," session "," doc_id "," file[i])
if (err_flag == 0) {
err_msg = dmAPIGet("getmessage," session)
print "Error appending content to the document object"
print err_msg
exit
}
}

err_flag = dmAPIExec("save," session "," doc_id)
if (err_flag == 0) {
err_msg = dmAPIGet("getmessage," session)
print "Error saving the document object"
print err_msg
exit
}
```

# Appendpart

## Purpose

Appends a component to a virtual document.

## Syntax

```
dmAPIGet("appendpart,session,document_id,component_id
[,version_label][,use_node_ver_label][,follow_assembly]
[,copy_child][,containment_type,containment_desc]")
```

## Arguments

**Table 2-18. Appendpart method arguments**

| Argument | Description |
|---|---|
| *session* | Identifies an open repository session. |
| *document_id* | Identifies the virtual document to which you are appending the new component. Use the object's ID or an indirect reference (*@object_id*) that points to the object. |
| *component_id* | Identifies the component you want to add to the virtual document. Use either the component's object ID or its chronicle ID (the value in the component's i_chronicle_id attribute).<br><br>A component can be any SysObject or SysObject subtype except a folder, cabinet, or folder or cabinet subtype. |
| *version_label* | Identifies which version of the new component you want to append. You can use either an implicit or symbolic label. (Refer to Identifying the new component, page 86 for more information about using version labels.) |

| Argument | Description |
|---|---|
| *use_node_ver_label* | Determines how the server chooses late-bound descendants of a component. This is useful only if the component is a virtual document.<br><br>T (TRUE) directs the server to use the component's early-bound symbolic label to resolve late-bound descendants. F (FALSE) directs the server to resolve the component's descendants using the late-bound version label specified in the IN DOCUMENT clause. (Refer to The use_node_ver_label argument, page 87 for an expanded explanation.)<br><br>The default is F (FALSE). |
| *follow_assembly* | Determines whether the server uses the containment objects or the component's assembly to find the component's descendants. This argument is only useful if the component is a virtual document.<br><br>T (TRUE) directs the server to use the component's assembly (if it has an assembly). F (FALSE) directs the server to use the containment objects. |
| *copy_child* | Determines whether the Documentum client copies or references the component when a user copies the containing virtual document. This is an integer attribute. Valid values are:<br><br>0, which lets users decide at the time the copy operation is requested<br><br>1, which directs the client to reference the component<br><br>2, which directs the client to copy the component<br><br>The default value is 0.<br><br>Refer to Defining copy behavior, page 88 for more information. |

| Argument | Description |
|---|---|
| *containment_type* | Used only if the containing virtual document (identified in *document_id*) is an XML document. This argument identifies what type of reference to the component is found in the containing document.<br><br>If you include this, you must also include *containment_desc*.<br><br>This argument is used primarily by the Documentum clients. Refer to XML support, page 88 for details. |
| *containment_desc* | Used only if the containing virtual document (identified in *document_id*) is an XML document. This argument identifies the text of reference to the component in the containing document.<br><br>If you include this, you must also include *containment_type*.<br><br>This argument is used primarily by the Documentum clients. Refer to XML support, page 88 for details. |

## Return value

The Appendpart method returns the object ID of the containment object that describes the relationship between the component and the virtual document.

## Usage notes

The Appendpart method appends a new component to the end of the ordered list of components in a virtual document.

**Note:** If you use user-defined order numbers, you cannot use the Appendpart method because this method has no facility for handling user-defined order numbers. You must use the Insertpart method to add all new components to the virtual document.

You must save (or check in) the virtual document after you append a new component to save the new component as part of the virtual document. Saving the virtual document requires Write permission on the document. Checking it in as a new version requires Version permission on the document.

### Identifying the new component

Identify the component you want to append by using either its object ID or the value in its chronicle ID attribute and, optionally, a version label. The object ID is found in the object's r_object_id attribute. The chronicle ID, which identifies the root (original) version

of the object, is found in the i_chronicle_id attribute. (If an object has no versions, then its r_object_id and i_chronicle_id attributes have the same value.)

#### Early binding

If you specify a version label, the server searches the version tree associated with the object specified in *component_id*, finds that version, and appends it to the virtual document. This is called early binding—a specific version of the object is bound to the virtual document when the object is added.

You can specify either an implicit or symbolic version label. If you specify an implicit label, there is an absolute link between that version and the virtual document. If you specify a symbolic label, then whatever version of the object carries the label is linked to your virtual document. (This means that the version that appears in your document may change if the symbolic label is moved from one version to another. Refer to Absolute links, page 166 and Symbolic links, page 166 of *Content Server Fundamentals*for a more detailed discussion of absolute and symbolic linking.)

#### Late binding

If you specify only the component's object ID or its chronicle ID (with no version label), the server associates the component's entire version tree with the virtual document. Later, when you assemble the virtual document, you must identify which version of the object you want to include. This is called late binding. Late binding allows you to assemble virtual documents conditionally.

### Defining assembly behavior

If the new component is a virtual document, the *use_node_ver_label* and *follow_assembly* arguments let you define how the server chooses the component's descendants when the virtual document that contains the component is assembled.

#### The use_node_ver_label argument

The *use_node_ver_label* argument sets the use_node_ver_label attribute in the containment object for the component.

If use_node_ver_label is TRUE for the component and the component is early bound to the virtual document, the server resolves any late-bound descendants of the component using the early-binding label specified for the component. It ignores any binding condition specified at the time of assembly. If an early-bound descendant is found that has use_node_ver_label set to TRUE, then that descendant's label is used to resolve descendants from that point in the hierarchy.

If use_node_ver_label is FALSE or if the component is late bound to the virtual document, the server resolves the component's late-bound descendants using the binding condition specified at the time of assembly.

(use_node_ver_label, page 167 in *Content Server Fundamentals* contains an expanded explanation of how use_node_ver_label affects the assembly of virtual documents.)

### The follow_assembly argument

The *follow_assembly* argument sets the follow_assembly attribute in the component's containment object. If the component has an associated assembly and follow_assembly is set to TRUE, the server will use the assembly rather than the containment objects to determine the component's descendants. In these cases, any binding conditions for the descendants are ignored. The descendants are taken from the assembly.

If follow_assembly is FALSE or if the component has no assembly, the server uses the containment objects and binding specifications to determine the component's descendants.

## Defining copy behavior

Setting the *copy_child* argument sets the copy_child attribute in the component's containment object. This attribute controls how the Documentum client handles the component when users copy the containing virtual document. The client can either copy or reference the component.

Copy_child is an integer attribute with three valid values:

- 0, which means that users make the decision to copy or reference the component when they request the operation
- 1, which directs the client to reference the component in the new copy
- 2, which directs the client to copy the component for the new copy

The default value is 0.

**Note:** The reference is an internal pointer. It is not a dm_reference object.

## XML support

The *containment_type* and *containment_desc* arguments are used internally by the DFC to correctly set the references in the containing document when the component is added to the containing document. The arguments set the a_contain_type and a_contain_desc attributes in the containment object.

Although you can use these arguments to set the attributes to values you choose, if you do, the behavior of the XML document when manipulated using DesktopClient or the DFC is undefined.

For more information about the values set by these arguments and their use, refer to XML support, page 165 in *Content Server Fundamentals*.

# Related methods

Insertpart, page 280
Removepart, page 364
Updatepart, page 486

## Example

```
component_id = dmget("create,s0,dm_document");
  if (component_id == NULL)
  {  err_mess = dmget("getmessage,s0")
     print err_mess
     exit
  }
  status = dmset("set,s0,last,object_name", "Chapter1")
  status = dmexec("save,s0,last")
  if (status != 1)
  {  err_mess = dmget("getmessage,s0")
     print err_mess
     exit
  }

  parent_id = dmget("create,s0,dm_document");
  if (parent_id == NULL)
  {  err_mess = dmget("getmessage,s0")
     print err_mess
     exit
  }

 status = dmset("set,s0,last,object_name", "Book")

 containment_id = dmget("appendpart,s0,last,"
component_id ",,T,F,2")
  if (containment_id == NULL)
  {  err_mess = dmget("getmessage,s0")
     print err_mess
     exit
  }

  status = dmexec("save,s0,last")
  if (status != 1)
  {  err_mess = dmget("getmessage,s0")
     print err_mess
     exit
  }
```

# Appendstate

## Purpose

Appends a state to a lifecycle.

## Syntax

```
dmAPIGet("appendstate,session,object_id")
```

## Arguments

**Table 2-19. Appendstate method arguments**

| Argument | Description |
|---|---|
| *session* | Identifies an open repository session. |
| *object_id* | The object ID of the lifecycle. |

## Return value

The Appendstate method returns the new state's index value. The index value identifies which row in each of the repeating attributes that define states is associated with the new state.

## Usage notes

The Appendstate method is used to install a lifecycle definition. Users must have Write permission of the object to add a new state.

The Appendstate method also sets the following default attribute values for the state:

**Table 2-20. Default attribute values set by appendstate**

| Attribute | Default |
|---|---|
| state_name | null (user must provide a value) |
| state_class | 0 (normal) |
| entry_criteria_id | null |
| user_criteria_id | null |

| Attribute | Default |
|---|---|
| action_object_id | null |
| user_action_id | null |
| exception_state | null |
| allow_attach | F (user is required to ensure at least one is set to TRUE) |
| allow_schedule | T |
| return_to_base | F |
| type_override | null |
| allow_demote | F |

## Related methods

Removestate, page 373

## Example

```
status = dmAPIGet("appendstate," sess ",l")a
```

# Apply

## Purpose

Invokes procedures that perform system administration tasks and run external procedures.

## Syntax

```
dmAPIGet("apply,session,object_id,admin_method_name
{,argument,datatype,value}")
```

## Arguments

**Table 2-21. Apply method arguments**

| Argument | Description |
|---|---|
| *session* | Identifies an open repository session. |
| *object_id* | Identifies the object that is the subject of the operation you are performing. Use the object's object ID. You can specify this as NULL in some instances. (Refer to Chapter 3, Administration Methods , of the *Content Server DQL Reference Manual* for a description of the syntax for individual methods.)<br><br>**Important Note:** You cannot use an indirect reference for this argument when executing the Apply method. All Apply operations must be performed locally. |
| *admin_method_ name* | Specifies the operation that you want to perform. Table 2–22, page 93 lists the methods and their purposes. Refer to Chapter 3, Administration Methods , in the *Content Server DQL Reference Manual* for detailed descriptions of each method. |
| *argument* | Specifies a valid argument for the specified method. |
| *datatype* | Identifies the datatype of the specified argument. The datatype is specified as a single letter. Valid datatypes are:<br><br>S or s (for String) B or b (for Boolean) I or i (for Integer) T or t (for Time) D or d (for Double) |
| *value* | Defines that value assigned to the argument. This is specified as a string containing an appropriate value for the specified datatype. |

## Return value

The Apply method returns a collection identifier for a collection that contains the results of the specified method.

## Usage notes

Table 2–22, page 93 lists the administration methods you can invoke using the Apply method and briefly describes the operation that each performs.

**Note:** The links in the second column take you to the description of the method in the *DQL Reference Manual*.

**Table 2-22. Administration methods by category for the Apply method**

| Category of operation | Function | Description |
|---|---|---|
| Process Management | BATCH_PROMOTE, page 175 | Promotes multiple objects to their next lifecyle states.<br><br>**Note:** This method is not available through Documentum Administrator or using DQL EXECUTE. |
| | CHECK_SECURITY, page 186 | Checks a user or group's permissions level for one or more objects. |
| | GET_INBOX, page 219 | Returns items in user's Inbox. |
| | MARK_AS_ ARCHIVED, page 256 | Sets the i_is_archived attribute of a dm_audittrail, dm_audittrail_acl, or dm_audittrail_group to T. |
| | PURGE_AUDIT, page 275 | Removes audit trail entries from the repository. |
| | RECOVER_AUTO_ TASKS, page 288 | Recovers work items claimed by a worflow agent master session but not yet processed. |
| | ROLES_FOR_USER, page 302 | Returns the roles assigned to a user in a particular client domain. |
| Execute methods | DO_METHOD, page 197 | Executes system-defined procedures such as lpq or who or user-defined procedures. |
| | HTTP_POST, page 228 | Directs the execution of a method to an application server. |

| Category of operation | Function | Description |
|---|---|---|
| Content storage management | CAN_FETCH, page 177 | Determines whether content in a distributed storage area component can be fetched by the server. |
| | CHECK_ RETENTION_ EXPIRED, page 182 | Finds SysObjects in content-addressed storage that have an expired retention period or no retention period. |
| | CLEAN_LINKS, page 189 | Provides maintenance for linked store storage areas. On UNIX platforms, it cleans up unneeded linkrecord objects, directories, and links associated with linked storage areas. On Windows platforms, it cleans up unneeded linkrecord objects and resets file storage object security. |
| | DELETE_REPLICA , page 193 | Removes a replica from a distributed storage area. |
| | DESTROY_ CONTENT, page 195 | Removes a content object and its associated file from the repository. (Do not use this for archiving; use PURGE_CONTENT instead.) |
| | GET_FILE_URL, page 217 | Returns the URL to a content file. |
| | GET_PATH, page 225 | Returns the path to a particular content file in a particular distributed storage area component. |
| | IMPORT_REPLICA, page 234 | Imports an external file as a replica of content already in the repository. |
| | MIGRATE_ CONTENT, page 260 | Moves content files from one storage area to another. |
| | PURGE_CONTENT, page 283 | Deletes a content file from a storage area. (Used as part of the archiving process.) |
| | PUSH_CONTENT_ ATTRS, page 285 | Sets the content metadata in a content-addressed storage system for a document stored in that storage system. |

| Category of operation | Function | Description |
|---|---|---|
| | REGISTER_ASSET, page 290 | Queues a request for the creation of a thumbnail, proxies, and metadata for a rich media content file. The request is queued to Media Server.<br><br>This method is only available or useful if you have Documentum Media Transformation Services running. |
| | REPLICATE, page 295 | Copies content in one component of a distributed storage area to another area. |
| | RESTORE_CONTENT, page 300 | Moves a file or files from archived storage to the original storage location. |
| | SET_CONTENT_ ATTRS, page 307 | Sets the content_attr_name and content_attr_value attributes in the content object associated with the content file. |
| | SET_STORAGE_ STATE, page 316 | Sets the state of a storage area to off-line, on-line, or read-only. |
| | SYNC_REPLICA_ RECORDS, page 321 | Synchronizes the replica record objects with the current definition of a distributed storage area.<br><br>**Note:** This method is not available through Documentum Administrator. |
| | TRANSCODE_ CONTENT, page 325 | Queues a request for a content transformation to Media Server.<br><br>This method is only available or useful if you have Documentum Media Transformation Services running. |
| Database Methods | DB_STATS, page 191 | Provides database operation statistics for a session. |
| | DROP_INDEX, page 206 | Drops an index. |
| | EXEC_SQL, page 211 | Executes SQL statements. |
| | EXPORT_TICKET_ KEY, page 213 | Exports the login ticket key. |

| Category of operation | Function | Description |
|---|---|---|
| | FINISH_INDEX_MOVES, page 215 | Completes an interrupted object type index move operation. |
| | IMPORT_TICKET_KEY, page 236 | Imports a login ticket key. |
| | MAKE_INDEX, page 253 | Creates an object type index. |
| | MOVE_INDEX, page 271 | Moves an object type index from one tablespace to another.<br>**Note:** Not supported on DB2. |
| | REORGANIZE_TABLE, page 292 | Reorganizes a database table for query performance. |
| | RESET_TICKET_KEY, page 298 | Resets the login ticket key |
| | UPDATE_STATISTICS, page 329 | Updates the statistics for a database table. |
| Full-Text Methods | ESTIMATE_SEARCH, page 208 | Returns the number of results matching a particular SEARCH condition. |
| | MARK_FOR_RETRY, page 258 | Finds all queue items representing objects that failed indexing and marks them as pending indexing. |
| | MODIFY_TRACE, page 269 | Sets the tracing level for full-text indexing operations. |
| Session Management | CHECK_CACHE_CONFIG, page 179 | Requests a consistency check on a particular cache config object. |
| | GET_LAST_SQL, page 223 | Returns the last SQL statement issued. |
| | GET_SESSION_DD_LOCALE, page 227 | Returns the locale in use for the current session. |
| | LIST_AUTH_PLUGINS, page 238 | Lists the authentication plugins loaded by Content Server. |
| | LIST_RESOURCES, page 240 | Provides information about the server operating system environment. |
| | LIST_SESSIONS, page 244 | Provides information about current, active sessions. |

| Category of operation | Function | Description |
|---|---|---|
| | LIST_TARGETS, page 248 | Lists the connection brokers defined as targets for the server.<br><br>The information is returned in a collection with one result object whose attributes list the connection brokers defined as targets for the server. |
| | LOG_ON, page 251 and LOG_OFF, page 250 | Turn server logging of information about RPC calls on or off. |
| | PING, page 273 | Determine if a client has an active server connection. |
| | SET_APIDEADLOCK, page 304 | Sets a deadlock trigger on a particular API method or operation. |
| | SET_OPTIONS, page 312 | Turn various tracing options on or off. |
| | SHOW_SESSIONS, page 319 | Provides information about current, active sessions and a user-specified number of timed-out sessions. |
| Web Publishing Management | WEBCACHE_ PUBLISH, page 332 | Invokes the dm_webcache_publish method to publish documents to a Web site. |

## Related methods

There are no related methods. However, you can also use the DQL EXECUTE statement to execute the Apply functions (with the exception of PING and WEBCACHE_PUBLISH).

## Examples

Refer to Chapter 3, Administration Methods, in the *Content Server DQL Reference Manual* for examples.

# Archive

## Purpose

Queues an archive request to the repository operator.

## Syntax

```
dmAPIGet("archive,session,predicate[,operator][,priority]
[,send_mail][,due_date]")
```

## Argument descriptions

**Table 2-23. Archive method arguments**

| Argument | Description |
|---|---|
| *session* | Identifies an open repository session. |
| *predicate* | Identifies the object or objects you want to archive. A valid predicate has the format:<br>`type_name WHERE qualification`<br>The *type_name* must be a SysObject subtype, and the *qualification* must be a valid WHERE clause qualification.<br>Do not specify the (ALL) keyword with *type_name*. The dmarchive utility called to execute the archive request automatically adds (ALL) to the type name in the predicate. |
| *operator* | Specifies who receives the archive request. Specify the repository user name of the operator. If you do not include the operator name, the request is forwarded to the inbox of the default operator. Refer to the Usage notes for more details. |
| *priority* | Specifies a priority level for the request. The server operator does not use this value. It is solely for the convenience of end users or user applications. The default value is zero. |
| *send_mail* | Directs the server to send an electronic mail notification about the archive request to the specified operator. The default is FALSE. |
| *due_date* | Informs the operator when the archive request should be completed. The default for this is NULLDATE. |

## Return value

The Archive method returns the object ID of the dmi_queue_item for the queued archive request.

Because the query defined by the predicate can return more than one object, the ID returned by the Archive method is the ID of the last object queued for archiving. To find all the objects queued for archiving, run a query against the dm_queue view.

## Usage notes

Executing the Archive method runs the query specified in the *predicate* to ensure that the specified objects exist, are online, and are SysObject subtypes. If these conditions are met, the method queues a DM_ARCHIVE event to the inbox of the repository operator, sending the predicate as the event message.

**Note:** Do not include the keyword (ALL) in the type name in the predicate. The dmarchive utility, called to process the archive requests, adds (ALL) to the type name automatically. Consequently, if you include the keyword in the predicate, the utility fails when it attempts to execute the query generated from the predicate because (ALL) appears twice in the syntax.

The operator is responsible for actually archiving the requested files or files. Content Server provides an archiving utility, called dmarchive, for this purpose. When the operator receives the archive request, dmarchive can be executed in response, to move the requested from file from its storage area to a designated archive area.

If you do not specify an operator, the event is queued to the default operator, who is the user specified in the operator_name attribute of the server's server config object.

Refer to Archiving and restoring documents, page 280 in the *Content Server Administrator's Guide* for more information about archiving and restoring content files.

## Related methods

Restore, page 389

## Example

```
event_id = dmAPIGet("archive,c,dm_sysobject where\
owner_name = 'user',sysadmin,1,T,11/01/94");
if (sevent_id == NULL)
{error_msg = dmAPIGet("getmessage,s0");
print error_msg;
}
```

# Assemble

## Purpose

Selects the components for an assembly.

## Syntax

```
dmAPIExec("assemble,session,document_id[,virtual_doc_id]
[,interrupt_freq][,qualification][,nodesort_list]")
```

## Argument descriptions

**Table 2-24. Assembly method arguments**

| Argument | Description |
|---|---|
| *session* | Identifies an open repository session. |
| *document_id* | Identifies a document with which you want to associate the assembly. Use the document's object ID or an indirect reference (*@object_id*) that points to the document. |
| *virtual_doc_id* | Identifies the virtual document from which you are creating the assembly. This can be the same as the object ID provided for document_id. Use the virtual document's object ID or an indirect reference in the format *@object_id* that points to the virtual document. |
| *interrupt_freq* | Specifies how often the system stops the assembly process and turns control back to the user or application. Use an integer number for this argument.<br><br>If unspecified, the default is -1, and the process continues, without stopping, until completion. All assembly records would attempt to be created before a commit.<br><br>If specified, the process stops whenever:<br>• The specified number of assembly objects (each representing one component of the virtual document) have been created and committed, or<br>• When the number of objects to create is less than the number specified. |

| Argument | Description |
|---|---|
| *qualification* | Determines which components are selected for inclusion in the assembly. Refer to the Usage notes for information about the form and content of this argument. |
| *nodesort_list* | Contains a comma-separated list of the attributes on which you want to sort versions that are candidates for inclusion in the assembly. Use this argument only if the qualification includes an IN DOCUMENT clause that contains the NODESORT BY option. Refer to the Usage notes for more details. |

## Return value

The Assemble method returns TRUE if successful or FALSE if unsuccessful.

**Note:** The Assemble method creates a collection of the items for the assembly. It does not actually create an assembly. Consequently, the return value only indicates whether the items for the collection were successfully collected. Refer to the Usage notes for more information about the process of creating an assembly.

## Usage notes

You must have at least Write permission on the specified virtual document to create an assembly for it.

An assembly represents a virtual document at a particular point in time and under the conditions you specify when you create the assembly. It consists of a collection of assembly objects, each representing one component of the virtual document.

The Assemble method creates a collection of the items that will compose the assembly. Creating the assembly is a four-step process. After you issue the Assembly method, you use Getlastcoll to obtain the collection's ID, and then you must iterate through the collection using Next, and then issue a Close. For a detailed explanation of the process, refer to Creating snapshots, page 176 in *Content Server Fundamentals*.

**Note:** Because the Assemble method opens and manages its own transaction, you cannot issue the Assemble method or create an assembly while an explicit transaction is open. An explicit transaction is a transaction that a user opens with a Begintran method call or a BEGIN TRAN DQL statement.

### Identifying the source and target documents

In general, you use the *document_id* argument to specify which virtual document is the source of the assembly. When this argument is specified and the optional *virtual_doc_id* argument is not specified, the server creates the assembly from and assigns the assembly

to the virtual document identified by *document_id*. In these instances, the *document_id* must identify a virtual document.

If both the *document_id* and the *virtual_doc_id* arguments are included, the server uses the document identified in the *virtual_doc_id* argument as the source of the assembly and assigns the finished assembly to the document identified in the *document_id* argument. Note that in this case, the *document_id* can identify a simple document. This ability to separate the source and target of the Assemble method gives you a way to define several different assemblies for one virtual document. Simply execute the Assemble method once for each assembly you want to create (using a different qualification) and assign the resulting assembly to a different document.

The virtual document identified in the *document_id* argument can also be the same as the virtual document identified in the *virtual_doc_id* argument. This simply associates the assembly with the virtual document specified in the *virtual_doc_id* argument.

### Specifying the qualification

The *qualification* determines which components are included in the assembly. This is an optional argument. If it is not included, the server assembles all the components of the specified virtual document, both the directly and indirectly contained components. The keyword DESCEND is implicit if the qualification is not included. Including the qualification allows you to specify which of the components you want. The statement assembles only those components that meet the criteria in the qualification.

The *qualification* is that portion of the SELECT statement that follows the keyword FROM in the statement. The *qualification* must begin with a type name and must include either an IN DOCUMENT or an IN ASSEMBLY clause. It can also include a SEARCH clause and/or a WHERE clause. The *qualification* is specified as a character string. (Refer to The IN DOCUMENT clause, page 134, The IN ASSEMBLY clause, page 137, The SEARCH clause, page 138, and The WHERE clause, page 142 of the *Content Server DQL Reference Manual* for more information.)

If you include an IN DOCUMENT clause that includes a NODESORT BY option, then you must include the *nodesort_list* argument in the method call. This argument must contain a comma-separated list of attributes that matches the list specified in the NODESORT BY option in the qualification. (Refer to The NODESORT BY option, page 137 in the *Content Server DQL Reference Manual* for information.)

The *object_id* that you specify in the qualification's IN DOCUMENT or IN ASSEMBLY clause must match the object ID of the document that is the source of the assembly (refer to Identifying the source and target documents, page 101 ).

If you do not specify a qualification, the server builds a SELECT statement with an IN DOCUMENT clause and uses the object ID specified as the source document in the Assemble method as the *object_id* for the clause. Also, if any of the selected direct components were added to the virtual document using late binding, the server chooses the root object of the version tree containing the component.

### Interrupt frequency

The *interrupt_freq* argument defines how many assembly objects are created with each execution of Next. After the specified number of objects are created, control is returned to the application. At that time, the application can issue another Next, to process the next batch or it can terminate the process. Specify a positive integer number. If you do not specify the *interrupt_freq*, its default value is -1, which means that all chosen components are processed before control is returned to the application.

Setting interrupt_freq commits and destroys objects in stages and thereby can keep the database log size small, and can provide more efficient use of the log, and can shorten the duration a component document is locked.

## Related methods

Disassemble, page 190

## Examples

These examples presume a virtual document called book_id that has three direct components, called Chap_1_id, Chap_2_id, and Chap_3_id. The Chap_1_id component has one component, called Section_id.

This example creates an assembly for the virtual document that includes components at all levels. (DESCEND by default)

```
status = dmexec("assemble,s0," book_id)
if (status != 1)
{  err_mess = dmget("getmessage,s0");
   print err_mess
   exit
}

coll_id = dmget("getlastcoll,s0")
if (coll_id == NULL)
{  err_mess = dmget("getmessage,s0");
   print err_mess
   exit
}

status = dmexec("next,s0," coll_id)
if (status != 1)
{  err_mess = dmget("getmessage,s0");
   print err_mess
   exit
}

status = dmexec("close,s0,q0")
if (status != 1)
{  err_mess = dmget("getmessage,s0");
   print err_mess
   exit
```

```
}
```

This example creates an assembly for the top-level virtual document. It also specifies an interrupt frequency of 2.

```
status = dmexec("assemble,s0," book_id "," book_id ",2, \
dm_sysobject in document id ('" book_id "')")
if (status != 1)
{  err_mess = dmget("getmessage,s0");
   print err_mess
   exit
}

coll_id = dmget("getlastcoll,s0")
if (coll_id == NULL)
{  err_mess = dmget("getmessage,s0");
   print err_mess
   exit
}

while (dmexec("next,s0," coll_id))
{  count = dmget("get,s0," coll_id ",count");
   print count
}

status = dmexec("close,s0,q0")
if (status != 1)
{  err_mess = dmget("getmessage,s0");
   print err_mess
   exit
}
```

This example creates an assembly and assigns it to the new document. It also has a WITH clause in it.

```
assembly_id = dmget("create,s0,dm_document");
if (assembly_id == NULL)
{  err_mess = dmget("getmessage,s0");
   print err_mess
   exit
}

status = dmset("set,s0,last,object_name", "Assembly_bok")
status = dmexec("save,s0,last")
if (status != 1)
{  err_mess = dmget("getmessage,s0");
   print err_mess
   exit
}

status = dmexec("assemble,s0," assembly_id "," book_id \
",,dm_sysobject in document id ('" book_id "')\
descend with any r_version_label = 'CURRENT'")
if (status != 1)
{  err_mess = dmget("getmessage,s0");
   print err_mess
   exit
}

coll_id = dmget("getlastcoll,s0")
```

```
if (coll_id == NULL)
{  err_mess = dmget("getmessage,s0");
   print err_mess
   exit
}

status = dmexec("next,s0," coll_id)
if (status != 1)
{  err_mess = dmget("getmessage,s0");
   print err_mess
   exit
}

status = dmexec("close,s0,q0")
if (status != 1)
{  err_mess = dmget("getmessage,s0");
   print err_mess
   exit
}
```

This example demonstrates the use of the NODESORT option in the qualification:

```
status = dmexec("assemble,s0," book_id "," book_id ",, \
dm_sysobject in document id ('" book_id "')\
NODESORT by title,title")
if (status != 1)
{  err_mess = dmget("getmessage,s0");
   print err_mess
   exit
}

coll_id = dmget("getlastcoll,s0")
if (coll_id == NULL)
{  err_mess = dmget("getmessage,s0");
   print err_mess
   exit
}

while (dmexec("next,s0," coll_id))
{  count = dmget("get,s0," coll_id ",count");
   print count
}

status = dmexec("close,s0,q0")
if (status != 1)
{  err_mess = dmget("getmessage,s0");
   print err_mess
   exit
}
```

# Assume

## Purpose

Gives ownership of an existing repository session to a new user.

## Syntax

```
dmAPIExec("assume,session,user_auth_name,
password|login_ticket
[,domain][,user_arg]")
```

## Argument descriptions

**Table 2-25. Assume method arguments**

| Argument | Description |
|---|---|
| *session* | Identifies the session for which the user wishes to assume ownership. This must be the current primary repository session, identified by a session identifier in the format s*n;* for example, s0. |
| *user_auth_name* | Identifies the user who wants ownership of the session. Use the value in the user's user_os_name attribute.<br><br>Do not supply this argument when attempting a Windows unified logon. |
| *password* | Specifies the user's password. Valid password formats are:<br><br>*password*<br>DM_PLUGIN=*plugin_id/password*<br><br>where *plugin_id* is the plugin identifier for an authentication plugin.<br><br>If the password is encrypted, the format for the password argument is:<br><br>DM_ENCR_PASS=*encrypted_password*<br><br>where *encrypted_password* is one of the following in encrypted form:<br><br>*password*<br>DM_PLUGIN=*plugin_id/password*<br><br>Refer to the Usage notes for details.<br><br>Do not supply this argument if you are using a login ticket or attempting a Windows unified logon. |

| Argument | Description |
|---|---|
| | A password may not exceed 8,192 characters in length. |
| *login_ticket* | A ticket used by Content Server, in place of a password, to authenticate the user assuming ownership of the session. Login tickets are returned by the Getlogin method. Refer to Using login tickets, page 109 for more information about obtaining and using login tickets.<br><br>Do not supply this argument if you provide the user's password or if you are attempting a Windows unified logon. |
| *domain* | Specifies the domain for the user name and password.<br><br>The domain may be provided as part of the username argument, in the format domain\username, for any version of the product.<br><br>On UNIX platforms, domain values are only used if the user authentication mechanism is set up to use domains.<br><br>You must have a client library (DMCL) that is 3.1.5 or later on the client machine if you are providing the domain as the fourth argument (for example: repository, username, password, domain).<br><br>Do not supply this argument when attempting a Windows unified logon.<br><br>If you are providing a user_arg argument, you must include the placeholding comma for the domain argument in the command line. For example:<br><br>assume, *session* , *user_auth_name* , *password* ,, *user_arg* |
| *user_arg* | Character string argument that can be used to send a value to a user-written user-validation program.<br><br>Do not supply this argument when attempting a Windows unified logon. |

## Return value

The Assume method returns TRUE if successful or FALSE if unsuccessful.

## Usage notes

The Assume method lets you transfer the ownership of a particular repository session from one user to another within an application.

When an application issues the Assume method, the server authenticates the new user and, if the authentication succeeds, gives ownership of the session to the user. The server automatically resets any security-related and cache information for the session as needed.

The Assume method supports application-level connection pooling. Enabling connection pooling by setting the appropriate keys in the dmcl.ini file is not needed.

### Authenticating the user

Content Server uses the values in the *user_auth_name* and *password* arguments to authenticate the user wishing to assume the connection. If the authentication is to be performed by an authentication plug-in module and the plug-in identifier is not specified in the user's user_source attribute, you must prefix the password with the plug-in module's identifier. (The identifier is defined by the plugin module.) The format of a password with a plugin identifier is:

`DM_PLUGIN=plugin_id/password`

The plug-in identifier tells Content Server which authentication plug-in to use to authenticate the user. If Content Server cannot find the module, the method fails.

If a plug-in module is not identified in the arguments, Content Server authenticates the user by the mechanism defined in the user_source attribute of the user's dm_user object.

- If the attribute indicates that the user is authenticated against the operating system using the default mechanism, the user is authenticated using the default mechanism.
- If the attribute is set to LDAP, Content Server authenticates the user against the LDAP server.
- If the attribute is set to a plug-in identifier, Content Server authenticates the user using the plug-in module. If Content Server cannot locate the plug-in, the server authenticates the user using the platform's default mechanism.

On Windows platforms, the user identified in the *user_auth_name* argument must have the right to logon interactively within the domain security policy.

### Using encrypted passwords

If you are using encrypted passwords, the *password* argument must specified as

`DM_ENCR_PASS=encrypted_password`

*encrypted_password* is the encrypted form of either of the valid password formats. The password must have been encrypted using the Encryptpass method.

When the DMCL receives the method, it decrypts the string using the AEK. The AEK must be initialized within the API session before the DMCL can execute the method. The recommended way to initialize the AEK is to use an Initcrypto method. If you do not use an Initcrypto method, the Assume method looks for the AEK

in the location identified in the environment variable DM_CRYPTO_FILE. If the AEK location is not identified in DM_CRYPTO_FILE, the method assumes it is in %DOCUMENTUM%\dba\secure\aek.key ($DOCUMENTUM/dba/secure/aek.key). When the key is found, the method initializes the AEK before proceding.

### Commas in user_auth_name and password

The value in the user_auth_name or password argument can contain commas. However, you must enclose the argument in single quotes if the value contains a comma. For example, suppose the user JaneDoe has the password “violet,eyes”. The Assume method would be:

```
dmAPIExec("assume,s0,JaneDoe,'violet,eyes'")
```

### Using login tickets

The first time a user assumes ownership of a session, you must provide the user’s password in the Assume method. After the user has ownership, use the Getlogin method to obtain a login ticket for the user. (Login tickets are valid only for the user who issues the method.) Then, you can use the login ticket in place of the user’s password if the user requires subsequent connections to the repository.

## Related methods

Connect, page 155
Getlogin, page 247
Initcrypto, page 269

## Example

```
repository = "testenv"
session = dmAPIGet("connect," & repository & ",user1,passwd1")
If session = "" Then
print "Error connecting to repository '" & repository & "'"
err_msg = dmAPIGet("getmessage," & session)
print err_msg
dmExit(1)
End If

inf_msg = dmAPIGet("getmessage," & session)
print inf_msg
err_flag = dmAPIExec("assume," & session & ",user2,passwd2")
If err_flag = 0 Then
print "Error assuming user 'user2'"
err_msg = dmAPIGet("getmessage," & session)
print err_msg
dmExit(1)
End If

inf_msg = dmAPIGet("getmessage," & session)
print inf_msg
```

```
err_flag = dmAPIExec("disconnect," & session)
if err_flag = 0 Then
print "Error disconnecting from repository '" & repository & "'"
err_msg = dmAPIGet("getmessage," & session)
print err_msg
dmExit(1)
End If
```

# Attach

## Purpose

Attaches an object to a lifecycle, replaces an object's lifecycle, or removes an object's lifecycle.

## Syntax

```
dmAPIExec("attach,session,object_id
[,default|policy_id[,position|state_name]]
[,alias_set_position|alias_set_name]")
```

## Arguments

**Table 2-26. Attach method arguments**

| Argument | Description |
|---|---|
| *session* | Identifies an open repository session. |
| *object_id* | Identifies the SysObject to which to attach the lifecycle. Use the object's object ID.<br><br>If the object ID identifies a replica object or you include an indirect reference (*@objectid*), the source object is affected, not the replica or mirror object, and the value in *policy_id* must reference a lifecycle in the same repository as the source object. |
| default | Attaches the default lifecycle, as defined in the data dictionary. |
| *policy_id* | Identifies the lifecycle to attach. Use the policy's object ID.<br><br>If you do not specify default or a *policy_id*, the SysObject is detached from its current policy and r_policy_id is set to NULL. |
| *position* | Identifies the state to which you want to attach the object by its position in the policy. |
| *state_name* | Identifies the state to which you want to attach the object by its name. |

| Argument | Description |
|---|---|
| *alias_set_position* | Identifies the alias set object used to resolve any aliases found in the attached object. The alias set object must be defined in the lifecycle. It is identified by its index position in the alias_set_ids attribute of the policy object.<br><br>If you use this argument, do not include alias_set_name. |
| *alias_set_name* | Identifies the alias set object used to resolve any aliases found in the attached object. The alias set object must be defined in the lifecycle. It is identified by its object name.<br><br>If you use this argument, do not include alias_set_position. |

## Return value

The Attach method returns TRUE if successful or FALSE if unsuccessful.

## Usage notes

The Attach method can:

- Attach an object with no current lifecycle to a lifecycle
- Replace an object's lifecycle with another lifecycle
- Detach an object from its lifecycle

### Permissions

You must have the following permissions to use Attach:

- You must own the document that you attaching to a lifecyle or you must be a superuser.
- You must have at least Relate permission on the lifecycle (the dm_policy object) to which you are attaching a document.
- If you are attaching a document to a lifecycle and the state to which you are attaching the document has action procedures that change the document, the user who is performing the action procedures must have Write permission on the document.

  **Note:** The user performing the action procedures is determined by the a_bpaction_run_as attribute in the docbase config object. Typically, this attribute is set to the session user.

### Attaching or replacing lifecycles

To associate a lifecycle with an object that currently has no associated policy or to replace an object's current lifecycle, use the following syntax:

```
dmAPIExec("attach,session,sysobject_id[,default|policy_id
[,position|state_name]][,alias_set_position|alias_set_name]")
```

To attach the object to the default lifecycle defined for the object's type, use the keyword default. To identify some other lifecycle, use the policy's object ID. You can specify any lifecycle for which the object's type is defined as a valid type. (Valid types for a lifecycle are defined in the included_types and include_subtypes attributes of the policy's definition. For details, refer to Chapter 10, Lifecycles, in *Content Server Fundamentals*.)

The lifecycle must be an installed lifecycle.

By default, the object is attached to the base state of the lifecycle. To attach the object to another state in the policy, use either the *position* or *state_name* arguments. Do not use both. The state you specify must be an attachable state.

If you attach an object to an exception state, then you can only resume the object back to the base state.

The Attach method launches the bp_transition method to evaluate the state's entry criteria and perform the state's actions.

Use the *alias_set_position* or alias_set_name argument to identify an alias set object. Typically, these arguments are used in applications that create new documents and attach the documents to a lifecycle before saving them to a repository.

The application creates a SysObject such as a document and sets any of the following attributes to aliases: owner_name, acl_name, or acl_domain. When the object is saved to the repository, these aliases are resolved to an actual value. If the application attaches the object to a lifecycle and identifies an alias set in the method, the server resolves the aliases using the alias set specified in the method.

The alias set identified in the Attach method must be listed in the alias_set_ids attribute of the lifecycle.

### Removing lifecyles

To detach an object from a lifecycle, use the following syntax:

```
dmAPIExec("attach,session,sysobject_id")
```

Failing to identify a lifecycle in the method removes any attached lifecycle from the object.

## Related methods

Demote, page 176
Promote, page 323
Resume, page 392
Suspend, page 453

## Example

```
status = dmAPIExec("attach," sess "," doc_id ","
pol_id  ",0")
```

# Audit

## Purpose

Initiates auditing of an event.

## Syntax

```
dmAPIExec("audit,session[,object_id],event_name
[,audit_subtypes][,controlling_app]
[,policy_id[,state_name]][,sign_audit][,authentication]
[,event_description][,attribute_list]
[,esignature_required]")
```

## Arguments

**Table 2-27. Audit method arguments**

| Argument | Description |
|---|---|
| *session* | Identifies an open repository session. |
| *object_id* | Identifies a particular object or object type for which the event is audited. Use the object's object ID for individual objects. For object types, use the object ID of the type's dm_type object.<br><br>The object or object type must be a valid type for the event specified in *event_name*. Appendix B, System Events, lists valid targets for system events.<br><br>If unspecified, all occurrences of the event are audited. |
| *event_name* | Defines the event to audit. Valid system-defined events are listed in Appendix B, System Events. Other valid events are:<br>• Application eventsApplication events, page 121 describes application events.<br>• Job executionsJob executions, page 121 describes auditing job executions. |

| Argument | Description |
|---|---|
| *audit_subtypes* | Whether to audit subtypes of the audited object type. If *object_id* identifies an object type and this attribute is T (TRUE), then the event is audited for all objects of the specified type and its subtypes. The default is T.<br><br>You cannot include this argument if:<br>• *object_id* is set to 0 or omitted<br>• *object_id* identifies a particular object |
| *controlling_app* | An application code. If this argument is set, Content Server only audits the event for those objects whose a_controlling_app value is set to this application code.<br><br>Setting this argument to dm_all_applications is equivalent to not including the argument. The event is audited for all objects of the specified type regardless of their controlling application.<br><br>You cannot include this argument if:<br>• *object_id* is set to 0 or omitted<br>• *object_id* identifies a particular object<br>• *event_name* is dm_all, all, or dm_all_workflow |
| *policy_id* | Identifies a lifecycle. Use the object ID of the lifecycle's dm_policy object. If the event is dm_bp_attach, you can set *policy_id* to the keyword default.<br><br>If *policy_id* is set and *object_id* identifies an object type, Content Server audits the event for objects of that type that are attached to all versions of the lifecycle. If *object_id* identifies an individual object, then the object is audited for the event only when it is attached to this lifecycle.<br><br>You cannot include this argument if:<br>• *object_id* is set to 0 or omitted<br>• *event_name* is dm_all, all, or dm_all_workflow |

| Argument | Description |
|---|---|
| *state_name* | Identifies a particular state in the lifecycle identified in *policy_id*. Use the state's name.<br><br>If *state_name* is included and *object_id* identifies an object type, Content Server audits the event for objects of the type that are in this particular lifecycle state. If *object_id* identifies an individual object, the object is audited for this event only when the object is in this lifecycle state.<br><br>*state_name* is ignored unless you also include *policy_id*.<br><br>You cannot include this argument if:<br>• *object_id* is set to 0 or omitted<br>• *event_name* is dm_all, all, or dm_all_workflow |
| *sign_audit* | Determines whether Content Server signs audit trail entries generated for the event registered by the method. Setting this to T (TRUE) directs the server to sign the generated audit trail entries. The signature is stored in the audit_signature attribute of the entry.<br><br>You cannot include this argument if you have included the *audit_subtypes* argument.<br><br>The default is False. |
| *authentication* | Indicates whether the application should authenticate the user before creating an audit trail entry for the event.<br><br>This setting must be enforced by the application in which the event occurs. Content Server ignores this argument.<br><br>You cannot include this argument if:<br>• *object_id* is set to 0 or omitted<br>• *event_name* is dm_all or all<br><br>(You can include this argument for the dm_all_workflow event.) |

| Argument | Description |
|---|---|
| *event_description* | Defines a user-friendly name for the event.<br><br>If included for a system-defined event, this value overrides the event's user-friendly name stored in the data dictionary.<br><br>You cannot include this argument if:<br>• *object_id* is set to 0 or omitted<br>• *event_name* is dm_all or all<br><br>(You can include this argument for the dm_all_workflow event.) |
| *attribute_list* | Identifies one or more attributes of the audited objects to be recorded in the audit trail entry. Identify the attributes by name, separate mulitple attribute names with commas, and enclose the list in single quotes. You can include single-valued attributes, repeating attributes, and custom attributes.<br><br>If the audited event is a workflow event, the attributes must those defined for dm_workflow (even though *object_id* identifies a process object).<br><br>The attribute's name and value is recorded in the audit trial entry. For details, refer to Recording attributes in an audit trail entry, page 121 .<br><br>You cannot include this argument if:<br>• *object_id* is set to 0 or omitted<br>• *event_name* is dm_all or all<br><br>(You can include this argument for the dm_all_workflow event.) |
| *esignature_required* | Specifies whether the event requires an electronic signature. The default is F (FALSE). |

## Return value

The Audit method returns TRUE if successful or FALSE if unsuccessful.

## Usage notes

You must have Config_audit privileges to use this method.

Executing Audit creates a dmi_registry object that records the auditing request. The values you include in the Audit arguments are recorded in the registry object. Thereafter, if the event is a system event, Content Server creates an audit trail entry whenever the event occurs. If the event is an application event, the application can examine the registry to determine whether to create an audit trail entry for the event. (Table 2–125, page 368 in the *EMC Documentum Object Reference* describes the attributes of a registry object. Auditing is described in detail in Auditing, page 406 in the *Content Server Administrator's Guide*. )

Auditing doesn't generate an email notification of the event's occurrence. To receive an email notification of an event's occurrence, you must use the Register method to register for the notification. You can both audit an event and register for notification of its occurrence.

You can execute the Audit method only once for a particular combination of event, object, and arguments. The Audit method fails if the particular combination of event, object, with other arguments is already being audited.

Executing the Audit method also creates an audit trail entry with the event name dm_audit to record the execution of the Audit method.

### Auditing an event on a particular object

To audit a specific event on a particular object, identify the object in the object_id parameter of the Audit method. For example, to audit all link events on a particular folder whose object ID is 0b0000015236712c, use:

```
audit,s0,0b0000015236712c,dm_link
```

Or, to audit all workflows created from the process definition (dm_process object) represented by 4b000001243583cf, use:

```
audit,s0,4b000001243583cf,dm_createworkflow
```

### Auditing an event on all objects of a type

To audit an event for all instances of a particular object type, identify the object type in the object_id argument. To identify the object type, use the object ID of the type's dm_type object.

For example, to audit check ins on all documents, first obtain the object ID for the dm_document type:

```
retrieve,s0,dm_type where name='dm_document'
```

Then, use the returned object ID in the Audit method:

```
audit,s0,returned object ID,dm_checkin
```

The object type and event must be compatible. That is, the object type must be a valid target of the given event.

### Auditing every occurrence of an event

To audit every occurrence of an event in a repository, specify 0 as the *object_id* value or leave *object_id* empty. For example, to audit all checkins:

```
audit,s0,0,dm_checkin
```

or

```
audit,s0,,dm_checkin
```

This example audits all workflow terminations:

```
audit,s0,0,dm_finishworkflow
```

or

```
audit,s0,,dm_finishworkflow
```

### API events

Table B–1, page 539 in Appendix B, System Events, lists the events associated with API methods, the object type that is a valid target for each event and its trigger.

### Workflow events

All workflow events take the object ID of a process object as the target of the Audit method.

When you audit a single workflow event, such as dm_addnote or dm_finishworkflow, and include an object ID, audit trail entries are created for all occurrences of the event on any workflow instance generated from the given process definition. If you don't include the object ID, the method initiates auditing of all occurrences of the event in the repository.

For the dm_all_workflow event, including the object ID audits all events that occur for all workflows generated from the given process definition. If you don't include an object ID, the server audits all workflow events that occur in the repository.

**Note:** Auditing dm_all_workflow does not generate audit trail entries for the dm_validate, dm_install, dm_invalidate, or dm_uninstall events.

If you audit all workflow events for a particular workflow or all workflows, you cannot include the optional arguments when you register for auditing.

Table B–2, page 544 in Appendix B, System Events, lists the events specific to workflows and the trigger for each. There are some events available for process objects, such as dm_install and dm_validate, that are not specific to workflows. These are listed in Table B–1, page 539.

### Lifecycle events

Table B–3, page 546 in Appendix B, System Events, lists the events specific to lifecycles and the trigger for each. There are some events available for lifecycles, such as dm_install and dm_validate, that are not specific to lifecycles. These are listed in Table B–1, page 539.

### Job executions

You can audit the start of a job's execution by specifying dm_jobstart as *event_name* and the object ID of the job object as the *object_id*. For example, the following method call audits the execution start of the Content Warning job:

```
audit,s0,080000012567321f,dm_jobstart
```

You can obtain the object ID of the job's job object, if needed, using the Retrieve method. For example:

```
API>retrieve,s0,dm_job where object_name='dm_ContentWarning'
. . .
080000012567321f
```

Use the job's internal name. Refer to Chapter 12, Tools and Tracing, in *Content Server Administrator's Guide* for the name of each job.

### All events in a repository

If you specify all as the event name, the server creates audit trail entries for every event in the repository. To start auditing all events, use the following syntax:

```
audit,session,0,all
```

or

```
audit,session,0,dm_all
```

If you choose to audit all events, you cannot include the optional arguments.

Auditing all events can create very large numbers of audit trail entries because frequently called methods, such as Fetch, are audited as part of all. If you audit all, you must manage the audit trail carefully, to ensure that you do not run out of space.

### Application events

Application events are events defined by an application. For example, suppose you want to audit a particular operation in an application. You can define an event name for the operation and use Audit to register that event for auditing. When the application performs the operation, it can check the registry objects to determine whether the operation is audited and if so, issue a Createaudit to create the audit trail entry for the event before completing the operation.

### Recording attributes in an audit trail entry

If you include the attributes_list argument, Content Server records the attribute names identified in the argument in the event's registry object. When the event occurs, the server stores both the attribute names and their current values in the audit trail entry. The names and values are stored in the audit trail entry's attribute_list attribute.

The attribute_list attribute is a string datatype. If the set of names and values is too long to store in the attribute_list attribute, the overflow is stored in a dmi_audittrail_attrs object. The object ID of the audit trail attrs object is stored in the audit trail entry in the attribute_list_id attribute.

## Related methods

Register, page 346
Unaudit, page 466
Unregister, page 484

## Example

```
status = dmexec("audit," session ",0900007b80005640,dm_save")
```

# Authenticate

## Purpose

Validates a user name and password.

## Syntax

```
dmAPIExec("authenticate,session [,repository_name],user_auth_name,
password|ticket[,domain][,user_arg]")
```

## Arguments

**Table 2-28. Authenticate method arguments**

| Argument | Description |
|---|---|
| *session* | Identifies an open repository session.<br><br>If the *repository_name* argument is included, you must specify apisession as the *session* value. |
| *repository_name* | Identifies a repository.<br><br>If you include this argument, you must specify apisession as the *session*. |
| *user_auth_name* | Identifies the current user. Use the value in the user's user_os_name attribute.<br><br>Do not supply this argument when attempting a Windows unified logon. |
| *password* | Specifies the user's password. Valid password formats are:<br><br>*password*<br>DM_PLUGIN=*plugin_id/password*<br><br>where *plugin_id* is the plugin identifier for an authentication plugin.<br><br>If the password is encrypted, the format for the password argument is:<br><br>DM_ENCR_PASS=*encrypted_password*<br><br>where *encrypted_password* is one of<br><br>*password*<br>DM_PLUGIN=*plugin_id/password*<br><br>Refer to the Usage notes for details.<br><br>Do not supply this argument if you are using a login ticket or attempting a Windows unified logon. |

| Argument | Description |
|---|---|
| | A password may not exceed 8,192 characters in length. |
| *ticket* | The value returned by a Getlogin method. Used by applications to establish a session with the current repository as the current user without providing the user's password.<br><br>Do not supply this argument when attempting a Windows unified logon. |
| *domain* | Identifies the domain for the user_auth_name and password.<br><br>The domain may be provided as part of the username argument, in the format *domain**user_auth_name*, for any version of the product.<br><br>You must have a client library (DMCL) that is 3.1.5 or later on the client machine if you are providing the domain as the fourth argument (for example: repository, username, password, domain).<br><br>Do not supply this argument when attempting a Windows unified logon.<br><br>If you are including the user_arg argument, you must include the placeholding comma for the domain argument. For example:<br><br>authenticate,*session*,*user_auth_name*,*password*,,*user_arg* |
| *user_arg* | Character string argument that can be used to send a value to a user-written user-validation program.<br><br>Do not supply this argument when attempting a Windows NT unified logon. |

## Return value

Authenticate returns TRUE if successful or FALSE if unsuccessful.

## Usage notes

Authenticate allows an application to authenticate a user and password against the current repository without opening another repository session. Some applications may

require user authentication before proceeding with a particular operation. In such circumstances, use Authenticate to perform the authentication. To do this, issue the method without including the *repository_name* argument.

If you include the *repository_name* argument, the method establishes a connection to the specified repository to perform the authentication.

### Authenticating the user

Content Server uses the values in the *user_auth_name* and *password* arguments to authenticate the user. If the authentication is to be performed by an authentication plug-in module and the plug-in is not identified in the user's user_source attribute, you must prefix the password with the plug-in module's identifier. (The identifier is defined by the plugin module.) The format of a password with a plugin identifier is:

`DM_PLUGIN=plugin_id/password`

The plug-in identifier tells Content Server which authentication plug-in to use to authenticate the user. If Content Server cannot find the module, the method fails.

If a plug-in module is not identified in the arguments, Content Server authenticates the user by the mechanism defined in the user_source attribute of the user's dm_user object.

- If the attribute indicates that the user is authenticated against the operating system using the default mechanism, the user is authenticated using the default mechanism.
- If the attribute is set to LDAP, Content Server authenticates the user against the LDAP server.
- If the attribute is set to a plug-in identifier, Content Server authenticates the user using the plug-in module. If Content Server cannot locate the plug-in, the server authenticates the user using the platform's default mechanism.

### Using encrypted passwords

If you are using encrypted passwords, the *password* argument must specified as

`DM_ENCR_PASS=encrypted_password`

*encrypted_password* is the encrypted form of either of the valid password formats. The password must have been encrypted using the Encryptpass method.

When the DMCL receives the method, it decrypts the string using the AEK. The AEK must be initialized within the API session before the DMCL can execute the method. The recommended way to initialize the AEK is to use an Initcrypto method. If you do not use an Initcrypto method, the Authenticate method looks for the AEK in the location identified in the environment variable DM_CRYPTO_FILE. If the AEK location is not identified in DM_CRYPTO_FILE, the method assumes it is in %DOCUMENTUM%\dba\secure\aek.key ($DOCUMENTUM/dba/secure/aek.key). When the key is found, the method initializes the AEK before proceding.

### Commas in user_auth_name and password

The value in the user_auth_name or password argument can contain commas. However, you must enclose the argument in single quotes if the value contains a comma. For example, suppose the user JaneDoe in the engr domain has the password “violet,eyes”. The Authenticate method would be:

```
dmAPIExec("authenticate,s0,JaneDoe,'violet,eyes',engr")
```

### Using a login ticket

If there are multiple Content Servers running against the repository and you provide a login ticket instead of a password in the method, the method must be executed by the same server that executed the Getlogin method to obtain the ticket. This server is identified in the ticket itself. (Refer to Getlogin, page 247, for details about the format of a login ticket.)

To ensure that the same server is used for both the Getlogin and subsequent Authenticate method, you can either specify a valid session with the server in the session argument or include the repository_name argument as a fully qualified name. For example:

```
dmAPIExec("authenticate,apisession,engineering.engr1@myengrhost,johndoe,fawn")
```

engineering.engr1@myengrhost is a fully qualified *repository_name* argument. engineering is the repository name, engr1 is the name of the server, and myengrhost is the name of the host machine.

## Related methods

Initcrypto, page 269

## Example

This example authenticates johndoe in the engineering repository:

```
dmAPIExec("authenticate,apisession,engineering,johndoe,fawn")
```

This example authenticates SallySue in the repository to which she is currently connected:

```
dmAPIExec("authenticate,S0,,SallySue,marvelous")
```

# Begintran

## Purpose

Opens an explicit database transaction.

## Syntax

```
dmAPIExec("begintran,session")
```

## Arguments

**Table 2-29. Begintran method arguments**

| Argument | Description |
|---|---|
| *session* | Identifies an open repository session. |

## Return value

The Begintran method returns TRUE if successful or FALSE if unsuccessful.

## Usage notes

If you execute a Begintran method, you open an explicit transaction. Any changes that you Save or Checkin to the repository are not committed to the repository until you explicitly commit them by issuing a Commit method call. (The Commit method also closes the transaction.)

Note that you can issue an Abort method to terminate an explicit transaction. When you do so, any changes that you Saved or Checked in are discarded.

## Related methods

Abort, page 40
Commit, page 151

## Example

```
status = dmexec("begintran,s0")
 if (status != 1)
 {  err_mess = dmget("getmessage,s0")
    print err_mess
    exit
 }

doc_id = dmget("create,s0,dm_document");
 if (doc_id == NULL)
 {  err_mess = dmget("getmessage,s0")
    print err_mess
    exit
 }

status = dmset("set,s0,last,object_name", "my_document")

status = dmexec("save,s0,last")
 if (status != 1)
 {  err_mess = dmget("getmessage,s0")
    print err_mess
    exit
 }

  status = dmexec("commit,s0")
  if (status != 1)
  {  err_mess = dmget("getmessage,s0")
     print err_mess
     exit
  }
```

# Bindfile

## Purpose

Binds a content file associated with an object to another object.

## Syntax

```
dmAPIExec("bindfile,session,object_id,page_number,src_id
,src_page_number")
```

## Arguments

**Table 2-30. Bindfile method arguments**

| Argument | Description |
|---|---|
| *session* | Identifies an open repository session. |
| *object_id* | Identifies the object to which you are binding the content. Use the object's ID or an indirect reference (*@object_id*) that points to the object. |
| *page_number* | Specifies the position of the new content in the ordered contents of the target object (the object specified in *object_id*). Specify 0 if this is the first content for the object. |
| *src_id* | Identifies an object in which the content currently resides. This can be any object to which the content is currently linked, but the object must be in the same repository as the object identified in *object_id*. Use the object's object ID. |
| *src_page_number* | Identifies the page number of the content within the source object. (Page numbers are the ordering numbers given to the contents of an object. Each number refers to one content file.) |

## Return value

The Bindfile method returns TRUE if successful or FALSE if unsuccessful.

## Usage notes

The format of the specified content must match the format specified in the target object's a_content_type attribute. If the target object is new and has not defined content type, the Bindfile method sets the content type to match that of the source object.

If you specify a page number that already exists for the target object, then the content that you are binding to the object overwrites the content holding that page number.

To use Bindfile, you must have Read permission on the specified source object.

The object and the content you are binding to the object must reside in the same repository.

## Related methods

Removecontent, page 355
Setfile, page 434

## Example

This example creates a document and associates a content object with that document. Then, the example creates a second document and binds the content of the first document to the second.

```
/* Create the first document */
doc_id = dmAPIGet ("create, " session ",dm_document")
dmAPIExec("setfile," session "," doc_id ",notice.txt, text")
dmAPIExec("save," session "," doc_id)

/* Create second document*/
new_doc_id = dmAPIGet ("create," session ",dm_document")

/* Bind content of first document to the second */
dmAPIExec("bindfile," session "," new_doc_id ",0," doc_id ",0");
dnAPIExec("save," session "," new_doc_id)
```

# Branch

## Purpose

Creates and checks out a new version of an object.

## Syntax

```
dmAPIGet("branch,session,object_id[,version_label]")
```

## Arguments

**Table 2-31. Branch method arguments**

| Argument | Description |
|---|---|
| *session* | Identifies an open repository session. |
| *object_id* | Identifies the object that you want to branch. Use the object's ID or an indirect reference (*@object_id*) that points to the object. |
| *version_label* | Specifies the object version from which you want to branch. (Refer to the Usage notes for an expanded description.) |

## Return value

The Branch method returns the ID of the new version of the object.

## Usage notes

If your repository is running under folder security, you must have at least Write permission for all the cabinets or folders where the document is stored or linked.

The Branch method is a convenient way to check out an unchangeable document. The method creates a copy of the document, assigns the copy a branched version label, and checks outs the copy to you. The original is not touched.

If you specify a version label, the server searches the version tree that contains the specified object and branches from the version that has the specified version label. If the specified version is not found on the version tree, the method does not succeed.

If the version that is the root of the branch is not unchangeable already, the Branch method marks it as immutable.

The new, branched copy is stored in the storage area defined by the default_storage attribute of the object's type.

Executing the Branch method sets the r_lock_machine attribute for the new, branched version of the object. This attribute records the name of the client machine on which the user who issued the method is working.

## Related methods

Saveasnew, page 409

## Example

```
dmAPIGet(sprintf("branch,%s,%s,,%s",session,doc_id,"1.2",T));
```

# Cachequery

## Purpose

Executes a query as a read-only query and stores the results in a query cache file.

## Syntax

```
dmAPIGet("cachequery,session,dql_query")
```

## Arguments

**Table 2-32. Cachequery method arguments**

| Argument | Description |
|---|---|
| *session* | Identifies an open repository session. |
| *dql_query* | Defines the DQL SELECT query to be executed. The query must be less than or equal to 255 characters in length if you are using DDE as the communications protocol between the external application and Content Server. |

## Return value

The Cachequery method returns a collection identifier.

## Usage notes

**Note:** The Cachequery method has been superseded by the Query_cmd, page 334 method, which was introduced with the persistent client caching functionality.

Use the Cachequery method when you are executing a SELECT query whose results are generally static. For example, you might use Cachequery to return the users in the repository. Query cache files are maintained within and across sessions.

The query results returned by Cachequery are cached only if the client_persistent_caching and cache_queries keys in the dmcl.ini file are set to T. If either key is set to F (FALSE), the results are not cached. The default values for these keys are T for client_persistent_caching and F for cache_queries. (For instructions on setting these keys, refer to Enabling and disabling persistent client caching, page 192, in the *Content Server Administrator's Guide*.)

When cached, the results are stored in a file in the client's local area. The collection identifier returned by the Cachequery method points to this file.

**Note:** If you want to execute a query other than a SELECT, use Execquery, Readquery, or Query.

## Related methods

Execquery, page 203
Readquery, page 342
Query, page 337
Query_cmd, page 334

## Example

This example:

- Queries the repository
- Processes the query results
- Closes the collection that holds the query results

```
cmd_str = "cachequery," session ",select name from dm_type"
query_id = dmAPIGet( cmd_str )
if (query_id == NULL) {
err_msg = dmAPIGet("getmessage," session)
print "Error querying the repository"
print err_msg
exit
}

resp_cntr = 0
while( dmAPIExec("next," session "," query_id) > 0 ) {
obj_id = dmAPIGet("get," session "," query_id ",name")
if (obj_id == NULL) {
err_msg = dmAPIGet("getmessage," sessi)
print "Error retrieving name from the query collection'
print err_msg
exit
}
print ++resp_cntr " : " obj_id
}

err_flag = dmAPIExec("close," session "," query_id)
if (err_flag == 0) {
err_msg = dmAPIGet("getmessage," session)
print "Error closing the query collection"
print err_msg
exit
}
```

# Changepassword

## Purpose

Changes a user's password from his or her current password to a new password. (This method is not supported for LDAP users if the directory server is MS Active Directory or Oracle Directory Server).

## Syntax

```
dmAPIExec("changepassword,session,
old_password,new_password")
```

## Arguments

**Table 2-33. Changepassword method arguments**

| Argument | Description |
|---|---|
| *session* | Identifies an open repository session. |
| *old_password* | Identifies the user's old password. Valid password formats are:<br>*password*<br>DM_PLUGIN=*plugin_id/password*<br>where *plugin_id* is the identifier for an authentication plugin.<br>A password may not exceed 8,192 characters in length. |
| *new_password* | Identifies the user's new password. The password cannot contain a comma. Valid password formats are:<br>*password*<br>DM_PLUGIN=*plugin_id/password*<br>where *plugin_id* is the identifier for an authentication plugin.<br>A password may not exceed 8,192 characters in length. |

## Return value

The Changepassword method returns TRUE if successful or FALSE if unsuccessful.

## Usage notes

The internal behavior of the Changepassword method depends on the user authentication mechanism in place for the user issuing the method and whether you include a plug-in identifier in the arguments. If the user is authenticated using a plug-in authentication module and the plug-in identifier is not specified in the user's user_source attribute, you must prefix the old and new passwords with the plug-in identifier. The format of the argument with a plug-in identifier is:

```
DM_PLUGIN=plugin_id/password
```

If you include the plug-in identifier, the method sends the old and new passwords to the specfied plug-in module. If the module is not found, the method fails.

If you do not include a plug-in identifier in the arguments, the method examines the user's user_source attribute (in the dm_user argument) to determine how to handle the request:

- If the attribute indicates that the user is authenticated against the operating system using the default mechanism, then the old and new passwords are passed to the dm_change_password program.
- If the attribute is set to LDAP, Content Server performs the password change internally. dm_change_password is not called.
- If the attribute is set to a plug-in identifier, the method passes the old and new passwords to the module. If Content Server cannot locate the plug-in, the server authenticates the user using the platform's default mechanism.

The dm_change_password program is provided with Content Server and resides in %DOCUMENTUM%\dba ($DOCUMENTUM/dba). You can write and install a custom password changing program. For more information about the password changing feature, refer to *Installing Content Server*.

For all users except LDAP users, the method fails if it cannot execute dm_change_password or a specified plug-in module.

For information about creating plug-in authentication modules, refer to Using authentication plug-ins, page 366 in the *Content Server Administrator's Guide*.

### Commas in passwords

The values in the *password* arguments can contain commas. However, you must enclose the argument in single quotes if the password contains a comma. For example, suppose the user JaneDoe wants to change her password "violet,eyes" to bigdog. The Changepassword method would be:

```
dmAPIExec("changepassword,s0,'violet,eyes',bigdog")
```

## Related methods

Initcrypto, page 269

## Example

```
status=dmAPIExec("changepassword,s0,old_password,
   new_password")
if (status != 0)
print dmAPIGet("getmessage,s0")
```

# Checkin

## Purpose

Creates a new version of an object in the repository and removes the lock from the previous version.

## Syntax

```
dmAPIGet("checkin,session,object_id[,save_lock]
{,version_label}")
```

## Arguments

**Table 2-34. Checkin method arguments**

| Argument | Description |
|---|---|
| *session* | Identifies an open repository session. |
| *object_id* | Identifies the object that you want to check in. Specify the object's ID or an indirect reference (*@object_id*) that points to the object.<br><br>You can use either a string literal or a variable. |
| *save_lock* | Determines whether a lock is placed on the newly created version. The default setting for this flag is F, which means that no lock is placed on the new version. To set a lock, specify T (TRUE) for this flag. |
| *version_label* | Defines a version label for the new version. You can specify more than one label.The label can be an implicit version label, a symbolic label, or a combination. If you do not define a label, the server automatically gives the new version an implicit version label and the symbolic label CURRENT. |

## Return value

The Checkin method returns the ID of the new version.

## Usage notes

The Checkin method unlocks an object and saves any changes you may have made to either its attributes or its content. To use the Checkin method, you must have:

- Used the Checkout method to get the object from the repository
- At least Version permission on the object
- At least Write permission on every cabinet or folder where the object is stored if the repository is running under folder security

If you do not assign an implicit version label, the server automatically assigns one to the new version by incrementing the implicit label of the checked out version. For example, if the checked out version has the label 1.4, the new version is given the label 1.5. Similarly, if the checked out version is 1.2.2.3, the new version receives the label 1.2.2.4.

If you do not assign a symbolic version label, the server assigns the symbolic label CURRENT to the new version.

When you check in a new version, the server sets the a_archive attribute of the new version to FALSE. This means that if you have restored an archived document and made changes to it, you will be able to archive the new version.

The server stores the new version in the storage area defined by the default_storage attribute for the object's type. This means that if you want to store the new version in an alternate location, you must explicitly issue a Set method to set the a_storage_type attribute of the object before you check it in. For example, if you are storing a document's content in turbo store, you must explicitly set a_storage_type to dm_turbo_store prior to each check in. (Note that is it not possible to set the default_storage attribute to dm_turbo_store.)

Executing the Checkin method clears the object's r_lock_machine attribute. (This attribute contains the name of the client machine on which the user was working when the object was locked.)

When applied to a policy object, the Checkin method sets the r_definition_state to DRAFT on the new copy, and clones the following attributes:

- entry_criteria_id
- user_criteria_id
- action_object_id
- user_action_id
- type_override_id

## Related methods

Checkout, page 146
Save, page 405
Saveasnew, page 409
Unlock, page 477

## Example

This example performs the following operations:

- Checks out the document
- Changes the title
- Checks in the document

```
err_flag = dmAPIGet("checkout," session "," doc_id)
if (err_flag == 0) {
err_msg = dmAPIGet("getmessage," session)
print "Error checking out the document"
print err_msg
exit
}

err_flag = dmAPIExec("set," session "," doc_id ",title,
"Reference Manual - Chapter One")
if (err_flag == 0) {
err_msg = dmAPIGet("getmessage," session)
print "Error setting document title"
print err_msg
exit
}

new_doc_id = dmAPIGet("checkin," session "," obj_id)
if (new_doc_id == NULL) {
err_msg = dmAPIGet("getmessage," session)
print "Error checking in changes to document '" doc_id "'"
print err_msg
exit
}
```

# Checkinapp

## Purpose

Creates a new version of an object in the repository and removes the lock from the previous version.

## Syntax

```
dmAPIGet("checkinapp,session,object_id[,save_lock]
[,version_label][,old_compound_arch_value]
[,old_special_app_value][,new_compound_arch_value]
[,new_special_app_value]")
```

## Arguments

**Table 2-35. Checkinapp method arguments**

| Argument | Description |
|---|---|
| *session* | Identifies an open repository session. |
| *object_id* | Identifies the object that you want to check in. Specify the object's ID or an indirect reference (*@object_id*) that points to the object.<br><br>You can use either a string literal or a variable. |
| *save_lock* | Determines whether a lock is placed on the newly created version. The default setting for this flag is F, which means that no lock is placed on the new version. To set a lock, specify T (TRUE) for this flag. |
| *version_label* | Defines a version label for the new version. You can specify more than one label. If you do so, the string of labels must be enclosed in single quotes and the labels separated by commas.<br><br>If you do not define a label, the server automatically gives the new version an implicit version label and the symbolic label CURRENT. |
| *old_compound _arch_value* | Sets the a_compound_architecture attribute of the checked-out object. Refer to Application-specific arguments, page 143 for an extended discussion of its use. |

| Argument | Description |
|---|---|
| *old_special_app _value* | Sets the a_special_app attribute of the checked-out object. Refer to Application-specific arguments, page 143 for an extended discussion of its use. |
| *new_compound_arch_ value* | Sets the a_compound_architecture attribute of the new version created by the check in. Refer to Application-specific arguments, page 143 for an extended discussion of its use. |
| *new_special_app _value* | Sets the a_special_app attribute of the new version created by the check in. Refer to Application-specific arguments, page 143 for an extended discussion of its use. |

## Return value

The Checkinapp method returns the ID of the new version.

## Usage notes

The Checkinapp method differs from the Checkin method by providing arguments to set application-specific attributes. To use the Checkinapp method, you must have:

- Used the Checkout method to lock the object in the repository
- At least Version permission on the object
- At least Write permission on the folder or cabinet in which the object is stored if the repository is running under folder security

Executing the Checkinapp method clears the object's r_lock_machine attribute. (This attribute contains the name of the client machine on which the user was working when the object was locked.) It also clears the r_lock_owner and r_lock_date attributes and sets a_archive to FALSE.

### Version labels

If you do not assign an implicit version label, the server automatically assigns one to the new version by incrementing the implicit label of the checked out version. For example, if the checked out version has the label 1.4, the new version is given the label 1.5. Similarly, if the checked out version is 1.2.2.3, the new version receives the label 1.2.2.4.

If you do not assign a symbolic version label, the server assigns the symbolic label CURRENT to the new version.

You can assign multiple version labels by placing them in a single-quoted string. To illustrate, the following example defines two symbolic labels for a document:

```
dmAPGet("checkinapp,s0,09000025e67354f2,'working,marketing'")
```

**Note:** This differs from how specifying multiple version labels is handled in the Checkin method.

### Application-specific arguments

Every SysObject has two attributes, a_compound_architecture and a_special_app, that are intended for internal and user-defined application use.

The a_compound_architecture attribute is used internally by the Virtual Document Manager (VDM) to determine if the object can be structurally changed in the VDM. If an object's a_compound_architecture attribute is set to any value other than an empty string, VDM assumes the object is read only and does not allow any structural changes in the object. The a_compound_architecture attribute is a string-valued attribute with a length of 16.

The a_special_app attribute is used by the server and some Documentum clients to determine if the object is currently reserved for use by another application such as the Documentum Link for Lotus Notes or Interleaf Integration applications. If this is any value other than an empty string, the system assumes that another application currently has control of the object. The a_special_app attribute is a string-valued attribute with a length of 32.

User-defined applications can use these attributes by setting them on check out and check in or when issuing the Unlock method. The Checkout and Checkinapp methods accept arguments that set these attributes. For Checkinapp, the arguments are:

- *old_compound_arch_value*, which sets a _compound_architecture of the checked out version
- *old_special_app_value*, which sets a_special_app for the checked out version

  **Note:** If *old_compound_arch_value* or *old_special_app_value* is not set during the checkin, the existing value for the attribute is not changed.
- *new_compound_arch_value*, which sets a _compound_architecture of the new version created by the check in
- *new_special_app_value*, which sets a_special_app for the new version created by the check in

  **Note:** If new_compound_arch_value or new_special_app_value is not set for the new version, the attribute's value is copied from the checked out object.

### Archiving

When you check in a new version, the server sets the a_archive attribute of the new version to FALSE. This means that if you have restored an archived document and made changes to it, you will be able to archive the new version.

### Specifying storage area

The server stores the new version in the storage area defined by the default_storage attribute for the object's type. If you want to store the new version in an alternate location, you must explicitly issue a Set method to set the a_storage_type attribute of

the object before you check it in. For example, if you are storing a document's content in turbo store, you must explicitly set a_storage_type to dm_turbo_store prior to each check in. (Note that it is not possible to set the default_storage attribute to dm_turbo_store.)

## Related methods

Checkin, page 138
Checkout, page 146
Save, page 405
Saveasnew, page 409
Unlock, page 477

## Example

This example performs the following operations:

- Checks out the document
- Changes the title
- Checks in the document and
  - Sets the version label of the new version to APPROVED and PROCEED.
  - Sets the a_compound_architecture and a_special_app attributes of the new version to Interleaf Compound and Interleaf App,
  - Keeps the a_compound_architecture and a_special_app attributes of the old version unchanged.

```
err_flag = dmAPIGet("checkout," session "," doc_id)
if (err_flag == 0) {
err_msg = dmAPIGet("getmessage," session)
print "Error checking out the document"
print err_msg
exit
}

err_flag = dmAPIExec("set," session "," doc_id ",title,Chapter One")
if (err_flag == 0) {
err_flag = dmAPIGet("getmessage," session)
print "Error setting document title"
print err_msg
exit
}

new_doc_id = dmAPIGet("checkinapp," session "," doc_id ",,'APPROVED,PROCEED' \
,,,Interleaf Compound,Interleaf App")
if (new_doc_id == NULL) {
err_msg = dmAPIGet("getmessage," session)
print "Error checking in changes to document '" doc_id "'"
print err_msg
exit
```

```
}
```

# Checkout

## Purpose

Places a lock on an object from the repository.

## Syntax

```
dmAPIGet("checkout,session,object_id[,version_label]
[,compound_arch_value][,special_app_value]")
```

## Arguments

**Table 2-36. Checkout method arguments**

| Argument | Description |
|---|---|
| *session* | Identifies an open repository session. |
| *object_id* | Identifies the object that you want to check out. Use the object's ID or an indirect reference (*@object_id*) that points to the object. |
| *version_label* | Identifies the version of the object. This can be an implicit or symbolic version label. Refer to the Usage notes for a description of the use of this argument. |
| *compound_arch _value* | Sets the a_compound_architecture attribute of the object you are checking out to the specified value. Refer to Application-specific arguments, page 147 for an extended discussion of this argument. |
| *special_app_value* | Sets the a_special_app attribute of the object you are checking out to the specified value. Refer to Application-specific arguments, page 147 for an extended discussion of this argument. |

## Return value

The Checkout method returns the ID of the specified object.

## Usage notes

You must have at least Version permission on the specified object to use the Checkout method.

If you specify a version label, the server searches the version tree that contains the specified object to find and check out the version that carries the specified version label. If the server does not find a version with the specified label, the method does not succeed.

**Note:** If you are checking out a remote object, including a version label in the method overrides any version label binding in the reference link.

Executing the Checkout method sets the r_lock_machine attribute for the checked out object. This attribute records the machine name of the client machine on which the user is working.

### Application-specific arguments

The *compound_arch_value* and *special_app_value* arguments set the object's a_compound_architecture and a_special_app attributes, respectively. These attributes are intended for internal and user-defined application use.

The a_compound_architecture attribute is used internally by the Virtual Document Manager (VDM) to determine if the object can be structurally changed in the VDM. If an object's a_compound_architecture attribute is set to any value other than an empty string, VDM assumes the object is read only and does not allow any structural changes in the object. The a_compound_architecture attribute is a string-valued attribute with a length of 16.

The a_special_app attribute is used by the server and Documentum clients to determine if the object is currently reserved for use by another application such as Lotus Notes or Interleaf. If this is any value other than an empty string, the system assumes that another application currently has control of the object. The a_special_app attribute is a string-valued attribute with a length of 32.

User-defined applications can use these attributes by setting them on check out and check in or during an unlock operation.

## Related methods

Branch, page 131
Checkin, page 138
Checkinapp, page 141
Unlock, page 477

## Example

This example performs the following operations:

- Checks out the document
- Changes the title
- Checks in the document

```
err_flag = dmAPIGet("checkout," session "," doc_id)
if (err_flag == 0) {
err_msg = dmAPIGet("getmessage," session)
print "Error checking out the document"
print err_msg
exit
}

err_flag = dmAPIExec("set," session "," doc_id ",title"\
"Reference Manual - Chapter One" )
if (err_flag == 0) {
err_msg = dmAPIGet("getmessage," session)
print "Error setting document title"
print err_msg
exit
}

new_doc_id = dmAPIGet("checkin," session "," obj_id)
if (new_doc_id == NULL) {
err_msg = dmAPIGet("getmessage," session)
print "Error checking in changes to document '" doc_id "'"
print err_msg
exit
}
```

# Close

## Purpose

Closes a collection.

## Syntax

```
dmAPIExec("close,session,collection_identifier")
```

## Arguments

**Table 2-37. Close method arguments**

| Argument | Description |
|---|---|
| *session* | Identifies an open repository session. |
| *collection_identifier* | Identifies the collection you want to close. The collection identifier is returned by any method that returns a collection of objects, for example, the Query or Readquery method. |

## Return value

The Close method returns TRUE if successful or FALSE if unsuccessful.

## Usage notes

A collection is a non-persistent object that holds a collection of objects. These objects can represent the results of a DQL query or a collection of objects representing items queued to an inbox.

The methods such as Query or Getevents that generate a collection return a value that identifies the collection holding the results of the query or the inbox items. After you have processed the query results or finished handling the items in your inbox, you close the collection, using the value returned by the method as the *collection_identifier* argument to identify the collection that you want to close.

Note that all collections are automatically closed when you disconnect a session.

## Related methods

Getevents, page 239
Next, page 316
Query, page 337

## Example

This example performs the following operations:

- Queries the document base
- Loops through the resulting collection, printing each respondent
- Closes the query collection

```
query_id = dmAPIGet("query," session ",\
select r_object_id from dm_document")
if (query_id == NULL) {
err_msg = dmAPIGet("getmessage," session)
print "Error querying the repository"
print err_msg
exit
}

resp_cntr = 0
while( dmAPIExec("next," session "," query_id) > 0 ) {
obj_id = dmAPIGet("get," session "," query_id ",r_object_id")
if (obj_id == NULL) {
err_msg = dmAPIGet("getmessage," session)
print "Error retrieving object ID from the query collection"
print err_msg
exit
}
print ++resp_cntr " : " obj_id
}

err_flag = dmAPIExec("close," session "," query_id)
if (err_flag == 0) {
err_msg = dmAPIGet("getmessage," session)
print "Error closing the query collection"
print err_msg
exit
}
```

# Commit

## Purpose

Commits an explicit database transaction.

## Syntax

```
dmAPIExec("commit,session")
```

## Arguments

**Table 2-38. Commit method arguments**

| Argument | Description |
|---|---|
| *session* | Identifies an open repository session. |

## Return value

The Commit method returns TRUE if successful or FALSE if unsuccessful.

(The Commit method will return FALSE if the underlying database (RDBMS) fails to commit the transaction for any reason.)

## Usage notes

Use the Commit method to permanently save any changes that are made during an explicit transaction. An explicit transaction is a database transaction that was opened by executing the Begintran method. When an explicit transaction is open, all changes saved or checked in to the repository are not permanent until a Commit method is executed.

A Commit method also closes the transaction.

## Related methods

Abort, page 40
Begintran, page 127

**Note:** DQL has synonymous statements: BEGIN TRAN, ABORT, and COMMIT.

## Example

```
status = dmexec("begintran,s0")
doc_id = dmget("create,s0,dm_document");
if (doc_id == NULL)
{err_mess = dmget("getmessage,s0")
print err_mess
exit
}

status = dmset("set,s0,last,object_name", "my_document")
status = dmexec("save,s0,last")
if (status != 1)
{err_mess = dmget("getmessage,s0")
print err_mess
exit
}

status = dmexec("commit,s0")
if (status != 1)
{err_mess = dmget("getmessage,s0")
print err_mess
exit
}
```

# Complete

## Purpose

Marks a work item as finished.

## Syntax

```
dmAPIExec("complete,session,workitem_id[,[return_value]
,[os_error],[result_id][,user_time][,user_cost]")
```

## Arguments

**Table 2-39. Complete method arguments**

| Argument | Description |
|---|---|
| *session* | Identifies an open repository session. |
| *workitem_id* | Identifies the work item to be completed. Use the object ID of the work item or an indirect reference (*@object_id*) the points to the work item. |
| *return_value* | An integer value to store in the work item's return_value attribute. |
| *os_error* | A string of up to 255 characters in length to store in the work item's r_exec_os_error attribute. |
| *result_id* | An object ID to store in the work item's result_id attribute. |
| *user_time* | Amount of time spent by user to complete the task. The value must be an integer value. |
| *user_cost* | Actual user cost spent to complete the task. The value must be an integer value. |

## Return value

The Complete method returns TRUE if successful or FALSE if unsuccessful.

## Usage notes

Only the performer, the workflow supervisor or a user with Sysadmin or Superuser privileges can use the Complete method to mark a work item as finished. The work item must be in the acquired state.

If the activity that generated the work item requires sign-off, then the Signoff method must be successfully executed for the work item before calling Complete. (An activity requires a signoff is its sign_off_required attribute is set to T.) The Complete method fails if a required sign-off is not obtained before calling Complete.

The Complete method updates the number of completed work items in the work items's associated activity run-time instance by incrementing r_complete_witem and updates the last performer in the associated activity run-time instance by setting r_last_performer.

## Related methods

Acquire, page 42
Delegate, page 174

## Example

```
status = dmAPIExec("complete,s0," rid)
  if (status != 1)
    Print  "Error: The work item was not acquired"
Print "The activity was completed"
```

# Connect

## Purpose

Establishes a session with a server.

## Syntax

```
dmAPIGet("connect,repository,user_auth_name,password|ticket
[,domain][,user_arg][,secure_mode]
[,application_token][,application_id][,force_authenticate]")
```

## Arguments

**Table 2-40. Connect method arguments**

| Argument | Description |
|---|---|
| *repository* | Identifies the repository with which you want to open a session. The repository specification must have one of the following formats:<br><br>*repository_name*<br><br>*repository.content_server_name*<br><br>*repository_name@host_name*<br><br>*repository.content_server_name@host_name*<br><br>where<br><br>*repository* is the name or ID of the repository. *content_server_name* is the name of the server you want to use. *host_name* is the name of the machine on which the repository resides.<br><br>The repository argument value cannot be longer than 120 characters. |
| *user_auth_name* | Identifies the current user. Specify the value in the user's user_os_name attribute.<br><br>Do not supply this argument when attempting a Windows unified logon. |

| Argument | Description |
|---|---|
| *password* | Specifies the user's password. Valid password formats are:<br>`password`<br>`DM_PLUGIN=plugin_id/password`<br>where *plugin_id* is the identifier for an authentication plugin.<br>If the password is encrypted, the format for the password argument is:<br>DM_ENCR_PASS=*encrypted_password*<br>where *encrypted_password* is one of<br>`password`<br>`DM_PLUGIN=plugin_id/password`<br>Refer to the Usage notes for details.<br>Do not supply this argument if you are using a login ticket or attempting a Windows unified logon.<br>A password may not exceed 8,192 characters in length. |
| *ticket* | A login ticket value.<br>Do not supply this argument if you are supplying a password or when attempting a Windows unified logon. |
| *domain* | Specifies the domain for the user_auth_name and password.<br>Do not supply this argument when attempting a Windows unified logon. |
| *user_arg* | Character string argument that can be used to send a value to a user-written user-validation program.<br>Do not supply this argument when attempting a Windows unified logon. |

| Argument | Description |
|---|---|
| *secure_mode* | Defines whether to establish a secure or native (unsecure) connection. Valid values are:<br>• secure<br>• native<br>• try_secure_first<br>• try_native_first<br><br>If undefined, the value defined in the secure_connect_default key in the dmcl.ini file is used. (The default in the dmcl.ini key is native.)<br><br>Requesting a connection mode, page 161 describes the argument and its interaction with the server's connection mode setting. |
| *application_token* | An application access control (AAC) token that allows access to the specified repository.<br><br>If the token includes an application identifier, you must include the *application_id* argument in the method also.<br><br>For more information about how AAC tokens are used, refer to the Usage notes. |
| *application_id* | A user-defined string that identifies the application.<br><br>Do not include this argument unless an AAC token with an encoded application identifier is also included in the arguments. In such cases, the string specified for *application_id* must match the application identifier encoded in the AAC token. |
| *force_authenticate* | A Boolean flag that controls whether an authenticate method is issued when a user uses a connection from the connection pool that he or she had previously used.<br><br>T means that an authenticate is issued. F means that the user is not re-authenticated. The default is F.<br><br>This argument is only effective if connection pooling is enabled.<br><br>For more details, refer to The force_authenticate argument, page 159. |

## Return value

The Connect method returns the session identifier for the new session.

## Usage notes

The Connect method establishes a session with a server on the specified repository.

If connection pooling is not enabled for the API session, the method opens a new repository session for the specified user.

When connection pooling is enabled, the method first checks the connection pool to determine if there is a free connection to the repository in the pool. If there is a free connection and it was last owned by the user currently requesting access, then the method returns the connection to the user. If the free connection was last owned by another user, the method first authenticates the current requesting user. The connection is returned to the user only if the authentication succeeds. If the authentication fails, the method fails. If there is no free connection to the repository in the connection pool, the method opens a new session, and registers that session with the connection pool.

### Identifying the repository and server

The variety of formats for the *repository* argument gives you several options for making the connection. If you specify only the repository name or ID, the connection broker will choose which server your session will use. You can make that choice yourself by specifying a server name in addition to the repository name or ID. For example:

```
dmAPIGet("connect,testresults.server3,myname,mypassword,domain")
```

Alternatively, you can specify a host name in addition to the repository name or ID. For example, assuming that dingo is a host machine name, the following statement opens a connection with the testresults repository server residing on dingo:

```
dmAPIGet("connect,testresults@dingo,myname,mypassword,domain")
```

In a distributed repository environment, this allows you to choose which distributed repository node you want to access.

Finally, you can control all aspects of the connection by specifying the repository name or ID, the server name, and the host name. For example:

```
dmAPIGet("connect,testresults.server3@dingo,
     myname,mypassword,domain")
```

Whichever format you use, the *repository* argument cannot contain more than 120 characters.

### Windows unified logon

If the system is running in Windows unified logon mode, specify only the repository argument. Do not include the username, password or ticket, or domain. The syntax is:

```
dmAPIGet("connect,repository")
```

### Authenticating the user

Content Server uses the values in the *user_auth_name* and *password* arguments to authenticate the user. If the authentication is to be performed by an authentication plug-in module and the plug-in is not specified in the user's user_source attribute, you must prefix the password with the plug-in module's identifier. (The identifier is defined by the plugin module.) The format of a password with a plugin identifier is:

`DM_PLUGIN=plugin_id/password`

The plug-in identifier tells Content Server which authentication plug-in to use to authenticate the user. If Content Server cannot find the module, the method fails.

If a plug-in module is not identified in the arguments, Content Server authenticates the user by the mechanism defined in the user_source attribute of the user's dm_user object.

- If the attribute indicates that the user is authenticated against the operating system using the default mechanism, the user is authenticated using the default mechanism.
- If the attribute is set to LDAP, Content Server authenticates the user against the LDAP server.
- If the attribute is set to a plug-in identifier, Content Server authenticates the user using the plug-in module. If Content Server cannot locate the plug-in, the server authenticates the user using the platform's default mechanism.

### The force_authenticate argument

If connection pooling is enabled, a user requesting a connection is assigned a free connection from the pool. By default, if the user was the last user of that connection, the user is not re-authenticated when re-using the connection. The user is only authenticated if he or she is assigned a connection previously used by another user. If you want the user to be authenticated even if he or she was the previous owner of the connection, set the force_authenticate argument to T.

### Using encrypted passwords

If you are using encrypted passwords, the *password* argument must specified as

`DM_ENCR_PASS=encrypted_password`

*encrypted_password* is the encrypted form of either of the valid password formats. The password must have been encrypted using the Encryptpass method.

When the DMCL receives the method, it decrypts the string using the AEK. The AEK must be initialized within the API session before the DMCL can execute the method. The recommended way to initialize the AEK is to use an Initcrypto method. If you do not use an Initcrypto method, the Connect method looks for the AEK in the location identified in the environment variable DM_CRYPTO_FILE. If the AEK location is not identified in DM_CRYPTO_FILE, the method assumes it is in %DOCUMENTUM%\dba\secure\aek.key ($DOCUMENTUM/dba/secure/aek.key). When the key is found, the method initializes the AEK before proceding.

### Commas in user_auth_name and password

The value in the user_auth_name or password argument can contain commas. However, you must enclose the argument in single quotes if the value contains a comma. For example, suppose the user JaneDoe in the engr domain has the password “violet,eyes”. The Connect method would be:

```
dmAPIExec("connect,devrepository,JaneDoe,'violet,eyes',engr")
```

### Using login tickets

A login ticket is a value that is provided in place of a user’s password in a Connect method. Login tickets are typically used when an application wants to open another connection with either the current repository or another respository.

For complete information about login tickets and how they can be used, refer to Login tickets, page 35, in *Content Server Fundamentals*. Login tickets are generated using a Getlogin method.

### Using application access control tokens

An application access control (AAC) token may be required by the server receiving the connection request. The tokens are encoded strings that describe who can access the repository through that server. The constraints encoded into a token may limit access to a particular user or group or to a particular application or from a particular host.

The connection request must satisfy the constraints in the token. If it does not, the connection request fails. For example, if the token specifies that the request must come from a particular client host, then the Connect method must be issued from that client host machine. Or, if the token requires a particular user to issue the request, then the Connect method must be issued by that user.

If the token constraints limit access to a particular application, you must include the *application_id* argument in the Connect method. The value specified for the argument must match the application ID encoded into the AAC token.

The application issuing the Connect method can generate the token at runtime, using the getApplicationToken DFC method (defined in the IDfSession interface). Alternatively, the client library can be enabled to retrieve and append a stored token to the method. For information about this capability and how to enable it, refer to Enabling token retrieval by the client library, page 452, in the *Content Server Administrator’s Guide*. For more information about AAC tokens, how they work, and how to generate them, refer to Application access control tokens, page 39, in *Content Server Fundamentals*.

### The user_arg argument

The *user_arg* argument lets you pass a value to a user-defined user validation program. If the application is run in an environment that uses the default user validation program provided with Content Server, that program will ignore any value in the *user_arg* argument.

### Trusted logins

When the installation owner is installing Content Server or using methods that depend on connecting with trusted login, the list of domains that can be used includes the domain of the running server process.

### Requesting a connection mode

If you are connecting to a trusted Content Server, you have the option of requesting either a native (unsecure) or secure connection. Secure connections use a Secure Socket Layer (SSL) for communications between the server and client.

Clients have a default connection mode. This default is set in the dmcl.ini file used by the clients, in the secure_connect_default key. Setting the secure_mode argument in the Connect method overrides the setting in the dmcl.ini file.

How the Connect method behaves when you include the secure_mode argument depends not only on the argument's setting but also on the server's secure_connect_mode setting.

The secure_connect_mode is a attribute in the server's server config object. Its setting determines whether the server listens on an unsecure port, a unsecure port, or both for client requests. The valid settings are: native, secure, or dual. The dual setting indicates that the server is listening on both a secure and a native port. Table 2–41, page 161 lists the possible combinations of client and server settings and the resulting Connect behavior.

**Table 2-41. Interaction of secure connection settings in Connect method**

| Client security mode request | Server sSetting for secure_connect_mode | Connect behavior |
|---|---|---|
| secure | native | The method fails because the server is listening only on the native port and the client is requesting the secure port. |
| | secure | The method succeeds if a secure connection can be established. |
| | dual | The method succeeds if a secure connection can be established. |

| Client security mode request | Server sSetting for secure_connect_mode | Connect behavior |
|---|---|---|
| native | native | The method succeeds if a native connection can be established. |
| | secure | The method fails because the server is listening only on the secure port and the client is requesting the native connection. |
| | dual | The method succeeds if a native connection can be established. |
| try_secure_first | native | The method succeeds if a native connection can be established. |
| | secure | The method succeeds if a secure connection can be established. |
| | dual | The method succeeds if either a secure or native connection can be established. The method attempts to first to establish a secure connection. If that fails, it attempts to establish a native connection. If neither succeed, the method fails. |
| try_native_first | native | The method succeeds if a native connection can be established. |
| | secure | The method succeeds if a secure connection can be established. |
| | dual | The method succeeds if either a secure or native connection can be established. The method attempts to first to establish a native connection. If that fails, it attempts to establish a secure connection. If neither succeed, the method fails. |

### Opening multiple sessions

An application or user can issue multiple Connect methods to open multiple concurrent sessions. The maximum number of sessions than can be open concurrently is controlled by the max_session_count key in the dmcl.ini file. This key is set to 10 by default.

## Related methods

Assume, page 106
Authenticate, page 123
Disconnect, page 192
Getlogin, page 247
Initcrypto, page 269

IDfSession.getApplicationToken()

## Example

This example performs the following operations:

- Connects to the server
- Activates tracing at level 1
- Disconnects from the server

```
cmd_str =
session = dmAPIGet("connect," repository "," user "," passwd)
if (session == NULL) {
print "Error connecting to repository '" repository "'"
err_msg = dmAPIGet("getmessage," session)
print err_msg
exit
}

inf_msg = dmAPIGet("getmessage," session)
print inf_msg
err_flag = dmAPIExec("trace," session ",1")
if (err_flag == 0) {
print "Error establishing trace level '1'"
err_msg = dmAPIGet("getmessage," session)
print err_msg
exit
}

err_flag = dmAPIExec("disconnect," session)
if (err_flag == 0) {
print "Error disconnecting from repository '" repository "'"
err_msg = dmAPIGet("getmessage," session)
print err_msg
exit
}
```

# Count

## Purpose

Returns the number of attributes in an object.

## Syntax

```
dmAPIGet("count,session,object_id")
```

## Arguments

**Table 2-42. Count method arguments**

| Argument | Description |
|---|---|
| *session* | Identifies an open repository session. |
| *object_id* | Identifies the object whose attributes you want to count. Use the object's ID or an indirect reference (*@object_id*) that points to the object. |

## Return value

The Count method returns the number of attributes in the specified object.

## Example

This example performs the following operations:

- Fetches the specified object
- Determines the number of attributes
- Loops through each attribute, retrieving its name, datatype, and value and then prints them to the screen.

  **Note:** If the attribute is a repeating attribute, all of its values are retrieved and printed.

```
err_flag = dmAPIExec("fetch," session "," obj_id)
if (err_flag == 0) {
err_msg = dmAPIGet("getmessage," session)
print "Error fetching the object"
print err_msg
exit
```

```
}

attr_cnt = dmAPIGet("count," session "," obj_id)
if (attr_cnt == NULL) {
err_msg = dmAPIGet("getmessage," session)
print "Error retrieving the number of attributes"
print err_msg
exit
}

for (i=0; i<attr_cnt; i++) {
attr_name = dmAPIGet("get," session "," obj_id ",_names[" i "]")
if (attr_name == NULL) {
err_msg = dmAPIGet("getmessage," session)
print "Error retrieving the attribute name : " i
print err_msg
exit
}

d_type = dmAPIGet("datatype," session "," obj_id "," attr_name)
if (d_type == NULL) {
err_msg = dmAPIGet("getmessage," session)
print "Error retrieving datatype for attribute '" attr_name "'"
print err_msg
exit
#}

if (d_type == 0) d_type_str = "boolean"
if (d_type == 1) d_type_str = "integer"
if (d_type == 2) d_type_str = "string"
if (d_type == 3) d_type_str = "ID"
if (d_type == 4) d_type_str = "time"
if (d_type == 5) d_type_str = "float"

r_flag = dmAPIGet("repeating," session "," obj_id "," attr_name)
if (r_flag == NULL) {
err_msg = dmAPIGet("getmessage," session)
if (err_msg != NULL) {
print \
"Error retrieving repeating flag for attribute '" attr_name "'"
print err_msg
exit
}
}

if (r_flag == 0) {
value = dmAPIGet("get," session "," obj_id "," attr_name)
if (value == NULL) value = "NULL"
out_str = sprintf("%2s)%19s : %s", i,attr_name,d_type_str, value)
}
else {
cnt = dmAPIGet("values," session "," obj_id)
if (cnt == NULL) {
err_msg = dmAPIGet("getmessage," session)
print
"Error retrieving number of values for'" attr_name "'"
print err_msg
exit
}
```

```
out_str =
sprintf("%2s)%19s : ( %s value(s))", i, attr_name, d_type_str, cnt)
}
print out_str
if (r_flag != 0 && cnt > 0) {
for (j=0; jcnt; j++) {
value=dmAPIGet("get,"session"," obj_id ","attr_name"[" j "]")
  if (value == NULL) value = "NULL"
  out_str = sprintf("%29s : %s", j, value)
  print out_str
}
}
}
```

# Create

## Purpose

Creates an object of the specified type.

## Syntax

```
dmAPIGet("create,session,object_type")
```

## Arguments

**Table 2-43. Create method arguments**

| Argument | Description |
|---|---|
| *session* | Identifies an open repository session. |
| *object_type* | Identifies the type of the object that you want to create. Use the name of the type, for example, dm_document. |

## Return value

The Create method returns the ID of the new object.

## Usage notes

To create a new cabinet, you must have the Create Cabinet user privilege.

You cannot use the Create method to create a new type; use the DQL CREATE TYPE statement for that purpose.

## Related methods

Branch, page 131
Checkin, page 138
Destroy, page 186
Saveasnew, page 409

## Example

This example:

- Creates a document object
- Associates content of the desired file with the document
- Saves the document object

```
doc_id = dmAPIGet("create," session ",dm_document")
if (doc_id == NULL) {
err_msg = dmAPIGet("getmessage," session)
print "Error creating a document object"
print err_msg
exit
}

the_file = "chap_1.wp5"
err_flag = dmAPIExec("setfile," session "," doc_id "," the_file ",wp5")
if (err_flag == 0) {
err_msg = dmAPIGet("getmessage," session)
print "Error setting file contents"
print err_msg
exit
}

err_flag = dmAPIExec("save," session "," doc_id)
if (err_flag == 0) {
err_msg = dmAPIGet("getmessage," session)
print "Error saving the document object"
print err_msg
exit
}
```

# Createaudit

## Purpose

Creates an audit trail entry for application events.

## Syntax

```
dmAPIGet("createaudit,session,audited_obj_id,event_name
[,string_1][,string_2][,string_3][,string_4]
[,string_5][,id_1][,id_2][,id_3][,id_4][,id_5]")
```

## Arguments

**Table 2-44. Createaudit method arguments**

| Argument | Description |
|---|---|
| *session* | Identifies an open repository session. |
| *audited_obj_id* | Identifies the object of the audited event. Use the object's object ID. |
| *event_name* | Identifies the audited event. Use the event's name. |
| *string_1 - string_5* | Additional arguments used to set the user-defined string arguments in the audit trail entry. Values in these arguments must be character strings. |
| *id_1- id_5* | Additional arguments used to set the user-defined ID arguments in the audit trail entry. Values in these arguments must be object IDs. |

## Return value

Createaudit returns the object ID of the audit trail entry.

## Usage notes

Use Createaudit in an application to create an audit trail entry of type dm_audittrail for application events. Using Createaudit to create a dm_audittrail object automatically sets many of the attributes in the object and saves the object when the method completes. (If

you used Create to create the audittrail object, you have to set each attribute individually and issue an explicit Save.)

Createaudit does not require a Save method to save the audittrail entry. However, if you issue Createaudit in an explicit transaction, the audit trail object is not created until the transaction is explicitly committed.

Anyone can use Createaudit. No special permissions or privileges are needed.

## Related methods

None

## Example

```
dmAPIGet("createaudit,S0,0900000173e3ac12,
     draft_approve,johndoe,HRDept")
```

# Datatype

## Purpose

Returns the datatype of an attribute.

## Syntax

```
dmAPIGet("datatype,session,object_id,attribute")
```

## Arguments

**Table 2-45. Datatype method arguments**

| Argument | Description |
|---|---|
| *session* | Identifies an open repository session. |
| *object_id* | Identifies the object that contains the specified attribute. Use the object's ID or an indirect reference (*@object_id*) that points to the object. |
| *attribute* | Identifies the attribute whose datatype you want to determine. Use the attribute's name. |

## Return value

The Datatype method returns a number corresponding to the datatype of the specified attribute:

| | |
|---|---|
| 0 | Boolean |
| 1 | Integer |
| 2 | String |
| 3 | ID |
| 4 | Time or date |
| 5 | Double |
| 6 | Undefined |

## Example

This example performs the following operations:

- Fetches the specified object
- Determines the number of attributes
- Loops through each attribute, retrieving its name, datatype, and value and then prints them to the screen.

  **Note:** If the attribute is a repeating attribute, all of its values are retrieved and printed.

```
err_flag = dmAPIExec("fetch," session "," obj_id)
if (err_flag == 0) {
err_msg = dmAPIGet("getmessage," session)
print "Error fetching the object"
print err_msg
exit
}

attr_cnt = dmAPIGet("count," session "," doc_id)
if (attr_cnt == NULL) {
err_msg = dmAPIGet("getmessage," session)
print "Error retrieving the number of attributes"
print err_msg
exit
}

for (i=0; i<attr_cnt; i++) {
attr_name = dmAPIGet("get," session "," obj_id ",_names[" i "]")
if (attr_name == NULL) {
err_msg = dmAPIGet("getmessage," session)
print "Error retrieving the attribute name : " i
print err_msg
exit
}

d_type = dmAPIGet("datatype," session "," obj_id "," attr_name)
if (d_type == NULL) {
err_msg = dmAPIGet("getmessage," session)
print "Error retrieving the datatype for '" attr_name "'"
print err_msg
exit
}

if (d_type == 0) d_type_str = "boolean"
if (d_type == 1) d_type_str = "integer"
if (d_type == 2) d_type_str = "string"
if (d_type == 3) d_type_str = "ID"
if (d_type == 4) d_type_str = "time"
if (d_type == 5) d_type_str = "float"

r_flag = dmAPIGet("repeating," session "," obj_id "," attr_name)
if (r_flag == NULL) {
err_msg = dmAPIGet("getmessage," session)
if (err_msg != NULL) {
print
"Error retrieving the repeating flag for '" attr_name "'"
```

```
print err_msg
exit
}
}

if (r_flag == 0) {
value = dmAPIGet("get," session "," obj_id "," attr_name)
if (value == NULL) value = "NULL"
out_str = sprintf("%2s)%19s : %s", i, attr_name, d_type_str, value)
}
else {
cnt = dmAPIGet("values," session "," obj_id)
if (cnt == NULL) {
err_msg = dmAPIGet("getmessage," session)
print
"Error retrieving the number of values for '" attr_name "'"
print err_msg
exit
}

out_str =
sprintf("%2s)%19s : ( %s value(s))", i, attr_name, d_type_str, cnt)
}
print out_str
if (r_flag != 0 && cnt > 0) {
for (j=0; jcnt; j++) {
value = dmAPIGet("get,"session","obj_id","attr_name"[" j "]")
if (value == NULL) value = "NULL"
out_str = sprintf("%29s : %s", j, value)
print out_str
}
}
}
```

# Delegate

## Purpose

Reassigns a work item.

## Syntax

```
dmAPIExec("delegate,session,workitem_id,user_name")
```

## Arguments

**Table 2-46. Delegate method arguments**

| Argument | Description |
|---|---|
| *session* | Identifies an open repository session. |
| *workitem_id* | Identifies the work item that you want to delegate. Use the object ID of the work item or an indirect reference (*@object_id*) that points to the work item. |
| *user_name* | Identifies the user or group to whom to delegate the work item. |

## Return value

The Delegate method returns TRUE if successful or FALSE if unsuccessful.

## Usage notes

The Delegate method reassigns a work item to a user or group and changes the work item's state to dormant. If the work item is reassigned to a group, only one member of the group can acquire the work item.

Under default workflow security, only a work item's performer, the workflow supervisor, or a user with Sysadmin or Superuser privileges can execute the Delegate method. If the performer is issuing the Delegate method, the activity definition that generated the work item must allow delegation and the work item must be in the acquired state. If an activity definition does not allow delegation, only the workflow supervisor or a user with Sysadmin or Superuser privileges can delegate work items generated by the activity.

However, if the workflow_security_disabled key in the server.ini file is set to T (TRUE), default workflow security is turn off and any user can issue a Delegate

method to delegate any work item. (For more information about this key, refer to The enable_workitem_mgmt key, page 234, in *Content Server Fundamentals*.)

If the user identified in user_name is not available, the server sends a warning message to the workflow supervisor.

## Related methods

Acquire, page 42
Addpackage, page 56
Complete, page 153

## Example

```
status = dmAPIExec("delegate," sess "," r_id ",user6")
    if status != 1
        Print "Error: The work item was not delegated"
```

# Demote

## Purpose

Demotes a SysObject from its current normal lifecycle state.

## Syntax

For unscheduled demotions:

```
dmAPIExec("demote,session,sysobject_id[,to_state|to_position
,return_to_base]")
```

For scheduled demotions:

```
dmAPIExec("demote,session,sysobject_id
,scheduled_date,[pattern],to_state|to_position|cancel[,return_to_base]")
```

## Arguments

**Table 2-47. Demote method arguments**

| Argument | Description |
|---|---|
| *session* | Identifies an open repository session. |
| *sysobject_id* | Identifies the SysObject to demote. Use the object's object ID. If you identify a replica object or use an indirect reference (*@objectid*), the source object is affected, not the local copy. |
| *to_state* | Is the state to which to demote the SysObject. It is required if you specify a the scheduled_date. If specified, the SysObject's current state must be the previous state in the associated policy definition; otherwise, the (scheduled or non-scheduled) demote fails. |
| *to_position* | Is the position of the state to which to demote the SysObject. If specified, the SysObject's current state must be the previous state in the associated policy definition; otherwise, the (scheduled or non-scheduled) demote fails. |
| *return_to_base* | TRUE or FALSE. When set to TRUE, demotes the object to the base state instead of the previous normal state. Otherwise, demotes the object to the previous normal state. |

| Argument | Description |
|---|---|
| *scheduled_date* | The date and time that this demotion is scheduled for. |
| *pattern* | The date-time format for *scheduled_date.* The pattern is defaulted to the client-side localized date-time format if not specified. |

## Return value

The Demote method returns TRUE if successful or FALSE if unsuccessful.

## Usage notes

If you are the object's owner or a superuser, you must have Write permission to demote an object in a lifecycle. Other users must have Write permission and Change State permission for the object. If you have only Change State permission, Content Server attempts to demote the object as the user defined in the a_bpaction_run_as attribute in the docbase config object. In those instances, that user must be either the owner or a superuser with Write permission or have Write and Change State permission on the object.

Additionally, the allow_demote flag in the current state's definition must be set to TRUE.

*to_state* and *to_position* are mutually exclusive. If you do not specify *to_state* or *to_position*, the state defaults to the previous normal state.

*return_to_base* indicates whether to demote the SysObject to the base state. If you specify both *return_to_base* and *to_state* or *to_position*, then the position or state you specify must be the base state.

If the current state is a normal state, the destination state must be the previous normal state. If the current state is an exception state, the destination state must be the base state.

When an object is demoted, the entry criteria for the destination state are not evaluated. However, the actions defined for the destination state are executed.

### Scheduling demotion

If you specify a scheduled_date and do not specify cancel, the Demote method adds a new job object whose method_name is bp_transition. The object name for the job has the following format:

Bp*sysobj_id scheduled date*

The scheduled_date is in the *yyyymmddhhmiss* format where

- *yyyy* is a 4-digit year
- *mm* is a 2-digit month

- *dd* is a 2-digit day
- *hh* is a 2-digit hour
- *mi* is a 2-digit minute
- *ss* is a 2-digit second

You cannot schedule the same SysObject for more than one state transition event at a specified time.

If you specify cancel and a *scheduled_date*, the Delete method cancels the scheduled delete by deleting the job whose name is Bp*sysobj_id scheduled date*.

## Related methods

Promote, page 323

## Example

The following example demotes the object to state number 5 in the attached lifecycle:

```
status = dmAPIExec("demote," session ",0900007b80000519,
6/4/99 10:40:16 AM,,5")
```

The following example demotes the object to the state named state1 in the attached lifecycle:

```
status = dmAPIExec("demote," session ",0900007b80000519,state1")
```

The following example cancels the demotion that was scheduled for June 4, 1999:

```
status = dmAPIExec("demote," session ",0900007b80000519,
6/4/99 10:40:16 AM,,cancel")
```

# Dequeue

## Purpose

Removes items from an inbox that were placed there using the Queue method.

## Syntax

```
dmAPIExec("dequeue,session,stamp")
```

## Arguments

**Table 2-48. Dequeue method arguments**

| Argument | Description |
|---|---|
| *session* | Identifies an open repository session. |
| *stamp* | The object ID of the dmi_queue_item object associated with the item that you want to remove from the queue. (Refer to the Usage notes for information about obtaining this value.) |

## Return value

The Dequeue method returns TRUE if successful or FALSE if unsuccessful.

## Usage notes

To remove an item from an inbox, you must have Sysadmin or Superuser privileges or you must be the user to whom the item was queued. If the item was sent to a group, then any user in the group can execute the Dequeue method against the item.

Every item on a user's queue is stored as an object of type dmi_queue_item. When you use the Queue method to put an item on a user's queue, the method returns the object ID of the dmi_queue_item for that queued item. If you want to remove the object from the user's queue, you must provide that object ID as the stamp argument to the Dequeue method.

You can obtain the object ID of a dmi_queue_item object using the Getevents method and retrieving the value in the stamp attribute from the results.You can also retrieve the object ID by querying the stamp attribute in the dm_queue view. The stamp attribute contains the object ID of the dmi_queue_item.

When you dequeue an item, the server sets three attributes of the queue item object representing the item:

- dequeued_by

  This attribute contains the name of the user who dequeued the item.

- dequeued_date

  This attribute contains the date and time that the item was dequeued.

- delete_flag

  This attribute changes to TRUE to indicate that the item was dequeued.

  **Note:** An item can be implicitly dequeued by actions such as forwarding the task associated with it. An implicit dequeueing sets the dequeued_by, dequeued_date, and delete_flag attributes just as if the item was dequeued using the Dequeue method.

## Related methods

Anyevents, page 75
Queue, page 339

## Example

This example queues an item to John's queue and then immediately dequeues the item.

```
stamp = dmAPIGet("queue," session "," doc_id ",john")
if (stamp == NULL) {
print "Error queuing the document '" doc_id '" to John"
print dmAPIGet(""getmessage," session)
}
if (dmAPIExec("dequeue," session "," stamp) ==0) {
print "Error dequeuing the item'" stamp "'"
print dmAPIGet("getmessage," session)
}
```

# Dereference

## Purpose

Returns the object ID of a remote object pointed to by a mirror object or replica in the local repository.

## Syntax

```
dmAPIGet("dereference,session,object_id")
```

## Arguments

**Table 2-49. Dereference method arguments**

| Argument | Description |
|---|---|
| *session* | Identifies an open repository session. |
| *object_id* | The object ID of a mirror object or a replica in the local repository that is associated with the original object in the remote repository. |

## Return value

The Dereference method returns the object ID of the object pointed referenced by the mirror object or replica.

**Note:** If the *object_id* argument does not identify a mirror object or replica, the method returns the object ID specified in *object_id*.

## Usage notes

Mirror objects are local objects that point to an object in a remote repository. They contain a copy of the remote object's metadata. Replicas are created by object replication jobs, which replicate objects in one repository into another repository. Use the Dereference method when you want to obtain the object ID of the remote object on which the mirror object or replica is based.

## Related methods

Refresh, page 344

## Example

This example script finds all the reference links in the Marketing cabinet, dereferences each, and then calls FindVersions to find all versions of the source object represented by the mirror object.

```
Dim cmd_str as String
Dim res_str as String
Dim err_msg as String
Dim result as Integer

Sub Main()
  Dim session_id as String
  Dim query_id as String
  Dim obj_id as String
  Dim src_id as String
  Dim chron_id as String

  print "Testing scoping example:"
  print "  - Find versions of document pointed at by reference."
  print "  "

  print "Connect to repository 1."
  cmd_str = "connect,testrepository,harvey,shme"
  session_id = dmAPIGet(cmd_str)
  if (session_id = "") then
     print "Could not connect to repository 1"
     dmExit(1)
  end if

  print "Query for references in Marketing Cabinet."
  cmd_str = "query," + session_id + ",select r_object_id from dm_document
  where folder('/Marketing') and i_is_reference=1"
  query_id = dmAPIGet(cmd_str)
  if (query_id = "") then
    err_msg = dmAPIGet("getmessage," + session_id)
    print "Error querying folder."
    print err_msg
    dmExit(1)
  end if

 while (dmAPIExec("next," +session_id+ "," + query_id) > 0)
    obj_id = dmAPIGet("get," +session_id+ "," + query_id + ",r_object_id")
    print "Dereference object " + obj_id + ":"
    src_id = dmAPIGet("dereference," +session_id+ "," + obj_id)
    if (src_id <> "") then

     chron_id = dmAPIGet("get," +session_id+ "," + src_id + ",i_chronicle_id")
      call FindVersions(session_id, src_id, chron_id)
    else
      err_msg = dmAPIGet("getmessage," +session_id)
      print "Error dereferencing " + obj_id + "."
      print err_msg
```

```
    end if
  wend

End Sub

Sub FindVersions(session_id as string, src_id as string, chron_id as string)
  Dim query_id as String
  Dim version_id as String
  Dim version_label as String

 '# Set scope to source document repository
  result = dmAPISet("set," + session_id +
      ",sessionconfig,docbase_scope",src_id)


  cmd_str = "query," + session_id + ",select r_object_id,
      r_version_label from
  dm_document (all) where i_chronicle_id = '" + chron_id + "'"
  query_id = dmAPIGet(cmd_str)
  if (query_id = "") then

   err_msg = dmAPIGet("getmessage," +session_id)
    print "Error Querying for versions of " + src_id + "."
    print err_msg
    exit sub
  end if

 while (dmAPIExec("next," +session_id+ "," + query_id) > 0)
    version_id = dmAPIGet("get," +session_id+ "," +
     query_id + ",r_object_id")
    version_label = dmAPIGet("get," +session_id+ "," +
     query_id + ",r_version_label")
    print "  " + version_id + "    " + version_label
  wend

 '# Set scope back to default setting

  result = dmAPISet("set," + session_id + ",sessionconfig,
     docbase_scope", "")
End Sub
```

# Describe

## Purpose

Describes the attributes of a type or registered table.

## Syntax

```
dmAPIGet("describe,session,type,object_type")

dmAPIGet("describe,session,table,tablename")
```

## Arguments

**Table 2-50. Describe method arguments**

| Argument | Description |
|---|---|
| *session* | Identifies an open repository session. |
| type | Identifies the object being described as a type. |
| *object_type* | Specifies the type that you want to describe. Use the type's name, for example, dm_document. |
| table | Identifies the object being described as a registered table. |
| *tablename* | Specifies the registered table that you want to describe. |

## Return value

The Describe method returns the attributes and their datatypes for the specified type or table. The description of type attributes indicates whether they are single-valued or repeating and, for the string datatypes, includes their length.

## Usage notes

The description of the attributes is returned as a formatted string intended for end users. The lines in the output are terminated with line feeds.

## Related methods

Dump, page 194

## Example

This example:

- Queries the repository for type definitions
- Prints an attribute list for each type definition retrieved

```
query_id = dmAPIGet("query," session ",select name from dm_type")
if (query_id == NULL) {
err_msg = dmAPIGet("getmessage," session)
print "Error querying the repository"
print err_msg
exit
}

resp_cntr = 0
while( dmAPIExec("next," session "," query_id) > 0 ) {
name = dmAPIGet("get," session "," query_id ",name")
if (name == NULL) {
err_msg = dmAPIGet("getmessage," session)
print "Error retrieving type name from the query collection"
print err_msg
exit
}

desc = dmAPIGet("describe," session ",type," name)
if (desc == NULL) {
err_msg = dmAPIGet("getmessage," session)
print "Error retrieving '" name "' type description"
print err_msg
exit
}
print ++resp_cntr " : " desc
}

err_flag = dmAPIExec("close," session "," query_id)
if (err_flag == 0) {
err_msg = dmAPIGet("getmessage," session)
print "Error closing the query collection"
print err_msg
exit
}
```

# Destroy

## Purpose

Removes an object from the repository.

## Syntax

```
dmAPIExec("destroy,session,object_id[,force_flag]")
```

## Arguments

**Table 2-51. Destroy method arguments**

| Argument | Description |
|---|---|
| *session* | Identifies an open repository session. |
| *object_id* | Identifies the object that you want to remove from the repository. Use the object's ID or an indirect reference (*@object_id*) that points to the object. |
| *force_flag* | Valid only when *object_id* represents an ACL or lifecycle object, this flag allows Superusers to force the removal of an ACL or lifecycle that is still in use. The default is F. Set this to T to remove an ACL or lifecycle that is still in use. |

## Return value

The Destroy method returns TRUE if successful or FALSE if unsuccessful.

## Usage notes

Using the Destroy method has the following constraints:

- You cannot destroy a locked object.
- You can only destroy a lifecycle, workflow definition or activity definition that is in draft or validated state. An installed lifecycle, workflow definition, or activity definition must be uninstalled before you can destroy it.
- You cannot destroy a workflow definition that is in use by a workflow, an activity that is referenced by a workflow definition, or a workflow that is not in the DORMANT, FINISHED, or TERMINATED state.

- You cannot destroy an object that is part of a virtual document if:
  - The virtual document is frozen, or
  - The compound_integrity attribute is set to T (This attribute is found in the server's server config object.)
- You cannot destroy a replica of a SysObject or SysObject subtype. To remove replicas of SysObjects or SysObject subtypes, use the Prune method.
- You cannot destroy a load record object while you are in an explicit transaction.
- To destroy a cabinet or folder, the object must be empty and you must have Create Cabinet privilege.
- You cannot destroy an object if it is under the control of an active retention policy or if it is stored in a content-addressed storage area and has an unexpired retention period.

  **Note:** If the object is associated with an active retention policy (dm_retainer object), you cannot remove the object using Destroy directly even if its retention period has expired. You must use a privileged delete to delete a document under the control of an active retention policy.

  If the object is in content-addressed storage, it may have multiple content addresses, each with its own retention period. If any of these are unexpired, Content Server does not allow you to destroy the object. You must use a forced delete to delete a document in these circumstances.
- To destroy audit trail entries (objects of type dm_audittrail, dm_audittrail_acl, dm_audittrail_group, or dmi_audittrail_attrs), you must have Purge Audit privileges.

  When you specify an audit trail entry as the object_id in a Destroy method, Content Server translates the method to a PURGE_AUDIT administration method call.

If the repository is running under folder security, in addition to Delete permission on the object you are destroying, you must have at least Write permission on the object's primary folder (the cabinet or folder represented in i_folder_id[0]).

This method only removes the specified object from its version tree. Consequently, if you want to remove more than one version, you must use the Prune method.

Destroying an object removes it from any folder or cabinet in which it previously appeared.

Destroying an ACL, user, or group also removes any registry objects that have the ACL, user, or group as the subject of the audit or event notification request.

### Destroying a lifecycle

You cannot destroy an installed lifecycle. You can only destroy a draft or validated lifecycle.

If there are no SysObjects associated with the lifecycle, the Destroy method removes the lifecycle. If there are sysobjects associated with the lifecycle, the Destroy method fails, unless you specify force_flag=TRUE.

The objects identified by the entry_criteria_id, user_criteria_id, action_object_id, user_action_id, type_override_id and associated with the policy are destroyed when you destroy the policy object.

#### Destroying a lifecycle with associated objects

If you have Superuser user privileges, you can use the Destroy method to remove a lifecycle which has SysObjects associated with it. Doing so, however, may cause problems when promoting or demoting those referencing objects. Be sure to modify the objects referencing the policy.

To destroy a lifecycle that has objects associated with it, issue the destroy method with the force_flag argument set to TRUE. Note that the server does not send a notification message to the owners of the SysObjects associated with the lifecycle.

### Destroying the root of a version tree

If the object is the root of a version tree, the object is not actually removed from the repository but its i_is_deleted attribute is set to TRUE and all associated objects, such as relation objects, are removed as they would be for any destroyed object.

The object itself is unlinked from any cabinets or folders in which it is found and is placed in the Temp cabinet.

If the object is carrying the CURRENT label, that label is moved to the version in the version tree that has been most recently modified, that is, to the version with the highest r_modify_date value.

### Destroying ACLs still in use

If you have Superuser user privileges, you can use the Destroy method to remove an ACL that is still referenced by one or more objects. Doing so, however, may cause problems accessing those referencing objects. Be sure to modify the objects referencing so that access to those objects is not impaired.

To remove an ACL that is still in use, issue the Destroy method with the *force_flag* argument set to TRUE.

## Related methods

Prune, page 326

## Example

This example:

- Creates a document object
- Saves the document object
- Destroys the document object

```
doc_id = dmAPIGet("create," session ",dm_document")
if (doc_id == NULL) {
err_msg = dmAPIGet("getmessage," session)
print "Error creating a document object"
print err_msg
exit
}

err_flag = dmAPIExec("save," session "," doc_id)
if (err_flag == 0) {
err_msg = dmAPIGet("getmessage," session)
print "Error saving the document object"
print err_msg
exit
}

err_flag = dmAPIExec("destroy," session "," doc_id)
if (err_flag == 0) {
err_msg = dmAPIGet("getmessage," session)
print "Error destroying the document object"
print err_msg
exit
}
```

# Disassemble

## Purpose

Destroys an assembly.

## Syntax

```
dmAPIExec("disassemble,session,document_id")
```

## Arguments

**Table 2-52. Disassembly method arguments**

| Argument | Description |
|---|---|
| *session* | Identifies an open repository session. |
| *document_id* | Identifies the virtual document that contains the assembly. Use the document's object ID or an indirect reference (*@object_id*) that points to the document. |

## Return value

The Disassemble method returns TRUE if successful or FALSE if unsuccessful.

## Usage notes

To destroy an assembly, the following conditions must be met:

- You must have at least Version permission for the document specified in *document_id*.
- The assembly must not be frozen. (To destroy a frozen assembly, you must first unfreeze it with the Unfreeze method.)

## Related methods

Assemble, page 100

## Example

This example destroys the assembly associated with the virtual document represented by the assembly_id variable.

```
status = dmexec("disassemble,s0," assembly_id)
  if (status != 1)
  {  err_mess = dmget("getmessage,s0")
     print err_mess
     exit }
```

# Disconnect

## Purpose

Terminates a repository session.

## Syntax

```
dmAPIExec("disconnect,session")
```

## Arguments

**Table 2-53. Disconnect method arguments**

| Argument | Description |
|---|---|
| *session* | Identifies the session that you want to terminate. This must be an open repository session. |

## Return value

The Disconnect method returns TRUE if successful or FALSE if unsuccessful.

## Usage notes

The Disconnect method disconnects the user from the repository session.

If connection pooling is enabled, the Disconnect method determines how many time the session has been re-used. If the number is less than the maximum number of permitted reuses, it sets the repository session's status to free in the connection pool. If the session has been re-used the maximum number of times, the method destroys the session.

If connection pooling is not enabled, the Disconnect method autmatically destroys the session.

Disconnecting closes any open collections and deletes from the server common area any files that were copied to the common area during the session.

## Related methods

Assume, page 106
Connect, page 155

## Example

This example:

- Connects to the server
- Activates tracing at level 1
- Disconnects from the server

```
session = dmAPIGet("connect," repository "," user "," passwd)
if (session == NULL) {
print "Error connecting to repository '" repository "'"
err_msg = dmAPIGet("getmessage," session)
print err_msg
exit
}

inf_msg = dmAPIGet("getmessage," session)
print inf_msg
err_flag = dmAPIExec("trace," session ",1")
if (err_flag == 0) {
print "Error establishing trace level '1'"
err_msg = dmAPIGet("getmessage," session)
print err_msg
exit
}

err_flag = dmAPIExec("disconnect," session)
if (err_flag == 0) {
print "Error disconnecting from repository '" repository "'"
err_msg = dmAPIGet("getmessage," session)
print err_msg
exit
}
```

# Dump

## Purpose

Lists an object's attributes and their values.

## Syntax

```
dmAPIGet("dump,session,object_id|type_identifier")
```

## Arguments

**Table 2-54. Dump method arguments**

| Argument | Description |
|---|---|
| *session* | Identifies an open repository session. |
| *object_id* | Identifies the object whose attributes and associated values you want to list. Use the object's ID or an indirect reference (@*object_id*) that points to the object.<br><br>Do not include this argument if you include *type_identifier*. |
| *type_identifier* | Returns data dictionary information about the specified type or attribute of a specified type. The format of this is:<br><br>`ttype_name[.attribute_name]`<br><br>where *type_name* is the name of the type, and *attribute* is the name of an attribute of that type. Use the type's formal name; for example tdm_document or tmytype.<br><br>Do not include this argument if you include *object_id*.<br><br>Refer to the Usage notes for more information about type identifiers. |

## Return value

The Dump method returns the attributes and their values as a formatted string. Each line in the output describes one attribute, and lines are separated by line feeds. The attribute descriptions are grouped into the following categories:

- User-defined

- Application-specific
- System-related attributes

## Usage notes

If you include the *object_id* argument, the method lists the attributes and their values for a specified object.

If you include the *type_identifier* argument, the method returns the data dictionary information stored for the specified object type or attribute of the object type. A type identifier is a syntax format that returns the values in either a dmi_dd_type_info or dmi_dd_attr_info object. These objects are the objects in which data dictionary information is stored internally.

If you include a type identifier in the format "t*typename*", the method returns the data dictionary information found in a dmi_dd_type_info object for the specified type. If you use the format "t*typename.attribute_name*", it returns the data dictionary information found in a dmi_dd_attr_info object for the specified attribute.

## Related methods

Describe, page 184

Type, page 464

## Example

This example:

- Fetches the specified object
- Dumps the names and values of the object's attributes

**Note:** A similar, but more flexible script, is shown in the descriptions of the Count, Datatype, Repeating, and Values methods.

```
if (dmAPIExec("fetch," session "," obj_id) == 0) {
print "Error fetching the object '" object_id "'"
print dmAPIGet("getmessage," session)
exit
}

out_str = dmAPIGet("dump," session "," obj_id)
if (out_str == NULL) {
print "Error retrieving the formatted attribute values"
print dmAPIGet("getmessage," session)
exit
}

/* print out_str */
```

# Dumpconnection

## Purpose

Provides a list of subconnections and the repositories to which they are connected.

## Syntax

```
dmAPIGet("dumpconnection[,session]")
```

## Arguments

**Table 2-55. Dumpconnection method arguments**

| Argument | Description |
|---|---|
| *session* | Identifies an open repository session. Use just the session identifier in the format S*n*. For example: S0 or S1. Do not include a subconnection identifier.<br><br>If the session is not specified, the method returns a list of subconnections and associated repositories for all sessions the current user has open with Content Server. |

## Return value

If a session is specified, the method returns a list of subconnections and the repositories to which they are connected and the current repository scope of the session.

If the session is not specified, the method returns a list of all sessions open for the user, the current repository scope of each session, the subconnections in each session, and the repositories to which the subconnections are connected.

Refer to the Example for a sample of the output of Dumpconnection.

## Usage notes

Any user can issue a Dumpconnection method. No special privileges are required.

The Dumpconnection method is intended for interactive use. It is not intended for use in applications.

## Related methods

Getconnection, page 229
Listconnection, page 300

## Example

For the following example, suppose there are two active connections: one with test_repository and one with mktg_repository. The following IAPI excerpt illustrates the output of Dumpconnection:

```
API>dumpconnection
. . .
s0: current scope is test_repository
    s0c0 test_repository
    s0c1 mktg_repository
```

# Dumploginticket

## Purpose

Dumps a login ticket or application access control token in a readable text format.

## Syntax

```
dmAPIGet("dumploginticket,session,login_ticket|aac_token")
```

## Arguments

**Table 2-56. Dumploginticket method arguments**

| Argument | Description |
|---|---|
| *session* | Identifies an open repository session. Use the session identifier in the format S*n*. |
| *login_ticket* | The login ticket you want to dump. If you include a login ticket, do not include an application access control token. |
| *aac_token* | The application access control token that you want to dump. If you include a token, do not include a login ticket. |

## Return value

The Dumploginticket method returns the values encoded into a login ticket or application access control token in a readable text format.

## Usage notes

Use Dumploginticket when troubleshooting a problem with a login ticket or application access control token. The method returns the information that is encoded into the ticket or token. Table 2–57, page 199, lists the information returned for login tickets, and Table 2–58, page 199, lists the information returned for tokens.

**Table 2-57. Values encoded into login tickets**

| Value | Description |
|---|---|
| Version | Version level of the repository in the ticket was created |
| Scope | Scope of the ticket |
| Single Use | Yes or No, depending whether the ticket was created for single or multiple use |
| Create Time | Date and time at which the ticket was created |
| Expiration Time | Date and time at which the ticket expires |
| User | Name of the user for whom the ticket was created |
| Domain | Domain of the user for whom the ticket was created, if applicable |
| Server | Name of the server that issued the method to create the ticket |
| Docbase | Name of the repository from which the ticket was generated |
| Host | Name of the host machine on which the server resides |

**Table 2-58. Values encoded into AAC tTokens**

| Value | Description |
|---|---|
| Version | Version level of the repository in which the token was created |
| Scope | Scope of the token<br><br>Valid values for this are server, docbase, and global. |
| Single Use | Yes or No, depending whether the token was created for single or multiple use |
| Create Time | Date and time at which the token was created |
| Expiration Time | Date and time at which the token expires |
| User | Name of the user for whom the token was created |
| Domain | Domain of the user for whom the token was created, if applicable |
| Server | Name of the server that issued the method to create the token |
| Docbase | Name of the repository from which the token was generated |

| Value | Description |
|---|---|
| Host | Name of the host machine on which the server resides |
| MachineID | Host machine identification. Valid values are MAC address of the host, a system ID, or anything that uniquely identifies the host. |
| ThisMachineOnly | Whether use of the token is restricted to the host identified in MachineID. T means use is restricted to that machine. F means the token may be used from any host machine. |
| ApplicationID | A user-defined string that uniquely identifies the application that can use the token. |

## Related methods

Getlogin, page 247

## Example

```
ticket=dmAPIGet("getlogin,s0,jannine,,30,T,engr_svr")
     readable_ticket=dmAPIGet("dumploginticket,s0,ticket")
```

# Encryptpass

## Purpose

Encrypts a password.

## Syntax

```
dmAPIGet("encryptpass,session,password")
```

## Arguments

**Table 2-59. Encryptpass method arguments**

| Argument | Description |
|---|---|
| *session* | Use the keyword apisession. |
| *password* | The password to be encrypted. This must be a text string. If the string includes a comma, the string must be enclosed in single quotes. |

## Return value

Encryptpass returns an encrypted password in the format:

```
DM_ENCR_PASS=encrypted_password
```

The encrypted password is also Base64 encoded.

## Usage notes

Encryptpass is used by the DMCL to encrypt repository passwords used by Content Server when connecting to repositories for operations such as object replication, federation-related jobs, surrogate get, and so forth. These passwords are stored in files specific to the operation type. The files are created when the operations are configured.

The method is also used by some Documentum clients to encrypt the repository passwords used by these products. The encrypted passwords are passed to the DMCL when the client is connecting to the repository. Custom client applications can use Encryptpass in a similar manner.

The method uses AEK (Administration Encryption Key) to encrypt a password. After encrypting the password, the method Base64 encodes the encrypted string and then prefixes the string with DM_ENCR_PASS. As a result of these operations, the encrypted password is longer in size than the unencrypted password.

The AEK must be initialized within the API session before the Encryptpass method can be executed. The recommended way to initialize the AEK is to use an Initcrypto method. If you do not use an Initcrypto method, the Encryptpass method looks for the AEK in the location identified in the environment variable DM_CRYPTO_FILE. If the AEK location is not identified in DM_CRYPTO_FILE, the method assumes it is in %DOCUMENTUM%\dba\secure\aek.key ($DOCUMENTUM/dba/secure/aek.key). When the key is found, the method initializes the AEK before proceding.

## Related methods

Initcrypto, page 269

## Example

The following example initializes the AEK and then calls Encryptpass to encrypt the password foo.

```
Status = dmAPIExec("initcrypto,apisession,C:\Documentum\ARP\Crypto")
if (Status = 0) Then
   Call DmPrint("Could not open AEK at
      C:\Documentum\ARP\Crypto")
Else
   encrypted_password$ = dmAPIGet("encryptpass,apisession,foo")
   If (encrypted_password = "") Then
      Call DmPrint("Could not encrypt password")
   End if
End if
```

This short excerpt illustrates how to quote a password that includes a comma:

```
encrypted_password$ = dmAPIGet("encryptpass,apisession,'furry,dog'")
   If (encrypted_password = "") Then
      Call DmPrint("Could not encrypt password")
   End if
End if
```

# Execquery

## Purpose

Executes a DQL statement.

## Syntax

```
dmAPIExec("execquery,session,[readquery_flag],dql_query")
```

## Arguments

**Table 2-60. Execquery method arguments**

| Argument | Description |
|---|---|
| *session* | Identifies an open repository session. |
| *readquery_flag* | Indicates that this is a read-only query. The default is F (FALSE). |
| *dql_query* | Specifies the DQL query that you want to execute. This can be any length. |

## Return value

The Execquery method returns TRUE if successful or FALSE if unsuccessful.

## Usage notes

Use the Execquery method when you want to send a long query to Content Server. A long query is defined as a query containing more than 255 characters.

Use the readquery flag for better performance when you execute a SELECT statement whose results will be processed without any database changes occurring during the processing.

To obtain the ID of the collection holding the results of an Execquery execution, use the Getlastcoll method. (Refer to Using execquery in an application, page 351 in *Content Server Fundamentals* for more information about using Execquery and processing query results.)

## Related methods

Cachequery, page 133
Execsql, page 205
Readquery, page 342

**Note:** See also the EXEC_SQL administration method, which is executed by either the Apply method or the DQL EXECUTE statement.

## Example

```
 status = dmexec("execquery,s0,T,select subject from \
dm_document where owner_name = USER")
  if (status != 1)
  {  err_mess = dmget("getmessage,s0")
     print err_mess
     exit
  }
 query_id = dmget("getlastcoll,s0")
 while(dmexec("next,s0,query_id"))
  {  subject = dmget("get,s0," query_id ",subject")
      print subject
  }
  dmexec("close,s0," query_id)
```

# Execsql

## Purpose

Executes SQL (Structured Query Language) statements.

## Syntax

```
dmAPIExec("execsql,session,sql_statement")
```

## Arguments

**Table 2-61. Execsql method arguments**

| Argument | Description |
|---|---|
| *session* | Identifies an open repository session. |
| *sql_statement* | Specifies the SQL statement that you want to execute. This can be any SQL statement except a SELECT statement. |

## Return value

The Execsql method returns TRUE if successful or FALSE if unsuccessful.

## Usage notes

You cannot use this method to execute an SQL SELECT statement.

You must have the Superuser privilege to execute this method.

## Related methods

Execquery, page 203
Readquery, page 342

**Note:** See also the EXEC_SQL administration method, which is executed by either the Apply method or the DQL EXECUTE statement.

## Example

```
sql_stmt = "create index new_index on my_table(col_1) \
commit"
 err_flag = dmAPIExec("execsql," session "," sql_stmt)
 if (err_flag == 0) {
   err_msg = dmAPIGet("getmessage," session)
   print "Error executing sql statement '" sql_stmt "'"
   print err_msg
   exit
 }
```

# Execute

## Purpose

Starts or restarts a workflow.

## Syntax

```
dmAPIExec("execute,session,workflow_id")
```

## Arguments

**Table 2-62. Execute method arguments**

| Argument | Description |
|---|---|
| *session* | Identifies an open repository session. |
| *workflow_id* | Identifies the workflow that you want to start. Use the object ID of the workflow. |

## Return value

The Execute method returns TRUE if successful or FALSE if unsuccessful.

## Usage notes

You must be the workflow creator or supervisor or have Sysadmin or Superuser privileges to use the Execute method.

The Execute method generates a run-time instance for each start activity with r_act_state set to dormant, and generates a run-time instance for each activity that requires a pre-timer and sets r_pre_timer.

## Related methods

Abort, page 40
Halt, page 265
Restart, page 386
Resume, page 392

## Example

```
status = dmAPIExec("execute," session ",4d00007b80000100")
```

# Fetch

## Purpose

Fetches an object from the repository without placing a lock on the object.

## Syntax

```
dmAPIExec("fetch,session,object_id[,type]
[,persistent_cache][,consistency_check_value]")
```

## Arguments

**Table 2-63. Fetch method arguments**

| Argument | Description |
|---|---|
| *session* | Identifies an open repository session. |
| *object_id* | Identifies the object that you want to fetch. Use the object's ID or an indirect reference (*@object_id*) that points to the object. |
| *type* | Identifies the object's type. Using this enhances performance for the Fetch method but it is not required. Use the type's internal name, for example, dm_document. |
| *persistent_cache* | Whether to put the object in the persistent client cache. T (TRUE) means to store the object in the persistent client cache. The default is F (FALSE), meaning not to store the object in the cache. |
| *consistency_check_value* | Defines how to handle consistency checking when fetching the object. Valid values are:<br>• Name of a cache config object<br>• A positive integer<br>• One of three keywords or the integer equivalents<br><br>If you identify a cache config object, the server uses the consistency check rules defined in the cache config object. Use the object name of the cache config object.<br><br>A positive integer value is interpreted in seconds and defines the length of time that the object can be used without being checked for consistency.<br><br>The keywords are: |

| Argument | Description |
|---|---|
| | **check_always**, meaning the object is always checked and refetched if needed. check_always is the equivalent of specifying 0 for the argument.<br><br>**check_never**, meaning the object is never checked. If the object is cached, the object in the cache is used. check_never is the equivalent of specifying -1 for the argument.<br><br>**check_first_access**, meaning the object is checked if it is found in the cache and has not been previously checked for currency by the client application. The object is not checked on second and subsequent accesses by the application. check_first_access is the equivalent of specifying -2 for the argument.<br><br>The default is check_always. |

## Return value

The Fetch method returns TRUE if successful or FALSE if unsuccessful.

## Usage notes

To fetch a SysObject, you must have at least Browse permission on the specified object. If you want to save the object after you fetch it, you must have at least Write permission on the object.

When you fetch a SysObject from the repository, the server does not place a lock on the object. This means there is no guarantee that you can save any changes you make because another user may check out the object while you have it fetched.

Using the Fetch method does not allow you to use the Checkin method to write the SysObject back to the repository. You must use the Save method.

Fetching audit trail entries, page 210 describes the rules for fetching audit trail entries. You cannot modify or save fetched audit trail entries.

### Fetching audit trail entries

Fetching audit trail entries is subject to the following rules:

- If you have Superuser or View_audit privileges, you can fetch any audit trail entry, including audittrail acl and audittrail group objects.
- You must have Superuser or View_audit privileges to fetch audittrail attrs objects.

- If you do not have Superuser or View_audit privileges and an entry's i_audited_obj_class attribute value is 0, to fetch the entry, you must have Browse privileges defined in the ACL identified by the acl_name and acl_domain attributes of the entry. (If there is no ACL identified in the audit trail entry or the ACL does not exist, you cannot fetch that audit trail object either.)
- If you do not have Superuser or View_audit privileges and an entry's i_audited_obj_class attribute value is -1, 1, 2, or 3, you cannot fetch that entry.
- If you are the owner of an audited object, you can fetch the audit trail entries that record the events on the audited object. No special privileges are required.
- Any user can fetch audit trail entries for application events.

### Caching objects

All fetched objects are placed in an in-memory cache on the client side. If *persistent_cache* is set to T in the Fetch method, the object is marked for persistent caching. All objects marked for persistent caching are written to the client disk when the session terminates. The cache is stored in a directory identified by the name of the user who issued the Fetch method. The cache file is loaded into memory the next time the user starts a session. Storing objects in a persistent cache can improve performance for operations that access the object and is most useful for objects that are infrequently changed. (Refer to Using persistent client caching in an application, page 48 in *Content Server Fundamentals* for an introduction to persistent client caching.)

### Refreshing fetched or cached objects

By default, an object is checked for consistency against the object in the repository every time the object is fetched. To change the default, set the *consistency_check_value* argument. The argument defines how the fetched object is checked for consistency for that Fetch call.

Consistency checking, page 49 in *Content Server Fundamentals* describes in detail how consistency checking is implemented.

## Related methods

Checkout, page 146

## Example

This example:

- Fetches the specified object
- Dumps the names and values of the object's attributes

**Note:** An example script that produces a similar result in a more flexible manner is shown for the Count, Datatype, Repeating, Value methods.

```
if (dmAPIExec("fetch," session "," obj_id) == 0) {
```

```
print "Error fetching the object '" object_id "'"
print dmAPIGet("getmessage," session)
exit
}

out_str = dmAPIGet("dump," session "," doc_id)
if (out_str == NULL) {
print "Error retrieving the formatted attribute values"
print dmAPIGet("getmessage," session)
exit
}

# print out_str
```

# Flush

## Purpose

Removes cached queries from the client's query caches; ACLs or groups from the session cache; objects from the persistent client caches; or non-persistent objects from the data dictionary cache.

## Syntax

```
dmAPIExec("flush,session,cache_keyword,[object_type][,reset_
change_bit][,include_subtypes]")
```

## Arguments

**Table 2-64. Flush method arguments**

| Argument | Description |
|---|---|
| *session* | Identifies an open repository session. |
| *cache_keyword* | Identifies the objects or cache to be flushed. Valid keywords are:<br><br>**querycache**, to flush the user's persistent query cache<br><br>**aclcache**, to flush the ACLs in the session cache<br><br>**groupcache**, to flush the groups in the session cache<br><br>**ddcache**, to flush the data dictionary objects cached on the client host<br><br>**registrycache**, to flush registry objects in the session's registry cache<br><br>**persistentcache**, to flush the objects and query results in the user's persistent client cache<br><br>**persistentobjcache**, to flush the objects in the user's persistent client cache<br><br>You can include only one cache keyword in the arguments. If you include ddcache, you must include the *cache_key* argument. |

| Argument | Description |
|---|---|
| *object_type* | Identifies an object type. Include *object_type* only when *cache_keyword* identifies either the data dictionary cache or the registry cache.<br><br>When the data dictionary cache is the target, Flush removes the data dictionary objects for the specified object type and its subtypes. When the registry cache is the target, you must include the *include_subtypes* argument to flush registry objects for subtypes of the specified object type. |
| *reset_change_bit* | T (TRUE) directs Content Server to reset the change bit in the registry cache to false for all cache entries that are marked true when the cache is flushed. Use this only when the registry cache is the target of the operation. For more information about its use, refer to Flushing the registry cache, page 214. The default is F (FALSE). |
| *include_subtypes* | Whether you want to flush registry objects that reference subtypes of the object type identified in *object_type*. The default is F (FALSE), meaning not to flush objects referencing subtypes. |

## Return value

The Flush method returns TRUE if successful and FALSE if unsuccessful.

## Usage notes

Use the Flush method to

- Remove cached registry objects
- Persistent client cache files
- Free up server memory by removing cached ACLs or groups
- Remove cached data dictionary objects

### Flushing the registry cache

When a Content Server is started, the server creates an in-memory cache of all the registry objects currently in the repository. Each session started with a server creates a copy of that server's registry cache. When you flush a registry cache, the operation affects your session copy and the root server's cache. The operation does not affect the session-level registry caches in use by other current sessions.

The information in the registry cache includes a change bit for each object type referenced by a registry object. When a Register, Unregister, Audit, or Unaudit method is executed that affects one of those object types, the change bit is set to T (TRUE). The Flush method affects all cache entries for object types whose change bit is set to T. If you include the *include_subtypes* argument set to T, the method also flushes entries for all subtypes of the affected object types. Including the *reset_change_bit* argument set to T resets the change bit to F (FALSE) for the affected object types after flushing their entries.

### Flushing the persistent client cache

Flushing a persistent cache, or any part of it, flushes the cache for the user who executes the Flush method. It is not possible to flush another user's cache.

There are three options for flushing a persistent client cache. You can flush the entire cache (objects and queries), only the objects in the cache, or only the query results. To flush the entire cache, include the persistentcache keyword. To flush only the objects, include the persistentobcache keyword. To flush only the query results, use the querycache keyword .

If you flush the query cache files, the method removes all files from the cache file in the user's client local area and recreates the cache.map file without entries. (The cache.map file maps the entries in an executed cached query to the results file.) The next time the user executes a cached query, a new result set is cached.

### Flushing the ACL or group cache

Including aclcache removes the cached ACLs from the session cache. Including groupcache removes the cached groups from the session cache. Removing cached ACLs, groups, or both can eliminate out of memory errors reported by the server if large numbers of objects were created during the session.

### Flushing the data dictionary cache

Include the ddcache keyword to remove the data dictionary objects for a specific object type and its subtypes that are cached on the client host. The *object_type* argument identifies the object type.

## Related methods

Flushcache, page 217
Purgelocal, page 332

## Examples

```
status = dmAPIExec("flush,c,querycache")
if (status == 0)
print dmAPIGet("getmessage,c")
```

```
status = dmAPIExec("flush,c,ddcache,dm_document")
```

```
if (status == 0)
print dmAPIGet("getmessage,c")
```

# Flushcache

## Purpose

Removes objects from the server and client in-memory caches. (It does not remove objects from a persistent cache. Use Flush to remove objects from a persistent cache.)

## Syntax

```
dmAPIExec("flushcache,session[,discard_changed]")
```

## Arguments

**Table 2-65. Flushcache method arguments**

| Argument | Description |
|---|---|
| *session* | Identifies an open repository session. |
| *discard_changed* | Indicates whether you want to discard objects in the caches that have been changed but not saved. The default is FALSE, which retains those items. Set it to TRUE to remove them. |

## Return value

The Flushcache method returns TRUE if successful or FALSE if unsuccessful.

## Usage notes

The Flushcache method cleans objects out of the client and server caches. These caches hold objects that the user has fetched during his or her session. If you set the *discard_changed* argument to TRUE, the system removes changed objects also. Note, however, that changed objects in the cache have not yet been saved to the repository so if you remove them, you are losing the work done in that session.

## Related methods

Flush, page 213
Purgelocal, page 332

## Example

```
status = dmAPIExec("flushcache,c")
if (status != 1)
print dmAPIGet("getmessage,s0");
```

# Flushconnectpool

## Purpose

Disconnects all free connections in the connection pool and removes them from the pool.

## Syntax

```
dmAPIExec("flushconnectpool")
```

## Arguments

Flushconnectpool has no arguments.

## Return value

Flushconnectpool returns TRUE if successful or FALSE if unsuccessful.

## Usage notes

Flushconnectpool affects only the connection pool established by the application from which the method is issued. When the method is executed, any free sessions in the pool are immediately removed from the connection pool and disconnected. Any sessions that are in use are removed and disconnected when they are released back to the connection pool.

Use this method cautiously in an application. If the application attempts to use a connection that has been removed from the connection pool and disconnected, undefined behavior may occur.

Any user can issue a Flushconnectpool method.

## Related methods

Flush, page 213
Flushcache, page 217

## Example

```
status=dmAPIExec("flushconnectpool")
if (status != 1)
print dmAPIGet("getmessage,s0");
```

# Freeze

## Purpose

Marks an object and, optionally, its components if the object is a virtual document, as unchangeable.

## Syntax

```
dmAPIExec("freeze,session,object_id[,freeze_components]")
```

## Arguments

**Table 2-66. Freeze method arguments**

| Argument | Description |
|---|---|
| *session* | Identifies an open repository session. |
| *object_id* | Specifies the object that you want to mark unchangeable.<br><br>Use the object's ID or an indirect reference (@object_id) that points to the object.<br><br>If you are freezing the assembly for a virtual document (setting freeze_components to T), *object_id* must identify the document with which the assembly is associated. Generally, this is the virtual document. However, a virtual document's assembly is sometimes associated with a simple document. In such cases, specify the simple document's object ID in this argument. |
| *freeze_components* | Indicates whether you want to freeze the components that make up the specified object's assembly as well as the object itself. Set this flag to TRUE to freeze the assembled components as well as the object. This flag is only set if the specified object is a virtual document that has an associated assembly. The default is FALSE. |

## Return value

The Freeze method returns TRUE if successful or FALSE if unsuccessful.

## Usage notes

Using the Freeze method, you can mark any SysObject as unchangeable. If the specified object is a virtual document, you also have the option to freeze the components that make up its associated assembly as well as the document itself.

When an object is frozen, you cannot change its content or its primary storage location. Additionally, many of its attributes are made unchangeable. (For a list of those that can be changed in frozen object, refer to Immutability, page 123 in *Content Server Fundamentals*.) Finally, if the object is a virtual document, you cannot add or delete components.

If you set the *freeze_components* flag, the server freezes the components that make up the assembly associated with the object. This means that you cannot modify the components nor can you delete them from the virtual document.

Freezing an object sets two attributes of the object:

- r_immutable_flag

  This attribute indicates that the object is unchangeable.

- r_frozen flag

  This attribute indicates that the object is unchangeable because it was specifically frozen.

If you set the *freeze_components* flag, the server sets the object's r_has_frzn_assembly attribute also. This attribute indicates that the document's assembly is frozen in addition to the document itself.

When you freeze an assembly, the method sets the following attributes for each component of the assembly:

- r_immutable_flag
- r_frzn_assembly_cnt

  This attribute maintains a count of the number of frozen assemblies to which this object belongs.

(For a detailed discussion of how frozen virtual documents and assemblies behave, refer to Chapter 7, Virtual Documents, in *Content Server Fundamentals*.)

## Related methods

Unfreeze, page 469

## Example

In this example, doc_id is a variable containing the object ID of the document that has the attached assembly.

```
  cmd = sprintf("freeze,c," doc_id ",T")
  status = dmexec(cmd)
  if (status != 1)
    {
      error_msg = dmget("getmessage,s0");
      printf "%s", error_msg;
    }
cmd = sprintf("save,c," doc_id)
  status = dmexec(cmd);
  if (status != 1)
    {error_msg = dmget("getmessage,s0");
      printf "%s", error_msg;
    }
```

# Get

## Purpose

Gets the value of an attribute or information about a Content Server installation.

## Syntax

```
dmAPIGet("get,session,object_id|type_identifier,attribute[[index]]
[,pattern]")
```

## Arguments

**Table 2-67. Get method arguments**

| Argument | Description |
|---|---|
| *session* | Identifies an open repository session. |
| *object_id* | Specifies the object that contains the attribute whose value you are getting. Use either the object's ID or an indirect reference (@*object_id*) that points to the object. Alternatively, you can use one of the following keywords:<br><br>apiconfig<br>sessionconfig<br>connectionconfig<br>docbaseconfig<br>serverconfig<br><br>Refer to the Usage notes for a description of the objects represented by the keywords. |
| *type_identifier* | Returns data dictionary information about the specified type or attribute of a specified type. The format of this is:<br><br>`ttype_name[.attribute_name]`<br><br>where *type_name* is the name of the type, and *attribute* is the name of an attribute of that type. Use the type's formal name; for example tdm_document or tmytype.<br><br>Do not include this argument if you include *object_id*. |

| Argument | Description |
|---|---|
| *attribute* | If you include *object_id*, this identifies the object attribute whose value you want to obtain. Use the attribute's name specified in lowercase.<br><br>If you include *type_identifier*, this identifies an attribute in either a dmi_dd_type_info object or a dmi_dd_attr_info object.<br><br>If the type_identifier references only a type name (no attribute), this value must identify an attribute in a dmi_dd_type_info object. If the type_identifier references a type name and an attribute, this value must identify an attribute in a dmi_dd_attr_info object. |
| [*index*] | For repeating attributes, specifies the position in the attribute's value list of the value you wish to get. Do not use this argument if attribute specifies a single-valued attribute. You must include the square brackets (refer to the Usage notes for an example). The default value is zero. |
| *pattern* | For date attributes, defines the format in which you want the date returned. You can specify any valid character string date format. You must enclose the value in single quotes if the pattern contains a comma. Refer to the Usage notes for a list of valid patterns for dates.<br><br>By default, the server returns date values in the short date format defined for the client. |

## Return value

The Get method returns the requested information as a character string.

## Usage notes

The Get method retrieves an attribute value from an object. Generally, you specify which object by specifying the object's ID in the method's argument list. However, the server provides special keywords for four objects. These keywords and the objects that they represent are:

- apiconfig

  This keyword represents the api config object, a non-persistent object that is constructed when the client issues a dmAPIInit call and is destroyed when the client issues the dmAPIDeInit call. This object's attributes reflect values in the client's

dmcl.ini file (unless the object's values for the session have been changed using a Set method). Using the Get method, you can determine the client's primary connection broker, its backup connection brokers, and several client parameters such as the client_cache_size, where the client's local common area is located, which dmcl.ini file the client is using, and so forth. Refer to Table 2–6, page 68 in the *EMC Documentum Object Reference Manual* for a complete list of its attributes.

- sessionconfig

  This keyword represents the session config object, a non-persistent object that is constructed when the client connects to a repository and is destroyed when the client disconnects from the repository. This object's attributes represent the operating parameters for that client's repository session. They are a merging of some attribute values from the client's api config object and some from its server config object. Refer to Session Config, page 441 in the *EMC Documentum Object Reference Manual* for a list of its attributes.

  You can also use this keyword to retrieve the value of the _isdeadlocked computed attribute for the session.

- connectionconfig

  This keyword represents a connection config object, a non-persistent object that is constructed when the client establishes a subconnection within a session. It is destroyed when the client closes the subconnection. The object's attributes represent the operating parameters for that repository connection. Refer to Table 2–34, page 140 in the *EMC Documentum Object Reference Manual* for a list of its attributes.

- docbaseconfig

  This keyword represents the docbase config object, a persistent object that is part of the repository's configuration definition. The attributes in this object control the repository's security level, the location of its events directory, and the location of the tablespace for type indexes. Refer to Table 2–46, page 181 in the *EMC Documentum Object Reference Manual* for a list of its attributes.

- serverconfig

  This keyword represents a server config object, a persistent object whose attributes define the configuration for a server. They contain information about where the server can find the directories that it needs, such as the content storage areas, the full-text index directories, and external scripts. Refer to Table 2–153, page 426 in the *EMC Documentum Object Reference Manual* for a list of its attributes.

### Obtaining repeating attribute values

To get a value from a repeating attribute, you must specify the index position of that value. The index position identifies the value's position in the list of values that make up the repeating attribute. Index position numbers begin at zero for the first value in the repeating attribute and increment by one for each succeeding value. You must

enclose the index position value in square brackets when you specify it. For example, the following statement gets the value in the third position of the authors attribute:

```
dmAPIGet("get,s0,0900023e1487620e,authors[2]")
```

### Specifying patterns for date attributes

The Get method allows you to define how you want a returned date value expressed. By including a pattern specification on the Get method command line, you tell the server how to format the returned date. Valid patterns are:

*mm*/*dd*/[*yy*]*yy*
*dd-mon*-[*yy*]*yy*
*month dd*[,] [*yy*]*yy*
*mon dd* [*yy*]*yy*
[*dd*/]*mm*/[*yy*]*yy* [*hh*:*mi*:*ss*]
[*yy*]*yy*/*mm*[/*dd*] [*hh*:*mi*:*ss*]
[*mon*-][*yy*]*yy* [*hh*:*mi*:*ss*]
*month*[,] [*yy*]*yy* [*hh*:*mi*:*ss*]

For example, the following method call returns the value of a document's r_creation_date attribute in the format *dd-mon-yy*:

```
get,s0,090000562e000021,r_creation_date,dd-mon-yy
```

If the document's creation date was November 22, 1995, the example above would return 22-Nov-95.

You must enclose the pattern specification in single quotes if the pattern contains a comma.

### Using a type_identifier

If you include the *type_identifier* argument, the method returns the data dictionary information stored for the specified object type or attribute of the object type. A type identifier is a syntax format that returns the values in either a dmi_dd_type_info or dmi_dd_attr_info object. These objects are the objects in which data dictionary information is stored internally.

If you include a type identifier in the format "t*typename*", the method returns the data dictionary information found in a dmi_dd_type_info object for the specified type. In this case, the *attribute* argument must reference an attribute of the dmi_dd_type_info type.

If you include a type identifier in the format "t*typename.attribute_name*", the method returns the data dictionary information found in a dmi_dd_attr_info object for the specified type. In this case, the *attribute* argument must reference an attribute of the dmi_dd_attr_info type.

For example, suppose you want to return only the default value assigned to mycustomattr in the dm_document type. In the data dictionary, an attribute's default value is stored in the dmi_dd_attr_info.default_value attribute. To obtain that value, use:

```
get,session,tdm_document.mycustomattr,default_value
```

## Related methods

Getmessage, page 251
Locate, page 304

## Example

This example:

- Queries the repository
- Loops through the resulting collection, printing each respondent
- Closes the query collection

```
query_id = dmAPIGet("query," session ",select r_object_id from dm_document")
if (query_id == NULL) {
err_msg = dmAPIGet("getmessage," session)
print "Error querying the repository"
print err_msg
exit
}

resp_cntr = 0
while( dmAPIExec("next," session "," query_id) > 0 ) {
obj_id = dmAPIGet("get," session "," query_id ",r_object_id")
if (obj_id == NULL) {
err_msg = dmAPIGet("getmessage," session)
print "Error retrieving object ID from the query collection"
print err_msg
exit
}
print ++resp_cntr " : " obj_id

err_flag = dmAPIExec("close," session "," query_id)
if (err_flag == 0) {
err_msg = dmAPIGet("getmessage," session)
print "Error closing the query collection"
print err_msg
exit
}
```

The following example retrieves the extended permissions of an object in string form and separated by commas.

```
dmAPIGet("get,s0,090000562e000021,_accessor_xpermit_names")
```

# Getconnection

## Purpose

Returns the subconnection identifier for a repository or establishes a subconnection to the repository if none exists.

## Syntax

```
dmAPIGet("getconnection,session,scoping_value
[,connect_flag]")
```

## Arguments

**Table 2-68. Getconnection method arguments**

| Argument | Description |
|---|---|
| *session* | Identifies an open repository session. Use a session identifier in the format S*n*. For example: S0 or S1. |
| *scoping_value* | Identifies the repository whose subconnection identifier you want to retrieve or with which you want to establish a connection. Valid values for *scoping_value* are:<br><br>A repository ID<br>A repository name<br>The object ID of an object in the repository<br>An indirect reference (*@object_id*) that points to an an object in the repository. |
| *connect_flag* | Indicates whether you want to establish a subconnection with the repository if none currently exists. This flag is TRUE by default. |

## Return value

The Getconnection method returns the subconnection identifier for the connection to the repository.

If no connections exists and the *connect_flag* is set to FALSE, the method returns a blank, or NULLSTRING.

## Usage notes

Use the Getconnection method when you want to establish a subconnection within a current session. You must be valid user in the target repository to issue a Getconnection requesting a connection to that repository.

Use the session identifier to define the session in which you want to establish the subconnection. Although you can specify a subconnection within the session without generating an error, Content Server will consider it the equivalent of only specifying the session identifier. That is, specifying S*n*C*n* in the *session* argument is treated as the equivalent of specifying S*n*.

### Working in environments with Trusted Content Services

The Getconnection method inherits the secure connection mode requested by the primary connection. For example, if the primary connection was established by a Connect method that requested a secure connection mode, then the Getconnection method is issued with an implicit request for a secure connection. Consequently, you must ensure that the target server is listening on a port that provides the requested connection mode.

## Related methods

Dumpconnection, page 196
Listconnection, page 300

## Example

The following example issues a Getconnection method and uses the returned subconnection identifier in a query to a remote repository:

```
subconnectID=dmAPIGet("getconnection,s0,Engineering")
cmd_str="query," subconnectID ",select owner_name from dm_document"
qry_id=dmAPIGet(cmd_str)
```

# Getcontent

## Purpose

Copies a content file in the repository into memory.

## Syntax

```
dmAPIGet("getcontent,session,document_id[,format]
[,page_num][,page_modifier][,get_resource]")
```

## Arguments

**Table 2-69. Getcontent method arguments**

| Argument | Description |
|---|---|
| *session* | Identifies an open repository session. |
| *document_id* | Identifies the document whose content you want to retrieve. Use the document's object ID or indirect reference (*@object_id*) that points to the object. |
| *format* | Identifies the file format in which you want the content. Use the format's name (the value in the name attribute of the format's format object). The default is the format identified in the document's a_content_type attribute. |
| *page_num* | Identifies the content page with which the content file is associated. The default is zero (the first content page). |
| *page_modifier* | Identifies a rendition. This is a character string defined when the rendition was created.<br><br>You must include the page modifier if there are multiple renditions in the specified format associated with the specified content page.<br><br>The default is an empty string. |
| *get_resource* | For files created on Macintosh systems, indicates whether you want the method to retrieve the data fork (the file containing the content) or the resource fork. FALSE (the default) returns the data fork (the content file). TRUE retrieves the resource fork (the file specified in the other_ticket attribute of the content object). |

## Return value

The Getcontent method returns a collection that has one result object. Table 2–70, page 232 lists the attributes in the result object.

**Table 2-70. Result object attributes for Getcontent method**

| Attribute | Description |
|---|---|
| buffer | The address of the buffer. |
| buffer_size | The size, in bytes, of the buffer (this was specified in the *buffer_size* argument). |
| current_length | The size, in bytes, of the retrieved segment. |
| content_size | The size, in bytes, of the requested content file. This attribute is correct for files up to 2GB in size. |
| full_content_size | The size, in bytes, of the requested content file.<br><br>This attribute records the full size of files over 2GB. |
| seekable | Indicates if this content is seekable. Refer to Seek, page 413 method. |
| byte_position | Indicates the position of the file pointer (Useful with Seek, page 413 method). |
| _content_buffer | Points to the location of the file in memory.<br><br>**Note:** This attribute is only accessible through the Getcontent method. The attribute is not displayed with the Dump command, but applications can access it using the Get method. |

## Usage notes

You must have at least Read permission on the object identified in *document_id* to use this method.

The Getcontent method copies a requested file in the repository into memory. This method is intended for use by applications rather than end users.

### Identifying the content file

The format, page_number, and page_modifier arguments together identify the particular content file wanted. If you want to return the primary content for the first content page, you can default all of these arguments. For example:

```
dmAPIGet("getcontent,s0,0900000258342e5c")
```

To return any other content file, whether a primary content or a rendition, you must include one or more of the arguments. For example, the following statement returns the primary content of the second page:

```
dmAPIGet("getcontent,s0,0900000258342e5c,,2")
```

The next example returns the PDF rendition associated with the second page:

```
dmAPIGet("getcontent,s0,0900000258342e5c,pdf,2")
```

Suppose that there were two PDF renditions associated with the third page. The first PDF rendition has the page modifier first and the second PDF rendition has the page modifier second. To retrieve the second PDF rendition of the third page, the method call is:

```
dmAPIGet("getcontent,s0,0900000258342e5c,pdf,3,second")
```

If you include a page modifier and a rendition with that modifier isn't found, the method returns an error.

If you don't include a page modifier, the server looks first for a content file or rendition with the specified format and no modifier. If one is not found, the server looks for a rendition in the correct format among those that have a modifier and returns the rendition having the smallest modifier. (Smallest is defined as the page modifier that is first if the modifiers are sorted alphanumerically in ascending order.) If the server can't find a rendition with the correct format among those that have a modifier, it will try to convert the content file to the correct format.

### Processing the return value

Getcontent returns a collection. To retrieve the actual file, iterate through the collection using the Next method. The number of iterations depends on the size of the actual content file.

When the file is larger than the buffer, the application must copy the contents of the buffer to another file after each execution of Next. With each iteration of Next, the server copies the next chunk of the file into the buffer.

The strings placed in the buffer are not zero terminated. To determine the actual length of the string in the buffer, examine the current_length attribute of the collection's result object.

## Related methods

Getfile, page 241

## Example

```
coll = dmAPIGet("getcontent," session "," doc_id")
while (dmAPIExec("next," session"," coll))
{msg = dmAPIGet("get," session "," coll "_content_buffer")
```

```
print msg
}
dmAPIExec ("close," session)
```

# Getdocbasemap

## Purpose

Returns information about the repositories that are known to the responding connection broker.

## Syntax

```
dmAPIGet("getdocbasemap,session[,protocol
,host_name,port_number]")
```

## Arguments

**Table 2-71. Getdocbasemap method arguments**

| Argument | Description |
|---|---|
| *session* | Identifies an open session. You can specify an open repository session or use the keyword apisession to specify the API session started by a dmAPIInit call. Use apisession if you want to issue the Getdocbasemap call when you are not connected to a repository. |
| *protocol* | Identifies the protocol used to talk to the connection broker. The only valid protocol is RPC_STATIC (uppercase).<br><br>This argument is optional. However, if you include it, it is not effective unless you include *host_name* and *port_number* also. |
| *host_name* | Identifies the host machine on which the connection broker resides.<br><br>This argument is optional. However, if you include it, it is not effective unless you include *protocol* and *port_number* also. |
| *port_number* | Specifies the port number which the connection broker is using for communications.<br><br>This argument is optional. However, if you include it, it is not effective unless you include *protocol* and *host_name* also. |

## Return value

The Getdocbasemap method returns a non-persistent ID for a docbase locator object.

## Usage notes

The Getdocbasemap method asks a connection broker to respond with a description of the repositories that are known to the connection broker. The optional arguments allow you to specify which connection broker responds. These arguments uniquely identify a connection broker. (You can obtain the values for these attributes for the connection brokers by issuing a Getdocbrokermap method call.)

If you do not include the arguments, the request is sent to all connection brokers defined in the user's dmcl.ini file. The returned repository map contains information about the repositories known to all connection brokers responding to the request.

The returned information includes the repository IDs, their names, and a verbose description of each. This information is returned in the attributes of a docbase locator object. To obtain the attribute's values, use a Get method, specifying the ID that the method returns.

## Related methods

Getdocbrokermap, page 237
Getservermap, page 256

## Example

```
repositorymap = dmAPIGet("getdocbasemap,c")
if (repositorymap == NULL)
print dmAPIGet("getmessage,s0");
```

# Getdocbrokermap

## Purpose

Returns information about the connection brokers that are known to the client DMCL.

## Syntax

```
dmAPIGet("getdocbrokermap,session")
```

## Arguments

**Table 2-72. Getdocbrokermap method arguments**

| Argument | Description |
|---|---|
| *session* | Identifies an open session. You can specify an open repository session or use the keyword apisession to specify the API session started by a dmAPIInit call. Use apisession if you want to issue the Getdocbasemap call when you are not connected to a repository. |

## Return value

The method returns an object ID for a docbroker locator object.

## Usage notes

The connection brokers that a particular client session can access are defined in the client dmcl file used by that session. A Getdocbrokermap method call returns a non-persistent ID for a docbroker locator object that contains, in repeating attributes, a list of the connection brokers that are known to the client dmcl file. This information includes the protocol used by the connection broker, which host machine it resides on, what port number it is using and the length of time a server waits for a reply from the connection broker before sending the request to another connection broker.

To obtain the values in the docbase locator object's attributes, use the Get method, with the ID returned by the Getdocbrokermap method.

## Related methods

Getdocbasemap, page 235
Getservermap, page 256

## Example

```
DBMapId = dmAPIGet("getdocbrokermap," session)
```

# Getevents

## Purpose

Returns all unread items in the user's inbox.

## Syntax

```
dmAPIGet("getevents,session")
```

## Arguments

**Table 2-73. Getevents method arguments**

| Argument | Description |
|---|---|
| *session* | Identifies an open repository session. |

## Return value

The Getevents method returns the collection identifier for a collection of objects of type dmi_queue_item. (Refer to Table 2–121, page 352 in the *EMC Documentum Object Reference Manual* for a description of the dmi_queue_item attributes.)

## Usage notes

Each item in an inbox is stored in an object of type dmi_queue_item. Getevents returns an identifier for the collection of all dmi_queue_item objects for the user that have a read_flag value of 0, which indicates an unread object. Every dmi_queue_item object has an attribute called read_flag. This attribute is set to 0 (FALSE) when the dmi_queue_item object is created (when the object is queued to the user). Issuing Getevents retrieves all dm_queue_items for the user that have read_flag = 0 and then resets read_flag to 1 (TRUE) for those items.

**Note:** When users open their Inboxes, the read_flag is also set to 1 for any new items in the Inbox. Consequently, Getevents only returns items that users have not viewed through their Inboxes or a previous Getevents.

Using the collection identifier, you can iterate through the collection.

## Example

This example retrieves all of the queued events for the current user:

```
coll = dmAPIGet("getevents," session);
while (dmAPIExec("next," session "," coll))
{
name = dmAPIGet("get," session "," coll ",name")
inst = dmAPIGet("get," session "," coll ",message")
event = dmAPIGet("get," session "," coll ",event")
stamp = dmAPIGet("get," session "," coll "_id")
printf "Event" event "has occurred on "name" stamp ("stamp")"
if (inst != "") print "Instructions: " inst
}
err_flag = dmAPIExec("close," session "," query_id)
if (err_flag == 0) {
err_msg = dmAPIGet("getmessage," session)
print "Error closing the query collection"
print err_msg
exit
}
```

# Getfile

## Purpose

Retrieves a content file from the repository.

## Syntax

```
dmAPIGet("getfile,session,object_id[,file_name][,format]
[,page_number][,get_resource][,page_modifier]")
```

## Arguments

**Table 2-74. Getfile method arguments**

| Argument | Description |
|---|---|
| *session* | Identifies an open repository session. |
| *object_id* | Specifies the object with which the file is associated. Use the object's ID or an indirect reference (*@object_id*) that points to the object. |
| *file_name* | Defines the location where you want to put the copy of the retrieved content file. This is an optional argument. If unspecified, the server places the copy in the client's local area or the Content Server common area.<br><br>**Note:** Files in the client local area or the common area are removed when the session is closed or when the user terminates the connection to the repository. |
| *format* | Specifies the format in which you want the file. This can be any valid file format or NULL. If you specify NULL, the server returns the file in the format specified in the a_content_type attribute of the object. |
| *page_number* | Identifies the content page with which the content file is associated. Use the content's page number. The default is zero (the first content page). |

| Argument | Description |
|---|---|
| *get_resource* | For files created on Macintosh systems, indicates whether you want the getfile to retrieve the data fork (the file containing the content) or the resource fork. FALSE (the default) returns the data fork (the content file). TRUE retrieves the resource fork (the file specified in the other_ticket attribute of the content object).<br><br>If the content is stored in an external storage area, the other_file argument is ignored. |
| *page_modifier* | Character string that identifies a rendition. You must include the page modifier if there are multiple renditions in the specified format associated with the specified content page.<br><br>The default is an empty string. |

## Return value

The Getfile method returns a character string that contains the location of the retrieved file as a full-path specification.

## Usage notes

Getfile retrieves content files associated with a document. Each Getfile retrieves one content file. The file can be primary content or a rendition. You must have at least Read permission on the document that contains the file. If the content file is not in the format requested, the server converts the file to the specified format. If the server cannot do the conversion, you receive an error message.

### Identifying the content file

The combination of the format, page_number, and page_modifier arguments identify which content file to return. To return the primary content of the first content page, default these arguments. For example,

```
dmAPIGet("getfile,c,0900000223643ec1,c:\temp\myfile")
```

All primary content of a document have the format defined in the document's a_content_type attribute. Page_number defaults to zero, which represents the first content page. And the page modifier for all primary content is an empty string.

To return any other content file associated with a document, whether it is a primary content or rendition, you must use one or more of the arguments. For example, to return the primary content associated with the second content page of 0900000223643ec1, include the page_number argument:

```
dmAPIGet("getfile,c,0900000223643ec1,c:\temp\myfile,,2")
```

To retrieve the PDF rendition of the third page of the document, include the format and the page number:

```
dmAPIGet("getfile,c,0900000223643ec1,c:\temp\myfile,pdf,3")
```

Suppose there are two PDF renditions associated with the second page of the document. The first has the page modifierfirst and the second has the page modifier second. To retrieve the first PDF rendition, include the format, page number, and page modifier:

```
dmAPIGet("getfile,c,0900000223643ec1,c:\temp\myfile,
pdf,2,first")
```

If you include a page modifier and a rendition with that modifier isn't found, the method returns an error.

If you don't include a page modifier, the server looks first for a content file or rendition with the specified format and no modifier. If one is not found, the server looks for a rendition in the correct format among those that have a modifier and returns the rendition having the smallest modifier. (Smallest is defined as the page modifier that is first if the modifiers are sorted alphanumerically in ascending order.) If the server can't find a rendition with the correct format among those that have a modifier, it will try to convert the content file to the correct format.

### Saving the content to a file

When the server saves the file to the specified location, it substitutes a hyphen (-) for any character in the file name that it cannot map to an ASCII character. For example, if a user's session code page is UTF-8 and the client_os_codepage is Shift_JIS or EUC-KR, the server substitutes hyphens for all characters in the file name that it cannot map to ASCII characters.

## Related methods

Setfile, page 434
Setpath, page 439

## Example

This example:

- Queries the repository for current parts of Chapter 8.
- Loops through the resulting collection, printing each respondent title, checking if the user wants the content from this document, andif so, getting the content.
- Closes the query collection.

```
query_str = "select r_object_id,title from dm_document"
query_str = queryd_str " where title like 'Chapter 8 -%'"
```

```
query_id = dmAPIGet("query," session "," query_str)
if (query_id == NULL) {
err_msg = dmAPIGet("getmessage," session)
print "Error querying the repository"
print err_msg
exit
}

resp_cntr = 0
while( dmAPIExec("next," session "," query_id) > 0 ) {
title = dmAPIGet("get," session "," query_id ",title")
if (title == NULL) {
err_msg = dmAPIGet("getmessage," session)
print "Error retrieving document title from the query collection"
print err_msg
exit
}

printf "Get content for " title "? (Y/N)"
getline answer < "-"
if (answer == "Y" || answer =="y") {
doc_id = dmAPIGet("get," session "," query_id ",r_object_id")
if (doc_id == NULL) {
err_msg = dmAPIGet("getmessage," session)
print `Error retrieving document ID from the query collection'
print err_msg
exit
}

filename = dmAPIGet("getfile," session "," doc_id)
if (filename == NULL) {
err_msg = dmAPIGet("getmessage," session)
print "Error retrieving document content"
print err_msg
exit
}
print "The contents are in file '" filename "'"
break
}
}

err_flag = dmAPIExec("close," session "," query_id)
if (err_flag == 0) {
err_msg = dmAPIGet("getmessage," session)
print "Error closing the query collection"
print err_msg
exit
}
```

# Getlastcoll

## Purpose

Returns the collection ID of the last executed query.

## Syntax

```
dmAPIGet("getlastcoll,session")
```

## Arguments

**Table 2-75. Getlastcoll method arguments**

| Argument | Description |
|---|---|
| *session* | Identifies an open repository session. |

## Return value

The Getlastcoll method returns a collection ID.

## Usage notes

The Getlastcoll method returns the collection ID of the collection that contains the results of the most recently executed Execquery, Query, or Readquery method. Because the Query and Readquery methods return this ID directly, the Getlastcoll method is primarily useful in processing the results of an Execquery method.

Refer to Using execquery in an application, page 351 in *Content Server Fundamentals* for more information about using the Getlastcoll method in conjunction with the Execquery method.

## Example

```
status = dmexec("execquery,s0,T,select subject from\ dm_document where
owner_name = USER")
query_id = dmget("getlastcoll,s0")
if (query_id == NULL)
{  err_mess = dmget("getmessage,s0")
     print err_mess
     exit
```

```
}
 while(dmexec("next,s0,query_id"))
  { subject = dmget("get,s0," query_id ",subject")
    print subject
  }
  dmexec("close,s0," query_id)
```

# Getlogin

## Purpose

Returns a ticket from the current session that an application can use to launch another session as the currently logged-in user.

## Syntax

```
dmAPIGet("getlogin,session [,user][,scope]
[,timeout_period][,single_use][,server_name]")
```

## Arguments

**Table 2-76. Getlogin method arguments**

| Argument | Description |
|---|---|
| *session* | Identifies the current session. |
| *user* | Identifies the user for whom to return the login ticket. You must be a superuser to include this argument. |
| *scope* | Defines the scope for the login ticket. Valid values are:<br><br>**server**, meaning the ticket may only be sent to the server that issued the ticket<br><br>**docbase**, meaning the ticket may be used to connect to any server servicing the repository in which the ticket was issued<br><br>**global**, meaning that the ticket may used to connect to any trusted respository<br><br>The default value is global. |
| *timeout_period* | Defines how long the login ticket is valid. The value is expressed in minutes. If the value is larger than the value in the max_login_ticket_timeout attribute of the docbase config object, the value in max_login_ticket_timeout is used.<br><br>If unspecified, the default is the value in the login_ticket_timeout attribute in the docbase config object. |

| Argument | Description |
|---|---|
| *single_use* | Indicates whether the ticket may be used only once or multiple times. T means the ticket may be used only once. F means the ticket may be used multiple times.<br><br>Setting this to T automatically scopes the ticket to a particular server. If the *server_name* argument is included, then that is the server that may use the ticket. If *server_name* is not included, the ticket may only be used by the server issuing the ticket.<br><br>If this is F, the ticket may be used multiple times. The scope is defined in this case by the *scope* argument. You cannot include the *server_name* argument when *single_use* is set to F.<br><br>The default is F. |
| *server_name* | Identifies a server for the scope of a single-use ticket. The *single_use* argument must be set to T if *server_name* is included.<br><br>If *single_use* is T and *server_name* is not provided, the scope is the issuing server. If *server_name* is included, the scope is the specified server. |

## Return value

The Getlogin method returns a value that has the following format:

`DM_TICKET=`*`ASCII-encoded_string`*

The *ASCII-encoded_string* consists of a set of values describing the login ticket and a signature generated from those values, all encoded into one ASCII string.

The entire value (including the prefix) may be up to 350 bytes long.

## Usage notes

The Getlogin method returns a unique login ticket that you can use in an application to allow the application to open a new session with the current repository or a different repository without specifying a user password. The ticket replaces the user's password in a method call.

The application using the ticket must connect as the user for whom the ticket is generated. If the method is issued without the *user* argument, the ticket is issued for the user issuing the Getlogin method. If the *user* argument is specified, the ticket is issued for the specified user. You must have Superuser privileges to issue Getlogin with the user argument.

The sessions that you start using login tickets are counted towards the maximum number of allowed concurrent sessions for the server. If this limit allows, you can have a maximum of 2000 active sessions that were started through a ticketed login.

The maximum number of tickets a user can obtain at any time cannot exceed 1.25 times the number of concurrent_sessions specified in the server.ini file. For example, if concurrent_sessions is 100, a user cannot have more than 125 tickets at a time.

### Specifying the ticket scope

The scope of a login ticket identifies which server or servers accept the ticket. You can define the scope as:

- The server that issues the ticket
- A single server other than the issuing server

  In this case, the ticket is automatically a single-use ticket.
- All servers in the repository served by the issuing server
- All servers servicing trusted respositories

To define the scope as the issuing server, set the scope argument to server. If you include the *single_use* argument, do not include the *server_name* argument.

To define the scope as a specific server other than the issuing server, set the *single_use* argument to T and identify the server in the *server_name* argument. The server may be any server in the issuing repository or in a trusted respository. The *scope* argument is ignored in this situation.

To define the scope as all servers in the repository served by the issuing server, set the *scope* argument to docbase. Do not set the *single_use* argument.

To define the scope as all servers serving trusted repositories, set the *scope* argument to global. Do not set the *single_use* argument.

(For information about trusted repositories, refer to Trusted repositories, page 37, in *Content Server Fundamentals*. For instructions on setting up trusted respositories, refer to Configuring a repository's trusted repositories, page 449, in the *Content Server Administrator's Guide*.)

### Specifying the ticket's expiration

A ticket is valid for a given period of time. After that period, the ticket expires and cannot be used.

If you do not include the *timeout* argument in the Getlogin method, the period defaults to the value defined at the repository level, in the docbase config object. If you include the argument, that value overrides the default unless the value is greater than the maximum allowed period specified in the docbase config object. If the argument value is greater than the maximum allowed value, the period defaults to the maximum allowed value. (To change the default and maximum periods, refer to Configuring login ticket use, page 450, in the *Content Server Administrator's Guide* for instructions. )

## Related methods

Assume, page 106
Authenticate, page 123
Connect, page 155

## Example

```
login = dmAPIGet("getlogin,c,,docbase,30")
if (getlogin == NULL)
print dmAPIGet("getmessage,s0");
```

# Getmessage

## Purpose

Returns all messages from the session at or above a specified severity level.

## Syntax

```
dmAPIGet("getmessage[,session][,severity_level]")
```

## Arguments

**Table 2-77. Getmessage method arguments**

| Argument | Description |
|---|---|
| *session* | Identifies an open repository session. This argument is optional if you are using the method to obtain the messages associated with a failure to connect to a session. In all other instances, you must specify the argument. |
| *severity_level* | Defines the minimum severity level of the messages you want to get. Use the integer value to specify any of the following valid severity levels:<br><br>1, for Information messages<br>2, for Warning messages<br>3, for Error messages<br>4, for Fatal error messages |

## Return value

The Getmessage method returns all the messages at the specified level or higher. The messages are returned as a formatted string with separating line feeds.

## Usage notes

You can also use this method to obtain error messages if an attempt to open a session fails.

## Related methods

Listmessage, page 302

## Example

This example:

- Connects to the server
- Uses Getmessage to retrieve any status or error messages associated with the connection attempt
- Disconnects from the server

```
session = dmAPIGet("connect," repository "," user "," passwd)
if (session == NULL) {
print "Error connecting to repository '" repository "'"
err_msg = dmAPIGet("getmessage," session)
print err_msg
exit
}

inf_msg = dmAPIGet("getmessage," session)
print inf_msg
err_flag = dmAPIExec("trace," session ",1")
if (err_flag == 0) {
print "Error establishing trace level "1'"
err_msg = dmAPIGet("getmessage," session)
print err_msg
exit
}

err_flag = dmAPIExec("disconnect," session)
if (err_flag == 0) {
print "Error disconnecting from repository '" repository "'"
err_msg = dmAPIGet("getmessage," session)
print err_msg
exit
}
```

# Getpath

## Purpose

Retrieves the storage location of a content file.

## Syntax

```
dmAPIGet("getpath,session,object_id[,page][,format]
[,page_modifier][,get_resource]")
```

## Arguments

**Table 2-78. Getpath method arguments**

| Argument | Description |
|---|---|
| *session* | Identifies an open repository session. |
| *object_id* | Object ID of the object with which the content is associated. |
| *page* | The content page number with which the content file is associated. The default is 0. |
| *format* | The format of the content file. Use the format's name as defined in the format's associated format object. The default is the value in a_content_type for the object identified in *object_id*. |
| *page_modifier* | Character string that identifies a rendition. You must include the page modifier if there are multiple renditions in the specified format associated with the specified content page.<br><br>The default is an empty string. |
| *get_resource* | For files created on Macintosh systems, indicates whether you want the method to retrieve the data fork (the file containing the content) or the resource fork. FALSE (the default) returns the data fork (the content file). TRUE retrieves the resource fork (the file specified in the other_ticket attribute of the content object).<br><br>If the file is stored on an external storage area, this argument's setting is ignored. |

## Return value

The Getpath method returns a string value containing the path to the content file. If the content file is stored in a CA store storage area, the method returns the content address.

## Usage notes

The Getpath method returns the storage location of a content file. The location is either a content address if the content is stored in a content-addressed storage system or Content Server's internal representation of the storage location. For example, suppose the primary content file for a particular document is a text file stored in a file store storage area called engr_filestore. Assuming that the object ID of the document is 0900000328561e7c, the following Getpath returns the internal path specification for the content file:

```
dmAPIGet("getpath,c,0900000328561e7c")
```

The returned path specification might look like this:

```
C:\DOCUMENTUM\data\production\engr_filestore\
00001313\80\00\16\e2.txt
```

The first portion identifies the path to the storage area. The remaining portion is derived from the content file's data ticket value. (For a complete description of how internal file paths for internal storage areas are generated, refer to Path specifications for content in file stores, page 244 in the *Content Server Administrator's Guide*.)

Using Getpath on files that are stored in encrypted storage areas does not generate an error, but provides no useful functionality because custom applications cannot decrypt content files.

### Identifying the content file

The combination of the format, page_number, and page_modifier arguments identify which content file path to return. To return the path to the primary content of the first content page, default these arguments. For example, the following statement returns the path to the primary content of the first page of the document represented by 0900000223643ec1:

```
dmAPIGet("getpath,c,0900000223643ec1")
```

The default format is the format in the document's a_content_type attribute, and a document's primary content must have the format defined in that attribute. Page_number defaults to zero, which represents the first content page. The default for page_modifier is an empty string, and the page modifier for all primary content is an empty string.

To return the path to any other content file associated with a document, whether it is a primary content or rendition, you must use one or more of the arguments. For example, to return the path to the primary content associated with the second content page of 0900000223643ec1, include the page_number argument:

```
dmAPIGet("getpath,c,0900000223643ec1,2")
```

To retrieve the path to the PDF rendition of the third page of the document, include the the page number and the format:

```
dmAPIGet("getpath,c,0900000223643ec1,3,pdf")
```

Suppose there are two PDF renditions associated with the second page of the document. The first has the page modifierfirst and the second has the page modifier second. To retrieve the path to the first PDF rendition, include the format, page number, and page modifier:

```
dmAPIGet("getpath,c,0900000223643ec1,2,pdf,first")
```

If you include a page modifier and a rendition with that modifier isn't found, the method returns an error.

If you don't include a page modifier, the server looks first for a content file or rendition with the specified format and no modifier. If one is not found, the server looks for a rendition in the correct format among those that have a modifier and returns the path to the rendition having the smallest modifier. (Smallest is defined as the page modifier that is first if the modifiers are sorted alphanumerically in ascending order.) If the server can't find a rendition with the correct format among those that have a modifier, it will return an error.

## Related methods

Setpath, page 439

## Example

The following example retrieves the path to the first primary content file found the document represented by the 0900007b8000566b object ID:

```
path = dmAPIGet("getpath," session ",0900007b8000566b")
```

# Getservermap

## Purpose

Returns information about the servers known to a connection broker.

## Syntax

```
dmAPIGet("getservermap,session,repository_name[,protocol]
[,host_name][,port_number]")
```

## Arguments

**Table 2-79. Getservermap method arguments**

| Argument | Description |
|---|---|
| *session* | Identifies an open session. You can specify an open repository session or you can use the keyword apisession to specify the API session. Use apisession if you want to issue the Getservermap call when you are not connected to a repository. |
| *repository_name* | Identifies the repository associated with the servers for which you are requesting information. |
| *protocol* | Specifies the protocol that the connection broker is using to communicate. |
| *host_name* | Identifies the machine on which the connection broker resides. Use the machine's host name. |
| *port_number* | Specifies the port that the connection broker is using for communications. |

## Return value

The Getservermap method returns a non-persistent ID for a server locator object.

## Usage notes

The Getservermap method asks a connection broker to respond with a description of the servers that are known to it. The optional arguments allow you to specify which Register responds. These arguments uniquely identify a connection broker. (You can obtain the

values for these attributes for the connection brokers by issuing a Getdocbrokermap method call.) It is not necessary to include all the arguments. If you include only one, for example, the port number, the request is sent to a connection broker using that port number. However, using all three does allow you to fully control which connection broker responds.

If you do not include the arguments, the request is sent to the connection broker defined as the primary connection broker for the session. This information is found in the client's dmcl.ini file. If the primary connection broker is not available, the request is sent to each backup connection broker, in turn, until one responds. (You will receive error messages for those that do not respond.) A server locator object has three attributes that will tell you which connection broker responded to the request.

The returned information about servers includes their names, the name of the host machines on which they reside, and their process IDs as well as other information. (Refer to Table 2–154, page 437 in the *EMC Documentum Object Reference Manual* for a description of the attributes of a server locator object.) To obtain this information, use the Get method, with the ID returned by the Getservermap method, to retrieve the values in the attributes of the server locator object.

## Related methods

Getdocbasemap, page 235
Getdocbrokermap, page 237

## Example

```
servermap = dmAPIGet("getservermap,c,repository")
if (servermap == NULL)
print dmAPIGet("getmessage,s0");
```

# Grant

## Purpose

Creates an entry in an ACL object.

## Syntax

```
dmAPIExec("grant,session,object_id[,accessor_name|alias]
[,permit_type][,unused][,basic_permission|extended_
permit][,application_permit]")
```

## Arguments

**Table 2-80. Grant method arguments**

| Argument | Description |
|---|---|
| *session* | Identifies an open repository session. |
| *object_id* | Directly or indirectly identifies the ACL to which you are adding the entry. The *object_id* can be the object ID of the ACL itself, for a direct identification, or it can be the object ID of a SysObject with which the ACL is associated, for an indirect identification.<br><br>If the object ID identifies a SysObject, you can use the object's ID or an indirect reference (*@object_id*) that points to the object.<br><br>**Note:** The behavior of the method varies depending on whether you reference the ACL directly or indirectly. Refer to the Usage notes for details. |
| *permit_type* | Defines the kind of entry. Valid values are:<br><br>AccessPermit<br>ExtendedPermit<br><br>With a Trusted Content Services license, the following values are also valid entries:<br><br>ApplicationPermit<br>AccessRestriction<br>ExtendedRestriction<br>ApplicationRestriction<br>RequiredGroup<br>RequiredGroupSet |

| Argument | Description |
|---|---|
| | The default is AccessPermit. |
| *accessor_name* | Identifies the user or group that is the subject of the entry. You can specify any individual user or any group in the repository.<br><br>If *permit_type* is RequiredGroup or RequiredGroupSet, this argument must be set to a group name.<br><br>If you include this argument, do not include *alias*. |
| *alias* | Defines an alias in place of an actual user or group name. The alias is resolved when the ACL is attached to an object. This argument is intended for use in template ACLs. The format of an alias is:<br><br>%[*alias_set_name*.]*alias_name*<br><br>*alias_set_name* is the name of an alias set object.<br><br>*alias_name* is a value in the alias_name attribute of the alias set object.<br><br>If *permit_type* is RequiredGroup or RequiredGroupSet, the alias must resolve to a group name. |
| *basic_permission* | Defines the base object-level permission you are assigning to the user or group specified in *accessor_name*. Specify this as an integer. Valid values are:<br><br>1, for None<br>2, for Browse<br>3, for Read<br>4, for Relate<br>5, for Version<br>6, for Write<br>7, for Delete<br><br>If this argument is included, *permit_type* must be AccessPermit or AccessRestriction and you may not include *extended_permission*.<br><br>For information about these levels, refer to the Usage notes. |

| Argument | Description |
|---|---|
| *extended_permissions* | Defines the extended object-level permissions that you are assigning to the user or group specified in *accessor_name*. Specify this in string form. Valid values are:<br><br>change_location<br>change_owner<br>change_permit<br>change_state<br>delete_object<br>execute_proc<br><br>If this argument is included, *permit_type* must be ExtendedPermit or ExtendedRestriction and you may not include *base_permission*.<br><br>For information about these levels, refer to the Usage notes. |
| *application_permit* | A user-defined permission to be interpreted and enforced by an application. Content Server ignores any permission set using this argument.<br><br>The maximum allowed length of the permission's name is 128 bytes.<br><br>If this argument is included, *permit_type* must be ApplicationPermit or ApplicationRestriction. |

## Return value

The Grant method returns TRUE if successful or FALSE if unsuccessful.

## Usage notes

The Grant method creates an access control entry in an ACL object. Access control entries are recorded in a set of repeating attributes in the ACL. The values at corresponding index levels across the attributes constitute the individual access control entries in the ACL.(Managing ACLs, page 387, in *Content Server Administrator's Guide* describes ACLs and access control entries in detail.)

You can execute Grant against a particular object, to change that object's permissions, or you can execute Grant against an ACL, to change the permissions of all objects controlled by that ACL. For instruction on using Grant to change an object's permissions, refer to Setting permissions on a particular object, page 261. For instructions on using

Grant to change an ACL directly, refer to Setting permissions for all objects controlled by a particular ACL, page 261.

### Permissions required to use Grant

To execute Grant against a SysObject, you must either own the object, have Change Permit permission on the object, or have Sysadmin or Superuser privileges.

To execute Grant against an ACL, you must either own the ACL or have Superuser privileges.

### Setting permissions on a particular object

To set the permissions on a particular object, set the *object_id* argument to the object's object ID.

When you specify the object ID of a SysObject (or subtype) in the Grant method arguments, the server copies the ACL associated with the object, makes the changes to the copy, and then applies the new copy to the object. Only the access to that object is changed. The server does not change the original ACL associated with the object. The access to other objects that use the original ACL is not changed.

For example, suppose there are three documents, each associated with the same ACL and suppose that after a review, Document A is ready for public consumption but Documents B and C need more work. To change the access for Document A only, you reference Document A in the Grant method:

```
fetch,s0,DocA_object_id
grant,s0,DocA_object_id,all_company,3
save,s0,DocA_object_id
```

Everyone in the company now has at least Read permission on Document A (assuming that all_company is a group that includes all company personnel). However, access to Document B and Document C remains unchanged.

### Setting permissions for all objects controlled by a particular ACL

To set permissions for all objects controlled by an ACL, you specify the ACL's object ID in the *object_id* argument. You must own the ACL or have Sysadmin or Superuser privileges.

If you specify an ACL's object ID in the Grant method arguments, it changes the ACL directly and, consequently, sets the access permissions for all SysObjects that use that ACL.

For example, suppose you have an ACL (*acl_object_id*) that controls access to Documents A, B, and C, for the flagship project. Now, three weeks into the project, you want to circulate the project's documents to a marketing manager, JamesT, for review. To give that person access to the project's documents, you could create an entry for that person in the ACL. You specify the ACL directly in the Grant method so that all the documents will be affected, giving JamesT access to all of them:

```
fetch,s0,acl_object_id
```

```
grant,s0,acl_object_id,jamest,4
save,s0,acl_object_id
```

JamesT now has Relate permission on Documents A, B, and C.

### Using an alias

Use the *alias* argument to create an access control entry that has an alias in the r_accessor_name attribute. Aliases are typically set in entries in template ACLs. When the template is associated with an object, the server creates a copy of the template, resolves the alias to a user name or group name, and applies the copy with the resolved alias to the object.

### Defining the entry type

The *permit_type* argument identifies which type of entry you are adding. There possible types of entries are:

- AccessPermit

  An AccessPermit entry identifies a user or group and the base permission you are assigning to the user or group.

- AccessRestriction

  An AccessRestriction entry identifies a user or group and restricts the user or group's access to the level specified in the entry even though that user or group may be granted a higher access level by another entry.

- ExtendedPermit

  An AccessPermit entry identifies a user or group and the extended permission you are assigning to the user or group.

- ExtendedRestriction

  An ExtendedRestriction entry identifies a user or group and restricts that user or group's access from the specfied extended permission.

- ApplicationPermit

  An ApplicationPermit entry identifies a user or group and a user-defined permission level granted to that user or group. Application permits are interpreted only by user applications. Content Server does not enforce application permits.

- ApplicationRestriction

  An ApplicationRestriction entry identifies a user or group and restricts that user or group from exercising the specified application permit. Application restrictions are interpreted only by user applications. Content Server does not enforce application restrictions.

- RequiredGroup

  A RequiredGroup entry identifies a group to which the user requesting access to an object controlled by the ACL must belong.

- RequiredGroupSet

  A RequiredGroupSet entry identifies a group. Typically, if RequiredGroupSet entries are defined for an ACL, the ACL will have multiple entries of that type. Each entry specifies one group. A user requesting access to an object controlled by the ACL must belong to at least one of the groups identified by RequiredGroupSet entries.

You must have installed Content Server with a Trusted Content Services license to grant the permit types AccessRestriction, ExtendedRestriction, RequiredGroup, RequiredGroupSet, ApplicationPermit, and ApplicationRestriction.

### Specifying the base and extended permissions

The base object-level permissions are hierarchical. That is, each level includes all levels below it. Consequently, when you grant a base permission, you are granting all permissions up to and including the specified permission. For example, granting Write permission gives the user or group the rights conferred by Browse, Read, Relate, and Version permissions in addition to Write permission.

Extended permission levels are not hierarchical. If you want a user to have an extended permission, you must grant each individually.

**Note:** Users with Browse permission or above on an object have Change Location and Execute Procedure extended permissions by default on the object.

If you are adding an access restriction entry, then specifying a base permission limits the user to the specified permission level even if other entries in the ACL grant the user a higher level of access. If the restrictive entry specifies an extended permission, it prohibits the user from exercising that extended privilege even if other entries give the user that privilege.

For more information about the object-level permissions, refer to Chapter 11, Protecting Repository Objects, in the *Content Server Administrator's Guide*.

## Related methods

Revoke, page 400

## Example

```
err = dmAPIExec ("fetch," session "," obj_id)
if (err == 0)
{
print "Error fetching the object '" obj_id "'"
print dmAPIGet ("getmessage," session)
exit
}

err = dmAPIExec ("grant," session "," obj_id ","\
accessor_name "," accessor_permit)
```

```
if (err == 0)
{
print "Error granting permit " accessor_permit "to\
 accessor '" accessor_name "'"
print dmAPIGet ("getmessage," session)
exit
}

err = dmAPIExec ("save," session "," obj_id)
if (err == 0)
{
print "Error saving the object '" obj_id "'"
print dmAPIGet ("getmessage," session)
exit
}
```

**Note:** obj_id can be either a SysObject ID or an ACL ID.

# Halt

## Purpose

Halts the specified workflow or a specific activity.

## Syntax

```
dmAPIExec("halt,session,workflow_id
[,activity_sequence_number[,suspend_interval]]")
```

## Arguments

**Table 2-81. Halt method arguments**

| Argument | Description |
|---|---|
| *session* | Identifies an open repository session. |
| *workflow_id* | Identifies the workflow that you want to halt. Use the workflow's object ID or an indirect reference (*@object_id*) that points to the object. |
| *activity_sequence_number* | Identifies a specific activity to halt. Use the activity sequence number as defined in the workflow. If you do not specify an activity, the method halts all activities in the workflow. |
| *suspend_interval* | Length of time for which the activity is halted. If set, the activity is automatically resumed when the interval has expired.<br><br>The value is interpreted in minutes. |

## Return value

The Halt method returns TRUE if successful or FALSE if unsuccessful.

## Usage notes

To halt a workflow, you must be the workflow's supervisor or a user with Sysadmin or Superuser privileges. The workflow must be in the running state and cannot have any automatic activities in the acquired state.

By default, to halt an activity, you must be the workflow's supervisor or user with Sysadmin or Superuser privileges. If workflow_security_disabled, a server.ini key, is set to TRUE, then any user can halt a running activity.

The Halt method changes the workflow's r_runtime_state or the activity's r_act_state to halted and marks all non-finished work items as paused.

Halting a workflow does not remove any queued tasks from users' inboxes. However, users cannot acquire or work on the tasks until the workflow has been resumed.

### Using the suspend_interval argument

If you include the *suspend_interval* argument when halting an activity, the activity is automatically resumed after the interval expires. If you do not include the argument, the activity is not resumed until you do so manually or programmatically, with a Resume method.

If the activity is assigned to a work queue, you must provide a value for this argument when halting the activity.

Only BPM or Webtop allows you to set this argument through a user interface. Workflow Manager does not support this argument.

## Related methods

Pause, page 320
Resume, page 392

## Example

```
status = dmAPIExec("halt,s0,4d0000013874e08a,20")
```

# Id

## Purpose

Returns the ID of the object that satisfies a specified qualification.

## Syntax

```
dmAPIGet("id,session,qualification")
```

## Arguments

**Table 2-82. Id method arguments**

| Argument | Description |
|---|---|
| *session* | Identifies an open repository session. |
| *qualification* | Defines the qualification that identifies the object for which you are searching. A qualification consists of that portion of a SELECT statement that appears after the keyword FROM. The first term in a qualification must be the type of the object for which you are searching. The remainder of the qualification can include a WHERE clause, an IN DOCUMENT clause, an IN ASSEMBLY clause, or a full-text search clause. |

## Return value

The Id method returns an object ID. If no object matches the qualification, the method returns a warning.

## Usage notes

You can use this method as a shortcut for a full DQL query when you want to obtain an object ID.

**Note:** Use the Query_cmd method instead of the Id method if you want to cache the query results.

## Example

The following example returns the object ID of the APPROVED version of the document whose name is Style Guide.

```
claus = "object_name='Style Guide' and any r_version_label = 'APPROVED'"
obj_type = "dm_document (all)"
doc_id = dmAPIGet("id," session "," obj_type "where" claus)
if (doc_id == NULL) {
print "Error retrieving document complying with conditions:"
print claus
print dmAPIGet("getmessage," session)
}
```

# Initcrypto

## Purpose

Initializes an AEK file.

## Syntax

```
dmAPIExec("initcrypto,apisession[,location][,passphrase]")
```

## Arguments

**Table 2-83. Initcrypto method arguments**

| Argument | Description |
|---|---|
| *location* | Identifies the location of the AEK file.<br><br>This is optional. Refer to the Usage notes for a description of the behavior if *location* is not included. |
| *passphrase* | Used to decrypt the AEK.<br><br>This is optional. Refer to the Usage notes for a description of the behavior if *passphrase* is not included. |

## Return value

The Initcrypto method returns TRUE if successful and FALSE if unsuccessful.

## Usage notes

Anyone can execute this method.

The -passphrase argument is used to decrypt the AEK identified in the -location argument.

If -passphrase is not included, the method does the following:

1. Tries to retrieve the passphrase from shared memory.

   (Typcially, a passphrase is obfuscated and placed in shared memory when multiple products on one host are using the same passphrase for their AEKs.)

2. If the passphrase is not in shared memory, the method tries to retrieve the AEK from shared memory.

   (The AEK in shared memory doesn't require a passphrase to use.)

3. If neither the passphrase or the AEK is in shared memory, the method assumes the passphrase is the default passphrase and uses the default to decrypt the AEK specified by -location.

If you do not include the *location* argument, the method examines the DM_CRYPTO_FILE environment variable to determine the location of the AEK. If that variable is not set, the method assumes that the AEK is found in %DOCUMENTUM%\dba\secure\aek.key ($DOCUMENTUM/dba/secure/aek.key). If the AEK is not found in that location, the method fails.

You can call the Initcrypto method only once in a single API session.

## Related methods

Encryptpass, page 201

## Example

The following example opens the AEK and then calls Encryptpass to encrypt the password foo.

```
Status = dmAPIExec("initcrypto,apisession,C:\Documentum\ARP\Crypto")
If (Status = 0) Then
  Call DmPrint("Could not open key store at
    C:\Documentum\ARP\Crypto")
Else
  encrypted_password$ = dmAPIGet("encryptpass,apisession,foo")
  If (encrypted_password = "") Then
    Call DmPrint("Could not encrypt password")
  End if
End if
```

# Insert

## Purpose

Inserts a value into a repeating attribute.

## Syntax

```
dmAPISet("insert,sesion,object_id,attribute[[index]]
[,pattern]","value")
```

## Arguments

**Table 2-84. Insert method arguments**

| Argument | Description |
|---|---|
| *session* | Identifies an open repository session. |
| *object_id* | Identifies the object that contains the repeating attribute. Use the object's ID or an indirect reference (*@object_id*) that points to the object.<br><br>If the object is the api config object, use the keyword apiconfig instead of an object ID or indirect reference. |
| *attribute* | Specifies the repeating attribute into which you are inserting the value. Use the attribute's name. |
| *pattern* | Valid only if the attribute is a date attribute, this argument defines a date format for the specified value. The pattern can be any valid input format for dates. |
| [*index*] | Defines the inserted value's position in the ordered list of values in the attribute. If you do not specify an index value, the value is inserted at position zero. You must include the square brackets. |

## Return value

The Insert method returns TRUE if successful or FALSE if unsuccessful.

## Usage notes

This method only inserts values into repeating attributes. You cannot use this method to set a single-valued attribute. (Use Set to put a value in a single-valued attribute.)

If the attribute contains date values, you may want to specify an input format for the value. This ensures that a specified value is not misinterpreted by the server. For example, 03/04/1995 could be interpreted as March 4, 1995 (*mm/dd/yyyy*) or as April 3, 1995 (*dd/mm/yyyy*). To ensure that the value is interpreted as you wish, specify the format's pattern in the Insert method. For example:

```
dmAPISet("insert,c,09000002648921e5,review_date[3],
dd/mm/yyyy","08/03/1996")
```

## Related methods

Append, page 77
Set, page 416

## Example

This example:

- Creates a document object
- Associates content of the desired file with the new document
- Inserts the authors' names
- Saves the document object

```
doc_id = dmAPIGet("create," session ",dm_document")
if (doc_id == NULL) {
err_msg = dmAPIGet("getmessage," session)
print "Error creating a document object"
print err_msg
exit
}

the_file = "chap_1.wp5"
err_flag = dmAPIExec("setfile," session "," doc_id "," the_file ",wp5")
if (err_flag == 0) {
err_msg = dmAPIGet("getmessage," session)
print "Error setting file contents"
print err_msg
exit
}

author_list = "W. Smith,J. Jones,R. Abnous,S. Ruppenthal"
count = split(author_list, author, ",")
cmd_str = "insert," session "," doc_id ",authors[0]"
for (i=1; i<=count; i++) {
err_flag = dmAPIExec( cmd_str, author[i] )
if (err_flag == 0) {
err_msg = dmAPIGet("getmessage," session)
```

```
print "Error inserting authors' name to the 'authors' attribute"
print err_msg
exit
}
}

err_flag = dmAPIExec("save," session "," doc_id)
if (err_flag == 0) {
err_msg = dmAPIGet("getmessage," session)
print "Error saving the document object"
print err_msg
}
exit
```

# Insertcontent

## Purpose

Inserts a new content file into an object.

## Syntax

```
dmAPISet("insertcontent,session,object_id[,page_number]
[,length][,set_resource]","content")
```

## Arguments

**Table 2-85. Insertcontent method arguments**

| Argument | Description |
|---|---|
| *session* | Identifies an open repository session. |
| *object_id* | Identifies the object into which you are inserting the new content. Use the object's ID or an indirect reference (*@object_id*) that points to the object. |
| *page_number* | Specifies the position of the new content in the object's ordered list of contents. If unspecified, this defaults to zero (the first position). |
| *length* | Specifies the length, in bytes, of the content. This is only required if the content is binary content. |
| *set_resource* | For files created on Macintosh systems, indicates whether you want the method to insert the data fork (the file containing the content) or the resource fork. FALSE (the default) inserts the data fork (the content file). TRUE inserts the resource fork. |
| *content* | Specifies the location of the data in working memory. |

## Return value

The Insertcontent method returns TRUE if successful or FALSE if unsuccessful.

## Usage notes

Use the Insertcontent method in applications when you want to insert data residing in working memory as content for an object. The application must interact with the user's system to obtain the values for the *length* and *content* arguments. This method cannot be used directly by users. (Users wanting to add new content to an object directly should use Setfile, Appendfile, or Insertfile.)

If you are inserting content created on a Macintosh machine, you must issue the Insertcontent twice before saving the object. Issue the method once to add the data fork (set_resource=F) and then again to add the resource fork (set_resource=T). When you add the data fork, the content argument must identify the location in memory of the data fork. When you add the resource fork, the content argument must identify the memory location of the resource fork.

You cannot use the Insertcontent method with objects stored in external storage areas.

## Related methods

Appendcontent, page 80
Appendfile, page 82
Insertfile, page 277
Removecontent, page 355
Setcontent, page 424
Setfile, page 434
Setpath, page 439

## Example

```
doc_id = dmget("create,s0,dm_document");
if (doc_id == NULL)
{err_mess = dmget("getmessage,s0")
   print err_mess
   exit
}

status = dmset("set,s0,last,object_name", "my_document")
status = dmset("set,s0,last,a_content_type", "text")

content = "This manual is intended for use with the B1.1 release."

status = dmset("insertcontent,s0,last,0", content)
if (status != 1)
{err_mess = dmget("getmessage,s0")
   print err_mess
   exit
}

status = dmexec("save,s0,last")
if (status != 1)
```

```
{err_mess = dmget("getmessage,s0")
   print err_mess
   exit
 }
```

# Insertfile

## Purpose

Inserts a new content file into an object.

## Syntax

```
dmAPIExec("insertfile,session,object_id,file_name
[,page_number][,other_file]")
```

## Arguments

**Table 2-86. Insertfile method arguments**

| Argument | Description |
|---|---|
| *session* | Identifies an open repository session. |
| *object_id* | Specifies the object to which you are adding the new content file. This must be an object, such as a document, that can have content.<br><br>Use the object's ID or an indirect reference (*@object_id*) that points to the object. |
| *file_name* | Specifies the file that you are adding to the object. Use either an absolute or relative path to specify the file. |
| *page_number* | Defines the file's position among the ordered content files associated with the object. Use this argument only if the specified object has multiple content files. If you do not specify a page number, the default is zero. (Refer to the Usage notes for an explanation of page numbers.) |
| *other_file* | Specifies the file that contains the resource fork for a Macintosh document. You can use either an absolute or a relative path. If this is set, the path must be valid or the method fails.<br><br>The other_file value is ignored when used with external stores. |

## Return value

The Insertfile method returns TRUE if successful or FALSE if unsuccessful.

## Usage notes

You must have at least Version permission on the specified document to use Insertfile.

Every content file associated with an object has a page number that defines its position within the ordered list of contents for the object. You can determine the number of content files associated with an object by looking at the value in the object's r_page_count attribute. When you use Insertfile, if you specify the page number of an existing content, the server inserts the file at that position and renumbers all pages that follow.

Note that you must specify the format of the file by setting the a_content_type attribute of the document before you use the Insertfile method.

## Related methods

Appendfile, page 82
Setfile, page 434

## Example

This example:

- Creates a document object
- Uses setfile to add a first content to the document
- Uses insertfile to add files in front of the content added with the Setfile.
- Saves the document object

```
doc_id = dmAPIGet("create," session ",dm_document")
if (doc_id == NULL) {
err_msg = dmAPIGet("getmessage," session)
print "Error creating a document object"
print err_msg
exit
}

err_flag = dmAPISet("set," session "," doc_id ",a_content_type", "wp5" )
if (err_flag == 0) {
err_msg = dmAPIGet("getmessage," session)
print "Error setting content type"
print err_msg
exit
}

err_flag = dmAPIExec("setfile," session "," doc_id ",section5.wp5")
if (err_flag == 0) {
err_msg = dmAPIGet("getmessage," session)
print "Error inserting first content page"
print err_msg
exit
}

file_list = "section1.wp5 section2.wp5 section3.wp5 section4.wp5"
```

```
count = split(file_list, file, " ")
for (i=1; i<=count; i++) {
err_flag = dmAPIExec("insertfile,"session","doc_id","file[i]"," i-1)
if (err_flag == 0) {
err_msg = dmAPIGet("getmessage," session)
print "Error inserting content page " i-1 `to the document object'
print err_msg
exit
}
}

err_flag = dmAPIExec("save," session "," doc_id)
if (err_flag == 0) {
err_msg = dmAPIGet("getmessage," session)
print "Error saving the document object"
print err_msg
}
exit
```

# Insertpart

## Purpose

Inserts a component into a virtual document.

## Syntax

```
dmAPIGet("insertpart,session,document_id,component_id
[,version_label],containment_id|order_no[,orderno_flag]
[,use_node_ver_label][,follow_assembly][,copy_child]
[,containment_type,containment_desc]")
```

## Arguments

**Table 2-87. Insertpart method arguments**

| Argument | Description |
|---|---|
| *session* | Identifies an open repository session. |
| *document_id* | Identifies the virtual document into which you are inserting the new component. Use the object's ID or an indirect reference (*@object_id*) that points to the object. |
| *component_id* | Identifies the component you are inserting into the virtual document. Use the component's object ID or its chronicle ID. (Refer to the Usage notes for a definition of a chronicle ID.)<br><br>A component can be any SysObject or SysObject subtype except a folder, cabinet, or folder or cabinet subtype. |
| *version_label* | Specifies which version of the new component you want to insert. You can use either an implicit or symbolic label. (Refer to the Usage notes for information about using version labels.) |
| *containment_id* | Defines the inserted component's position in the document. Use the containment ID of the component currently at the position where you want to insert the new component. (Refer to the Usage notes for more information about containment IDs.)<br><br>If you include this argument, do not use the *order_no* or *orderno_flag* arguments. |

| Argument | Description |
|---|---|
| *order_no* | Defines the inserted component's position in the document. Use this option only if your application defines and manages the order numbers for virtual document components itself. Specify an integer number for *order_no*. If you include *order_no*, you must set the orderno_flag to T and you cannot provide a *containment_id* value. |
| *orderno_flag* | Indicates whether the preceding argument's value is a user-defined order number for the inserted component. You must set this to T (TRUE) if the preceding argument is the *order_no* option. The default value for this flag is F (FALSE), indicating that the preceding argument provides a containment ID value. |
| *use_node_ver _label* | Determines how the server chooses the late-bound descendants of the component during assembly. This argument is only useful if the component is a virtual document.<br><br>T (TRUE) directs the server to use the component's early-bound symbolic label to resolve late-bound descendants. F (FALSE) directs the server to resolve the component's descendants using the late-bound version label specified in the IN DOCUMENT clause. (Refer to the Usage notes for an expanded explanation.) |
| *follow_assembly* | Determines whether the server uses the containment objects or the component's assembly to find the component's descendants. This argument is only useful if the component is a virtual document.<br><br>T (TRUE) directs the server to use the component's assembly (if it has an assembly). F (FALSE) directs the server to use the containment objects. |
| *copy_child* | Determines whether the Documentum client copies or references the component when a user copies the containing virtual document. This is an integer attribute. Valid values are:<br><br>**0**, which lets users decide at the time the copy operation is requested<br><br>**1**, which directs the client to reference the component<br><br>**2**, which directs the client to copy the component<br><br>The default value is 0.<br><br>**Note:** The reference is an internal pointer. Referencing the component does not create a dm_reference object. |

| Argument | Description |
|---|---|
| | This argument is only useful if the component is itself a virtual document. |
| *containment_type* | Used only if the containing virtual document (identified in *document_id*) is an XML document. This argument identifies what type of reference to the component is found in the containing document.<br><br>If you include this, you must also include *containment_desc*.<br><br>This argument is used primarily by the Documentum clients. Refer to XML support, page 285 for details. |
| *containment_desc* | Used only if the containing virtual document (identified in *document_id*) is an XML document. This argument identifies the text of reference to the component in the containing document.<br><br>If you include this, you must also include *containment_type*.<br><br>This argument is used primarily by the Documentum clients. Refer to XML support, page 285 for details. |

## Return value

The Insertpart method returns the object ID of the containment object that describes the relationship between the component and the virtual document.

## Usage notes

The Insertpart method inserts a new component into the ordered list of components in a virtual document.

You must save (or check in) the virtual document after you insert a new component. Otherwise, the new component is not saved as part of the virtual document. Saving a virtual document requires Write permission on the document. Checking it in as a new version requires Version permission on the document.

If you are adding large numbers of components to a virtual document, it is recommended that you add components and save in small batches, or use the Appendpart method. Content Server can only add a certain number of new components between any two existing components before a Save must be executed to reorder the components. When

the document is saved, the components are re-ordered and you can continue inserting more parts (until the limit is reached again and another Save must be executed).

## Identifying the new component

Identify the component you want to append by using either its object ID or the value in its chronicle ID attribute and, optionally, a version label. The object ID is found in the object's r_object_id attribute. The chronicle ID, which identifies the root (original) version of the object, is found in the i_chronicle_id attribute. (If an object has no versions, then its r_object_id and i_chronicle_id attributes have the same value.)

### Early binding

If you specify a version label, the server searches the version tree associated with the object specified in *component_id*, finds that version, and binds it to the virtual document.

You can specify either an implicit or symbolic version label. If you specify an implicit label, there is an absolute link between that version and the virtual document. If you specify a symbolic label, then whatever version of the object carries the label is linked to your virtual document. (This means that the version that appears in your document may change if the symbolic label is moved from one version to another. Refer to Absolute links, page 166 and Symbolic links, page 166 in *Content Server Fundamentals* for a more detailed discussion of absolute and symbolic linking.)

### Late binding

If you specify only the object ID or the chronicle ID (with no version label), then the server associates the component's entire version tree with the virtual document. Later, when you assemble the document, you must identify which version of the object you want to include. This feature, called late binding, allows you to assemble virtual documents conditionally.

## Defining the component's position

Generally, to define a new component's position in the virtual document, you specify the containment ID of the component currently in the desired position. For example, suppose a virtual document has three components and that you want to insert a new component between the first and second components. To do so, you specify the containment ID of the component currently in the second position.

A component's containment ID is the object ID of the containment object that links the component to the virtual document. You can obtain this ID with the following query:

```
SELECT r_object_id, CONTAIN_ID FROM dm_sysobject
IN DOCUMENT ID('virtual_doc_id')
```

This returns a list of the object IDs and containment IDs for all the direct components of the virtual document.

In some cases, you may be generating and managing a user-defined numbering system for the components of a virtual document. If this is true, when you insert a component, use the *order_no* argument to specify an order number appropriate for your system.

You must also set the *orderno_flag* to T. The *order_no* argument can accept any integer value. The *orderno_flag* informs the server that you are providing a user-defined order number rather than a containment ID.

## Defining assembly behavior

If the new component is a virtual document, the *use_node_ver_label* and *follow_assembly* arguments let you define how the server chooses the component's descendants when the virtual document that contains the component is assembled.

### Use_node_ver_label

The *use_node_ver_label* argument sets the use_node_ver_label attribute in the containment object for the component.

If use_node_ver_label is TRUE for the component and the component is early bound to the virtual document, the server resolves any late-bound descendants of the component using the early-binding label specified for the component. It ignores any binding condition specified at the time of assembly. If an early-bound descendant is found that has use_node_ver_label set to TRUE, then that descendant's label is used to resolve descendants from that point in the hierarchy.

If use_node_ver_label is FALSE or if the component is late bound to the virtual document, the server resolves the component's late-bound descendants using the binding condition specified at the time of assembly.

(use_node_ver_label, page 167, of *Content Server Fundamentals* contains an expanded explanation of how use_node_ver_label affects the assembly of virtual documents.)

### Follow_assembly

The *follow_assembly* argument sets the follow_assembly attribute in the component's containment object. If the component has an associated assembly and follow_assembly is set to TRUE, the server will use the assembly rather than the containment objects to determine the component's descendants. In these cases, any binding conditions for the descendants are ignored. The descendants are taken from the assembly.

If follow_assembly is FALSE or if the component has no assembly, the server uses the containment objects and binding specifications to determine the component's descendants.

## Defining copy behavior

Setting the *copy_child* argument sets the copy_child attribute in the component's containment object. The copy_child attribute controls how the Documentum client handles the component when users copy the containing virtual document. The client has the option to either copy or reference the component.

Copy_child is an integer attribute with three valid values:

- 0, which means that users make the decision to copy or reference the component when they request the operation

- 1, which directs the client to reference the component in the new copy
- 2, which directs the client to copy the component for the new copy

The default value is 0.

**Note:** The reference is an internal pointer. It is not a dm_reference object.

### XML support

The *containment_type* and *containment_desc* arguments are used internally by the DFC to correctly set the references in the containing document when the component is inserted into the containing document. The arguments set the a_contain_type and a_contain_desc attributes in the containment object.

Although you can use these arguments to set the attributes to values you choose, if you do, the behavior of the XML document when manipulated using DesktopClient or the DFC is undefined.

For more information about the values set by these arguments and their use, refer to XML support, page 165 in *Content Server Fundamentals*.

## Related methods

Appendpart, page 84
Removepart, page 364
Updatepart, page 486

## Example

```
Chap_1 = dmget("create,s0,dm_document");
if (Chap_1 == NULL)
{  err_mess = dmget("getmessage,s0")
   print err_mess
   exit
}

status = dmset("set,s0,last,object_name", "Chapter1")

status = dmexec("save,s0,last")
if (status != 1)
{  err_mess = dmget("getmessage,s0")
   print err_mess
   exit
}

Chap_2 = dmget("create,s0,dm_document");
if (Chap_2 == NULL)
{  err_mess = dmget("getmessage,s0")
   print err_mess
   exit
}

status = dmset("set,s0,last,object_name", "Chapter2")
```

```
status = dmexec("save,s0,last")
if (status != 1)
{  err_mess = dmget("getmessage,s0")
   print err_mess
   exit
}
```

```
parent_id = dmget("create,s0,dm_document");
if (parent_id == NULL)
{  err_mess = dmget("getmessage,s0")
   print err_mess
   exit
}
```

```
status = dmset("set,s0,last,object_name", "Book")
cont_1 = dmget("appendpart,s0,last," Chap_1 ",T,F,2")
cont_2 = dmget("insertpart,s0,last," Chap_2 ",CURRENT," cont_1)
if (cont_2 == NULL)
{  err_mess = dmget("getmessage,s0")
   print err_mess
   exit
}
```

```
  status = dmexec("save,s0,last")
  if (status != 1)
  {  err_mess = dmget("getmessage,s0")
     print err_mess
     exit
  }
```

# Insertstate

## Purpose

Inserts a state into a lifecycle.

## Syntax

```
dmAPIGet("insertstate,session,policy_id
[,position|state_name]")
```

## Arguments

**Table 2-88. Insertstate method arguments**

| Argument | Description |
|---|---|
| *session* | Identifies an open repository session. |
| *policy_id* | Identifies the lifecycle to which you are inserting the new state. Use the dm_policy object's object ID. |
| *position* | Identifies the position of the state into which the new state will be inserted. |
| *state_name* | Identifies the name of the state before which to insert the new state. |

## Return value

The Insertstate method returns the position of the new state if successful or -1 if unsuccessful.

## Usage notes

The Insertstate method inserts a new state in a lifecycle at the position specified by the user. The server automatically renumbers the values for the state numbers at the specified position and below.

If you do not specify a state name or position, the new state is inserted before the base state, creating a new base state.

The Insertstate method inserts rows into the repeating attributes that define the lifecycle. You must explicitly set the values of the attributes that do not have default values.

## Related methods

Removestate, page 373

## Example

```
StatePos = dmget("insertstate," session
",4600007b8000567f,0")
```

# Install

## Purpose

Installs a validated lifecycle, workflow, or activity definition.

## Syntax

```
dmAPIExec("install,session,object_id[,notify_flag]")
dmAPIExec("install,session,object_id[,install_activity]
[,resume]")
```

## Arguments

**Table 2-89. Install method arguments**

| Argument | Description |
|---|---|
| *session* | Identifies an open repository session. |
| *object_id* | Identifies the lifecycle, activity or process definition that you want to install. Use the object ID of the lifecycle, activity, or process definition object. |
| *notify_flag* | A Boolean flag that indicates whether to inform owners of SysObjects associated with this policy. The default is F (FALSE). |
| *install_activity* | A Boolean flag that indicates whether to try to install a process definition's referenced activity definitions, if necessary. The default is F (FALSE). |
| *resume* | Indicates whether to resume or abort existing workflows that refer to this process definition. The default is F (FALSE). |

## Return value

The Install method returns TRUE if successful or FALSE if unsuccessful.

## Usage notes

The Install method installs a validated lifecycle, process or activity. The Install method requires the Relate permission or Sysadmin or Superuser privileges.

The Install method changes the definition's state to Installed.

### Installing process definitions

For process definitions, the Install method verifies that all the referenced activity definitions are already installed. If install_activity is not set or is set to FALSE, the Install method reports errors about any uninstalled activity definitions which it finds. Otherwise, the Install method tries to install any uninstalled activity definitions.

The Install method does not validate any draft activity definitions.

The Install method locates existing workflows which are run-time instances of the specified process definition and either restarts or aborts those workflows according to the setting of *resume*.

### Installing lifecycles

If you specify notify_flag=TRUE, the server sends mail to each owner of the SysObjects associated with the policy.

## Related methods

Uninstall, page 472

## Example

The following example installs a lifecycle.

```
if (dmAPIExec("install," session "," policy_id) == 0) {
print "Error installing lifecycle policy '" policy_id "'"
print dmAPIGet ("getmessage," session)
}
```

# Invalidate

## Purpose

Changes the state of a validated workflow definition or activity definition to draft.

## Syntax

```
dmAPIExec("invalidate,session,object_id")
```

## Arguments

**Table 2-90. Invalidate method arguments**

| Argument | Description |
|---|---|
| *session* | Identifies an open repository session. |
| *object_id* | Identifies the lifecycle, activity or process definition that you want to invalidate. Use the object ID of the dm_policy, dm_activity, or dm_process object. |

## Return value

The Install method returns TRUE if successful or FALSE if unsuccessful.

## Usage notes

The Invalidate method changes a validated workflow definition or activity definition to the draft state.

The Invalidate method requires the Relate permission or Sysadmin or Superuser privileges.

## Related methods

Validate, page 494
Uninstall, page 472

## Example

The following example invalidates a lifecycle.

```
if (dmAPIExec("invalidate," session "," process_id) == 0) {
print "Error invalidate workflow_def'" process_id "'"
print dmAPIGet ("getmessage," session)
}
```

# Iscached

## Purpose

Indicates whether a particular object type is in the DMCL type cache.

## Syntax

```
dmAPIGet("iscached,session,typecache,cache_key")
```

## Arguments

**Table 2-91. Iscached method arguments**

| Argument | Description |
|---|---|
| *session* | Identifies an open repository session. |
| *cache_key* | Identifies the type you are checking. Use the type's internal name; for example, dm_document. |

## Return value

The Iscached method returns TRUE if the specified type is in the cache and FALSE if it is not.

## Usage notes

When changes are made to a type's definition, Content Server's type cache is not updated for current sessions. Use the Iscached method to tell you whether a type you have changed is in the server's type cache. If the answer is TRUE, you may want to restart the session to make the change visible to the server.

## Related methods

None

## Example

```
result=dmAPIGet("iscached,c,typecache,dm_document")
if(result=="T")
print("Type is cached");
else
print("Type is not cached");
```

# Kill

## Purpose

Terminates a repository session or flushes an object from the DMCL cache.

## Syntax

```
dmAPIExec("kill,session,session_to_kill[,immediacy_level]
[,message]")

dmAPIExec("kill,session,object_id")
```

## Arguments

**Table 2-92. Kill method arguments**

| Argument | Description |
|---|---|
| *session* | Identifies the current repository session. |
| *session_to_kill* | Identifies the repository session you want to terminate. Use the session's identifier. (You can obtain the identifier using the LIST_SESSIONS function.) |
| *immediacy_level* | Defines the immediacy or impact to the end user of the terminated session. This is a character string argument. Valid values are:<br><br>*nice*, which terminates the session when there are no more open transactions and collections<br><br>*after current request*, which terminates the session when it is finished executing the current request<br><br>*unsafe*, which terminates the session immediately |
| *message* | A user-defined message that the server will place in the session log. The message can be up to 255 characters. |
| *object_id* | Specifies an object to flush from the dmcl cache. |

## Return value

The Kill method returns TRUE if successful or FALSE if unsuccessful.

## Usage notes

You must have Superuser user privileges to the use the Kill method.

### Terminating sessions

Terminating a session nicely, after all transactions and collections are closed is the least disruptive. Users and client are functionally unaffected, although users may notice temporary performance degradation.

Terminating a session after the current request completes is safe and faster than waiting for all transactions and collections to close. However, some transactions may not complete and connections may be lost.

Terminating a session immediately, without regard for the status of transactions is unsafe. If you stop a session the *immediacy_level* set to unsafe, you should monitor the server for stability. For example, try to connect, look at some documents, or check the server log. If the server appears compromised, shut down the server.

### Flushing the DMCL cache

You can use the Kill method to flush an object from the DMCL cache. The client then re-fetches the object. Use this to refresh an object's attributes in a client application display.

## Related methods

Shutdown, page 447

## Example

```
status = dmexec("kill," session ",0100007b80006de0,nice,
testing kill method")
```

# Link

## Purpose

Associates an object with a folder or cabinet.

## Syntax

```
dmAPIExec("link,session,object_id,
folder_specification|alias")
```

## Arguments

**Table 2-93. Link method arguments**

| Argument | Description |
|---|---|
| *session* | Identifies an open repository session. |
| *object_id* | Specifies the object that you want to link to the specified folder. This can be any object of type SysObject or its subtypes except cabinets.<br><br>Use the object's ID or an indirect reference (*@object_id*) that points to the object. |
| *folder_specification* | Defines the folder or cabinet to which you want to link the object. You can use either the folder or cabinet's object ID or its folder path. This path has the format:<br><br>/*cabinet_name*{/*folder_name*} |
| *alias* | Defines an alias in place of *folder_specification*. The alias is resolved when method is executed. This argument is intended for use in applications. The format of an alias is:<br><br>%[*alias_set_name*.]*alias_name*<br><br>*alias_set_name* is the name of an alias set object.<br><br>*alias_name* is a value in the alias_name attribute of the alias set object.<br><br>For information about aliases, refer to Appendix A, Aliases of the *EMC Documentum Object Reference Manual*. |

## Return value

The Link method returns TRUE if successful or FALSE if unsuccessful.

## Usage notes

The first link that you perform for an object defines its primary cabinet; this is its primary link. If you do not explicitly link a new object to a folder or cabinet before you save it, the server automatically links the object to your default cabinet when you save it.

After the first link that establishes an object's primary cabinet, you can link it to other cabinets or folders. These links are called secondary links.

The method uses the repository named in the docbase_scope attribute of the session config object to identify the repository in which to search for the folder specified in the *folder_specification*. If the folder is not found in that repository, the method returns an error.

Using the Link method has the following constraints:

- To create a primary link, you must have at least Write permission for the object and Change Location permission on the object.
- To create a secondary link for the object:
  - You must have at least Browse access to the object.
  - You must have Change Location permission on the object.
- If the repository is running under folder security, to create either a primary or a secondary link, you must have at least Write permission for the folder or cabinet to which you are linking the object.
- You can link any SysObject or SysObject subtype except cabinets to a folder or cabinet.

## Related methods

Unlink, page 474

## Example

This example:

- Fetches the specified folder object
- Unlinks the folder from its cabinet and links to another cabinet
- Saves the folder object

```
err_flag = dmAPIExec("fetch," session "," fldr)
if (err_flag == 0) {
err_msg = dmAPIGet("getmessage," session)
print "Error fetching folder '" fldr "'"
print err_msg
exit
```

```
}

err_flag = dmAPIExec("unlink," session "," fldr "," cab1)
if (err_flag == 0) {
err_msg = dmAPIGet("getmessage," session)
print "Error unlinking folder '" fldr "' from cabinet '" cab1 "'"
print err_msg
exit
}

cerr_flag = dmAPIExec("link," session "," fldr "," cab2)
if (err_flag == 0) {
err_msg = dmAPIGet("getmessage," session)
print "Error linking folder '" fldr "' to cabinet '" cab2 "'"
print err_msg
exit
}

err_flag = dmAPIExec("save," session "," fldr)
if (err_flag == 0) {
err_msg = dmAPIGet("getmessage," session)
print "Error saving folder '" fldr "'"
print err_msg
exit
}
```

# Listconnection

## Purpose

Returns the subconnections and the repositories to which they are connected for a specified session.

## Syntax

```
dmAPIGet ("listconnection,session")
```

## Arguments

**Table 2-94. Listconnection method arguments**

| Argument | Description |
|---|---|
| *session* | Identifies an open repository session. Use a session identifier in the format S*n*. For example: S0 or S1.<br><br>Do not specify a subconnection. |

## Return value

The Listconnection method returns a collection whose attributes contain the identifiers for each subconnection in the specified session and the repository to which the subconnection is connected.

## Usage notes

Listconnection and Dumpconnection return the same information for a specified session. Listconnection differs from Dumpconnection in that Listconnection returns a collection which you must iterate through to obtain the subconnection information and you must specify a session. Listconnection cannot return information about all sessions open with Content Server in one collection.

## Related methods

Dumpconnection, page 196
Getconnection, page 229

## Example

The following example uses IAPI to execute the method to show its output. This example supposes that there are two active connections when the method is executed.

```
API>listconnection,c
. . .
q0
API>next,c,q0
. . .
OK
API>dump,c,q0
. . .
USER ATTRIBUTES

connection_name:s0c0
connection_id:010015b38000c103

SYSTEM ATTRIBUTES

r_docbase_id: 5552
r_docbase_name: test_repository
r_user_name: emily

APPLICATION ATTRIBUTES

INTERNAL ATTRIBUTES

API>next,c,q0
. . .
OK
API>dump,c,q0
. . .
USER ATTRIBUTES

connection_name:s0c1
connection_id:010015b80007107

SYSTEM ATTRIBUTES


r_docbase_id: 5524
r_docbase_name: mktg_repository
r_user_name: emily

APPLICATION ATTRIBUTES

INTERNAL ATTRIBUTES
```

# Listmessage

## Purpose

Returns all the messages generated by a session which are at or above a specified severity level.

## Syntax

```
dmAPIGet("listmessage,session[,severity_level]")
```

## Arguments

**Table 2-95. Listmessage method arguments**

| Argument | Description |
|---|---|
| *session* | Identifies an open repository session. |
| *severity_level* | Defines the minimum severity level of the messages you receive. You receive all messages at or above the specified level. Specify the level as an integer, corresponding to one of the following severity levels:<br><br>1, for Information messages<br>2, for Warning messages<br>3, for Error messages<br>4, for Fatal error messages<br><br>This is optional. If unspecified, the default is 1 (informational messages). |

## Return value

The Listmessage method returns the collection identifier of a collection of objects. These objects have three attributes that are generally useful to users:

- CODE, which is an integer datatype
- SEVERITY, which is an integer datatype
- MESSAGE, which is a string(255) datatype

Three other attributes, LOGGED_FLAG, COUNT, and STAMP are also returned. These attributes are not generally useful to end users. LOGGED_FLAG indicates whether the

error has been logged, COUNT indicates how many substitutions were made into the body of the error message, and STAMP is always blank.

## Usage notes

The CODE attribute contains the actual error number.

The SEVERITY attribute contains the severity level of the message specified as an integer (refer to the argument descriptions for a list of severity levels and corresponding integers).

The MESSAGE attribute contains the text of the error or message.

Messages that you retrieve using the Listmessage method cannot be subsequently obtained using the Getmessage method.

After you execute one Listmessage method, you cannot execute another until you close the collection generated by the first Listmessage execution.

## Related methods

Getmessage, page 251

## Example

```
if (dmAPIExec("fetch," session "," doc_id) == 0) {
# Ignore information messages
col1 = dmAPIGet("listmessage," session ",2")
while (dmAPIExec("next," session "," col1))
{
msg = dmAPIGet("get," session "," col1 ",MESSAGE")
print msg
}
dmAPIExec("close," session "," col1)
```

# Locate

## Purpose

Searches through a repeating attribute and returns the index value that corresponds to a specified value in the attribute.

## Syntax

```
dmAPIGet("locate,session,object_id,attribute,value")
```

## Arguments

**Table 2-96. Locate method arguments**

| Argument | Description |
|---|---|
| *session* | Identifies an open repository session. |
| *object_id* | Identifies the object that contains the specified attribute. Use the object's ID or an indirect reference (*@object_id*) that points to the object. |
| *attribute* | Names the repeating attribute. |
| *value* | Defines the repeating attribute value for which you are trying to find the corresponding index value. |

## Return value

The Locate method returns the index value associated with the specified attribute value. If the specified value does not exist in the attribute, the method returns -1.

## Usage notes

Use the Locate method only for persistent objects.

You cannot use the Locate method to return index values for the special attributes. The Locate method operates only on an object's base attributes.

## Related methods

Get, page 224

## Example

The following example finds a specific author in the authors attribute of a document.

```
if((index=dmAPIGet("locate," session "," doc_id ",authors,"\ name))==-1) {
if ((err_msg = dmAPIGet("getmessage," session)) != NULL) {
print "Error locating author '"name"' in doc '"doc_id"'
print err_msg
exit
}
}
print "Author '"name"' is author number "index
```

# Lock

## Purpose

Places a database lock on an object.

## Syntax

```
dmAPIExec("lock,session,object_id[,validate_stamp]")
```

## Arguments

**Table 2-97. Lock method arguments**

| Argument | Description |
|---|---|
| *session* | An open repository session. |
| *object_id* | Object on which to place the database lock. Use the object's repository object ID. |
| *validate_stamp* | Whether to validate the version stamp on an object. The default value is F (FALSE). Refer to the Usage notes for details of the use of this argument. |

## Return value

The Lock method returns TRUE if successful or FALSE if unsuccessful.

## Usage notes

The Lock method places an exclusive lock at the database level on an object. A user with Sysadmin or Superuser user privileges can place a lock on any object. Other users can place locks only on objects for which they have at least Version permission.

This method can only be executed inside an explicit transaction begun with a DQL BEGINTRAN statement or a Begintran method. The lock is released when the transaction is closed with either a COMMIT or ABORT statement or a Commit or Abort method.

The *validate_stamp* argument determines whether the version stamp of the object in the database is checked against the version stamp of the same object in the dmcl cache. The argument is false by default, which means the version stamp is not checked. If you set the argument to T (true), the method checks whether the object is in the dmcl cache and

if so, checks the version stamp of the object in the database against the version stamp of the cached object. If the version stamps do not match, the method reports an error. However, the operation is not aborted.

If the object is already locked, the method is blocked until the object is unlocked and the method can take the lock.

## Related methods

None

## Example

This example places a database lock on the document identified by the object ID 0900002c04761ca9:

```
dmAPIExec("lock,s0,0900002c04761ca9")
```

# Lpq

## Purpose

Retrieves the print queue for a specified printer.

## Syntax

```
dmAPIGet("lpq,session[,printer]")
```

## Arguments

**Table 2-98. Lpq method arguments**

| Argument | Description |
|---|---|
| *session* | Identifies an open repository session. |
| *printer* | Identifies the printer whose queue you want to see. Use the object name of the output device object that represents the printer. If you do not specify a printer, the default is the system printer. |

## Return value

The Lpq method returns a formatted string that contains a listing of the printing jobs in the print queue. The string contains a line feed between each job listing.

## Usage notes

Part of the information returned for each print job is its print job number. You can use this number to stop the print job with the Unprint method.

## Related methods

Print, page 321
Unprint, page 482

## Example

```
print "The current print queue status is: "
print dmAPIGet("lpq," session ",engr_printer")
```

# Mark

## Purpose

Assigns one or more symbolic version labels to an object.

## Syntax

```
dmAPIExec("mark,session,object_id,label{,label}")
```

## Arguments

**Table 2-99. Mark method arguments**

| Argument | Description |
|---|---|
| *session* | Identifies an open repository session. |
| *object_id* | Specifies the object to which you want to assign the label. Use the object's ID or an indirect reference (*@object_id*) that points to the object. |
| *label* | Defines the label that you want to assign to the object. You can specify one implicit label and/or one or more symbolic labels. |

## Return value

The Mark method returns TRUE if successful or FALSE if unsuccessful.

## Usage notes

Before you can use the Mark method, you must either check out or fetch the specified object. After you execute the Mark method, you must check in or save the object to keep the new label or labels.

If you are setting an implicit label on a virtual document, you must check out the document before issuing the Mark method. You cannot fetch a virtual document and set its implicit label. Doing so breaks the connections to the document's components.

If you specify an implicit label, you must specify the label in the format:

```
n[.x]
```

or

*n*.0

where *n* and *x* are an integers.

If you specify a symbolic label that is already assigned to another version of the object, the method moves the label from that other version to the object specified. However, if the version initially holding the label is locked, the method fails because it cannot access the object to move the label.

If you audit the Mark method, Content Server creates one audit trail entry for each version label included in the arguments.

## Related methods

Unmark, page 480

## Example

```
if (dmAPIExec("ckeckout," session "," doc_id) == 0) {
print "Error checking out the document"
print dmAPIGet("getmessage," session)
exit
}
if (dmAPIExec("mark," session "," doc_id ",APPROVED") == 0) {
print "Error marking doc '" doc_id "' approved"
print dmAPIGet("getmessage," session)
exit

if ((new_id = dmAPIGet("checkin," session "," doc_id)) == NULL) {
print "Error checking in document '" doc_id "'"
print dmAPIGet("getmessage," session)
exit
}
```

# Mount

## Purpose

Sets the content location for an external store object.

## Syntax

```
dmAPIExec("mount,session,object_id,path")
```

## Arguments

**Table 2-100. Mount method arguments**

| Argument | Description |
|---|---|
| session | Identifies an open repository session. |
| object_id | Identifies the object ID of the external file store. |
| path | Specifies the path of the external file store. |

## Return value

The Mount method returns TRUE if successful and FALSE if unsuccessful.

## Usage notes

The Mount method sets the content location for an external store object. Use the Mount method to change the path to content in an external file store dynamically. For example, if you set up external file storage for access to files on a CD-ROM, you can move the CD-ROM to a local client machine and use the Mount command to correct the path.

For client configurations where the path name specified in the client root location is invalid or inaccessible, use the Mount method to remap the client root location dynamically when executing the plug-in on the client.

If a client application does not execute the Mount method but issues a Getfile or a Getcontent method, the content is retrieved on the server side if a server-side plug-in is available and returned to the client application.

## Related methods

Getpath, page 253
Setpath, page 439

## Example

```
status = dmexec("mount," session ",6100007b80000105,
c:\dm\data\external")
```

# Movestate

## Purpose

Rearranges the state definitions within a lifecycle definition.

## Syntax

```
dmAPIExec("movestate,session,object_id,old_position,
new_position")
```

## Arguments

**Table 2-101. Movestate method arguments**

| Argument | Description |
|---|---|
| *session* | Identifies an open repository session. |
| *object_id* | Identifies the lifecycle that you want to modify. Use the dm_policy's object ID. |
| *old_position* | The current position of the state. |
| *new_position* | The new position in which to put the state. |

## Return value

The Movestate method returns TRUE if successful or FALSE if unsuccessful.

## Usage notes

The Movestate method moves a state definition within a lifecycle (a buisness policy object).

You cannot use the Movestate method on an installed lifecycle, and you cannot alter a lifecycle that is attached to any SysObjects.

The Movestate method changes the policy's state to Draft and renumbers the states at the new position and below in the lifecycle.

## Related methods

Appendstate, page 90
Insertstate, page 287
Removestate, page 373

# Next

## Purpose

Returns objects from collections.

## Syntax

```
dmAPIExec("next,session,collection_identifier")
```

## Arguments

**Table 2-102. Next method arguments**

| Argument | Description |
|---|---|
| *session* | Identifies an open repository session. |
| *collection_identifier* | Identifies the collection whose results you are processing. The value that you use here is returned by the method that generates the collection, for example, the Readquery method. |

## Return value

The Next method returns TRUE if successful or FALSE if unsuccessful.

## Usage notes

The Next method is used to retrieve query result objects in a collection that results from a DQL statement. It also returns the objects in the collection generated by an Assemble method call.

When you execute the Next method to retrieve query results, the server sends the values from one query result object to the client application. You must execute the Next once for each query result in the collection to retrieve all the values. (Refer to Chapter 4, Using DQL, in *Content Server Fundamentals* for more information about processing queries.)

When you execute the Next method to retrieve the objects in a collection generated by the Assemble method, Next retrieves some or all the objects with each execution. The number retrieved is determined by an argument in the Assemble method call. Refer to Chapter 7, Virtual Documents, in *Content Server Fundamentals* for information about using Next to create an assembly.

## Related methods

Assemble, page 100
Query, page 337

## Example

This example:

- Queries the repository
- Loops through the resulting collection and prints each respondent
- Closes the query collection

```
query_id = \
dmAPIGet("query," session ",select r_object_id from dm_document)
if (query_id == NULL) {
err_msg = dmAPIGet("getmessage," session)
print "Error querying the repository"
print err_msg
exit
}

resp_cntr = 0
while( dmAPIExec("next," session "," query_id) > 0 ) {
obj_id = dmAPIGet("get," session "," query_id ",r_object_id")
if (obj_id == NULL) {
err_msg = dmAPIGet("getmessage," session)
print "Error retrieving object ID from the query collection"
print err_msg
exit
}
print ++resp_cntr " : " obj_id
}

err_flag = dmAPIExec("close," session "," query_id)
if (err_flag == 0) {
err_msg = dmAPIGet("getmessage," session)
print "Error closing the query collection"
print err_msg
exit
}
```

# Offset

## Purpose

Determines an attribute's position in the list of attributes that belong to an object.

## Syntax

```
dmAPIGet("offset,session,object_id,attribute")
```

## Arguments

**Table 2-103. Offset method arguments**

| Argument | Description |
|---|---|
| *session* | Identifies an open repository session. |
| *object_id* | Specifies the object that contains the specified attribute. Use the object's ID or an indirect reference (*@object_id*) that points to the object. |
| *attribute* | Identifies the attribute whose offset you are determining. Use the attribute's name to identify the attribute. |

## Return value

The Offset method returns an integer that represents the attribute's position in the ordered list of the object's attributes.

## Usage notes

You can use the returned integer as an index value with the special attributes. Each object has a set of the special attributes. These are attributes that contain information about the object and its attributes. The special attributes that contain information about the object's attributes are repeating attributes. All of the rows at a particular index level describe one object attribute. You can use the Offset method to obtain an attribute's position in the list of object attributes; this number is also the index value of the rows that describe that attribute in the special attributes. For example, if the Offset method returns the number 3, then you can determine the datatype of the attribute by looking at the special attribute _datatype at the index level 3: _datatype[3].

## Related methods

None

## Example

This example uses the Offset method to determine the attribute number of the authors attribute. The attribute number is then used to find out the number of values that authors contains. You can also use this offset number to retrieve the datatype and length with the _datatype and _length special attributes, respectively.

```
index = dmAPIGet(sprintf("offset,%s,%s,authors",session,doc_id));
cmd = sprintf("get,%s,%s,_values[%s]",session,doc_id,index);
num_authors = dmAPIGet(cmd);
```

# Pause

## Purpose

Suspends a work item.

## Syntax

```
dmAPIExec("pause,session,workitem_id")
```

## Arguments

**Table 2-104. Pause method arguments**

| Argument | Description |
|---|---|
| *session* | Identifies an open repository session. |
| *workitem_id* | Specifies the work item that you want to pause. |

## Return value

The Pause method returns TRUE if successful or FALSE if unsuccessful.

## Usage notes

Using the Pause method has the following constraints:

- You must be the workflow supervisor or a user with Sysadmin or Superuser privileges.
- The work item must be in the dormant state.

Work item performers cannot pause their work items.

## Related methods

Halt, page 265

## Example

```
status = dmAPIExec("pause," session ",4a00007b80000100")
```

# Print

## Purpose

Prints a specified document.

## Syntax

```
dmAPIGet("print,session,object_id[,printer][,print_cover]
[,save_output][,num_copies][,starting_content_page]
[,ending_content_page]")
```

## Arguments

**Table 2-105. Print method arguments**

| Argument | Description |
|---|---|
| *session* | Identifies an open repository session. |
| *object_id* | Specifies the document that you want to print. Use the document's object ID or an indirect reference (*@object_id*) that points to the object. |
| *printer* | Specifies the printer that you want to use. Use the object name of the output device object that represents the printer. ID. If you do not specify a printer, the default is the system printer. |
| *print_cover* | Directs the printer to print a cover page. The default is T. If you do not want a cover page, specify F for this flag. |
| *save_output* | Directs the system to save the postscript output generated by the printing process. The default is T. Set this flag to F if you do not want to save the output. |
| *num_copies* | Defines the number of copies that you want to print. Use an integer number to specify this. The default value is one. |
| *starting_ content_page* | Defines the content file with which to start printing. If unspecified, the default is zero (representing the first content in the object). Use the content's page number. |
| *ending_ content_page* | Defines the content file at which to end printing. If unspecified, this defaults to the last content page. |

## Return value

The Print method returns the print job's number.

## Usage notes

**Note:** On Windows platforms, you must configure a printer in the repository before you can use the Print method. For instructions, refer toConfiguring a repository printer on Windows, page 81 in the *Content Server System Administrator's Guide*.

When you use the Print method, the server automatically converts the content to the page description language required by the printer. If the *save_output* flag is set to T (the default setting), then this page description language output is saved as a rendition with the content.

Every content associated with an object has a page number that defines the content's position within the ordered list of the object's contents. Use this number to refer to a particular content to define the starting and stopping points when printing a multiple content document.

**Note:** You can use the print job number returned by this method in the Unprint method, which removes a job from a print queue.

## Related methods

Lpq, page 308
Unprint, page 482

## Examples

The following examples shows a Print Method using default argument values.

```
job_id = dmAPIGet(sprintf("print,%s,%s,engr_printer", session, doc_id));
```

This example prints pages 1 to 5.

```
job_id =dmAPIGet(sprintf("print,%s,%s,engr_printer,T,,,%d,%d",\
session, doc_id, 1, 5));
```

# Promote

## Purpose

Promotes a SysObject from its current normal state to the next normal state defined in the attached lifecycle.

## Syntax

For unscheduled promotions:

```
dmAPIExec("promote,session,object_id[,to_state|to_position
,test_only_flag,override_flag]")
```

For scheduled promotions:

```
dmAPIExec("promote,session,object_id,scheduled_date
[,pattern],to_state|to_position|cancel[,override_flag]")
```

## Arguments

**Table 2-106. Promote method arguments**

| Argument | Description |
|---|---|
| *session* | Identifies an open repository session. |
| *object_id* | Identifies the SysObject to promote. Use the object's object ID. If you identify a replica object or use an indirect reference (*@objectid*), the source object is affected, not the local copy. |
| *test_only_flag* | TRUE or FALSE. When set to TRUE, Promote evaluates the entry criteria to determine if the document can be promoted. The default value is FALSE. |
| *override_flag* | TRUE or FALSE. When set to TRUE, Promote causes the transition to proceed even if the entry criteria of the destination state is not satisfied. Only users with Superuser user privileges or the lifecycle owner can specify this argument. The default is FALSE.<br><br>This flag is ignored if *test_only_flag* is set to TRUE. |
| *scheduled_date* | The date and time for which this promotion is scheduled. |
| *pattern* | The date-time format for *scheduled_date.* The pattern defaults to the client-side localized date-time format if not specified. |

| Argument | Description |
|---|---|
| *to_state* | Identifies, by name, the state to which to promote the object. This must be the state immediately following the current state; otherwise the promotion fails.<br><br>*to_state* and *to_position* are mutually exclusive.<br><br>If the *scheduled_date* is specified, you must include *to_state* or *to_position*. |
| *to_position* | Identifies, by position, the state to which to promote the object. The implementation is the same as that for *to_state*. |
| cancel | Cancels the scheduled promotion. To cancel a scheduled promotion, provide the original scheduled date. |

## Return value

The Promote method returns TRUE if successful or FALSE if unsuccessful.

## Usage notes

The Promote method promotes a SysObject from its current normal state to the next normal state defined in the lifecycle attached to the SysObject.

If you are the object's owner or a superuser, you must have Write permission to promote an object in a lifecycle. Other users must have Write permission and Change State permission for the object. If you have only Change State permission, Content Server attempts to promote the object as the user defined in the a_bpaction_run_as attribute in the docbase config object. In those instances, that user must be either the owner or superuser with Write permission or have Write and Change State permission on the object.

The current state must be the state previous to the state specified by *to_state* or *to_position*. The state to which you are promoting the object cannot be the terminal state and must not be an exception state.

If the requirements for the next state are not met, the method fails unless you specify *override_flag*= TRUE. If *override_flag* is set to TRUE and if the user is a lifecycle owner or has Superuser user privileges, the method sets r_current_state to destination_state and commits the change without performing the actions associated with the specified state.

### Testing before promotion

If *test_only_flag* is TRUE, the Promote method launches the bp_transition method to test the requirements of the next state.

### Scheduling promotion

If you specify a *scheduled_date* and do not specify cancel, the Promote method adds a new job object whose method_name is bp_transition and with proper values for bp_transition as arguments and due_time as scheduled_time. The object name for the job has the following format:

Bp*sysobj_id scheduled date*

The scheduled_date is in *YYYYMMDDHHMISS* format where
*YYYY* is a 4-digit year
*MM* is a 2-digit month
*DD* is a 2-digit day
*HH* is a 2-digit hour
*MI* is a 2-digit minute
*SS* is a 2-digit second

Note that you cannot schedule the same object for more than one state transition event at a specified time.

The bp_transition method is invoked by a job agent which periodically scans dm_job table to execute jobs which are overdue. The default scan time is 1 minute.

If you specify a scheduled_date and cancel is specified, the Promote method deletes the job with name equal to Bp*sysobj_id scheduled date.*

## Related methods

Demote, page 176

## Example

The following example promotes the object to state number 5 in the attached lifecycle, overriding the entry criteria:

```
status = dmexec("promote," session ",0900007b80000519,5,F,T")
```

The following example promotes the object to state number 5 in the attached lifecycle on June 4, 1999, at the time, overriding the entry criteria:

```
status = dmexec("promote," session ",0900007b80000519,
6/4/99 10:40:16 AM,,5,T")
```

The following example cancels the promotion to state number 5 in the attached lifecycle that was scheduled for June 4, 1999:

```
status = dmexec("promote," session ",0900007b80000519,
6/4/99 10:40:16 AM,,cancel")
```

# Prune

## Purpose

Removes unwanted versions of an object.

## Syntax

```
dmAPIExec("prune,session,object_id[,keepSLabel]")
```

## Arguments

**Table 2-107. Prune method arguments**

| Argument | Description |
|---|---|
| *session* | Identifies an open repository session. |
| *object_id* | Specifies the object whose version tree you want to prune. Use the object's ID or an indirect reference (*@object_id*) that points to the object. |
| *keepSLabel* | Directs the server to keep or remove versions having symbolic labels. The default is T. To remove versions that have symbolic labels, set this flag to F. |

## Return value

The Prune method returns TRUE if successful or FALSE if unsuccessful.

## Usage notes

You must have Delete permission for an object version that you want to prune. If the repository is running under folder security, you must also have at least Write permission for the object's primary folder. (An object's primary folder is the folder referenced in i_folder_id[0].)

The Prune method cleans up an object's version tree. A version tree begins with the original object and contains all the versions derived from the original. Using the Prune method, you can clean up the entire tree or some specified portion of it.

To prune the entire tree, specify the original object's ID in the method's argument list. (Every object has an attribute named i_chronicle_id that holds the ID of the object's original version. Query this attribute to obtain the original object's ID, if necessary.)

To prune a subset of the tree, specify the object ID of the object that begins the branch you want to prune. For example, assume that your tree begins with Object 1 and that Object 1 has two directly derived versions, Object 1A and Object 1B. If you want to prune the entire tree, you specify Object 1's ID. If you want to prune only those versions derived from version 1A, you specify the object ID of Object 1A.

If the object is a replica object, you must prune the entire tree.

Prune removes all versions that meet the following conditions:

- The version is not frozen.
- The version is not part of a virtual document.
- The version has no symbolic labels.
- The user has Delete permission on the version and, if folder security is turned on, the user has at least Write permission on the version's primary folder.
- The version does not have an unexpired retention period and is not associated with an active retainer object.

If you want to remove versions that have symbolic labels, set the *keepSLabel* flag to F. If you do this and the version with the CURRENT label is removed, the server automatically reassigns the CURRENT label to the closest remaining parent of the deleted version.

Any versions that remain on the version tree after pruning are not renumbered.

If the *object_id* identifies the root of the version tree and that version satisfies the pruning conditions, then the method will also remove that version if all other versions on the tree are removed. The original version is not removed if any node on the version tree remains in the repository.

If the compound_integrity attribute is defaulted or set to TRUE in the server's server config object, the system will not prune any version that is a component in a virtual document. Additionally, the server will not prune versions that are components in frozen virtual documents. (You cannot prune components of a frozen virtual document even if the compound_integrity attribute is set to FALSE.)

If the dm_prune event is audited, an event is generated for each object removed from the version tree.

## Related methods

Destroy, page 186

## Examples

This example creates a document, checks it out, saves a new version of it, and then prunes both versions.

```
cron_id = dmAPIGet("create,s0," dm_document)
dmAPIExec("save,s0," cron_id)
doc_id = dmAPIGet("checkout,s0," cron_id)
doc_id1 = dmAPIGet("checkin,s0," doc_id)
dmAPIExec("prune,s0," cron_id, "F")
```

The next example creates a document (cron_id) and two versions of that document (doc_id1 and doc_id2) and then prunes the original document's version tree. In this example, doc_id2 is a version of doc_id1, which is itself a version of cron_id. The operation removes only the version identified as doc_id1, because the version called doc_id2 has the symbolic label CURRENT.

```
cron_id = dmAPIGet("create,s0" dm_document)
dmAPIExec("save,s0," cron_id)
cron_id = dmAPIGet("checkout,s0," cron_id)
doc_id1 = dmAPIGet("checkin,s0," cron_id)
doc_id1 = dmAPIGet("checkout,s0," doc_id1)
doc_id2 = dmAPIGet("checkin,s0," doc_id1)
dmAPIExec("prune,s0," cron_id)
```

The final example creates a document and two versions of that document and then prunes a subset of versions on the tree. In this example, doc_id2 is a version of doc_id1, which is itself a version of cron_id. The argument doc_id1 in the Prune command identifies the start of the subset. In this example, doc_id1 is the only version removed, because the doc_id2 version has the symbolic label CURRENT and the original version (cron_id) is not part of the specified subset.

```
cron_id = dmAPIGet("create,s0" dm_document)
dmAPIExec("save,s0," cron_id)
cron_id = dmAPIGet("checkout,s0," cron_id)
doc_id1 = dmAPIGet("checkin,s0," cron_id)
doc_id1 = dmAPIGet("checkout,s0," doc_id1)
doc_id2 = dmAPIGet("checkin,s0," doc_id1)
dmAPIExec("prune,s0," doc_id1)
```

# Publish_dd

## Purpose

Publishes data dictionary information.

## Syntax

```
dmAPIExec("publish_dd,session[,locale][,type][,attribute]
[,force_publish]")
```

## Arguments

**Table 2-108. Publish_dd method arguments**

| Argument | Description |
|---|---|
| *session* | Identifies the current repository session. |
| *locale* | Identifies the locale for which you want to publish data dictionary information. You can specify a locale or the keyword DM_SESSION_DD_LOCALE. You can specify any locale found in the dd_locales attribute of the docbase config object.<br><br>Using the keyword directs the server to publish only the information for the current session locale.<br><br>If the argument is not included, the data dictionary information for all locales is published. |
| *type* | Identifies the object type for which you want to publish data dictionary information. Use the object type's internal name.<br><br>If this is not included, all data dictionary information for the specified locale is published. |

| Argument | Description |
|---|---|
| *attribute* | Identifies a specific attribute whose information you want to publish. If you include an attribute name, you must include the type argument, and the type argument must identify the object type for which the attribute is defined. |
| *force_publish* | Indicates whether to publish all data dictionary objects for the object type, attribute , or locales or only those whose resync_needed attribute is TRUE. This argument is FALSE by default. |

## Return value

The Publish_dd method returns TRUE if successful or FALSE if unsuccessful.

## Usage notes

The Publish_dd method creates or updates dd_type_info and dd_attr_info objects. A dd type info object contains the published data dictionary information for an object type in a particular locale. A dd attr info object contains the published data dictionary information for a specific attribute in a particular locale.

The method also updates or creates dd_common_info objects as needed. A dd common info object contains the published data dictionary information that is common to an object type and its attributes.

By default, the method publishes only the data dictionary objects that need to be refreshed. To force the method to publish all the data dictionary objects for the object type, attribute, or locales, set the force_publish argument to TRUE.

## Related methods

Type, page 464

## Example

```
Sub Entry_Point(ByVal DMT_DB As String, _
                ByVal DMT_USR As String, _
                ByVal DMT_USP As String, _
                ByVal type_name As String, _
                ByVal locale As String)
 sess_id$ = Connect(DMT_DB$, DMT_USR$, DMT_USP$)
'
```

```
  ' Publish Data Dictionary information for the
  ' specified type in the specified locale
 '
status = dmAPIExec("publish_dd," & sess_id & "," &
locale & "," & type_name)
  If (status = 0) Then
    Print "Error publishing Data Dictionary information
 for type " _
          & type_name & ", locale " & locale
    Call Err_Handle(sess_id, "0")
  Else Print "Successfully published Data Dictionary
information for type " _
             & type_name & ", locale " & locale
  End If
  Call Disconnect(sess_id$)
End Sub
```

# Purgelocal

## Purpose

Removes all files that have been copied into the local area during the current session.

## Syntax

```
dmAPIExec("purgelocal,session")
```

## Arguments

**Table 2-109. Purgelocal method arguments**

| Argument | Description |
|---|---|
| *session* | Identifies the current repository session. |

## Return value

The Purgelocal method returns TRUE if successful or FALSE if unsuccessful.

## Usage notes

The Purgelocal cleans up files in a client's local area. The server does not automatically clean up a local area when a session is terminated. Instead, this must be done with the Purgelocal method.

If the method encounters a file that it cannot delete, it does not stop but instead continues deleting those files that it can delete.

## Related methods

Flush, page 213
Flushcache, page 217

## Example

```
status = dmAPIExec("purgelocal,c")
if (status != 1)
print dmAPIGet("getmessage,s0");
```

# Query_cmd

## Purpose

Executes a DQL query.

## Syntax

```
dmAPIGet("query_cmd,session[,read_only][,persistent_cache]
[,consistency_check_value][,unused][,unused][,unused]
,dql_query")
```

## Arguments

**Table 2-110. Query_cmd method arguments**

| Argument | Description |
|---|---|
| *session* | Identifies an open repository session. |
| *read_only* | Whether the results are read-only or may be updated. T(TRUE) means the results are read only. F (FALSE) means that the results may be updated. The default is F. |
| *persistent_cache* | Whether to store the results in the persistent client cache. T(TRUE) means the results are stored in the cache. F (FALSE) means that the results are not stored in the cache. The default is F.<br><br>**Note:** Setting this flag is only effective if the the client_persistent_caching key in the user's dmcl.ini file is set to T (TRUE). |
| *consistency_check_value* | Defines how often the results should be recomputed. This argument is only used if the *persistent_cache* argument is set to T. The argument is ignored if *persistent_cache* is F.<br><br>Valid values are:<br>• Name of a cache config object<br>• A positive integer<br>• One of three keywords or the integer equivalents<br><br>If you identify a cache config object, the server uses the consistency check rules defined in the cache config object to determine whether to recompute the results. Use the cache config's object name to identify a cache config object. |

| Argument | Description |
|---|---|
| | A positive integer value is interpreted in seconds and defines the length of time that the results can be used without being recomputed.<br><br>The keywords are:<br><br>**check_always**, meaning the query is always rerun. check_always is the equivalent of specifying 0 for the argument.<br><br>**check_never**, meaning the results are never checked. If the results are cached, the cached results are used. check_never is the equivalent of specifying -1 for the argument.<br><br>**check_first_access**, meaning the query is run if it is the first request by the client. Subsequent calls that specify check_first_access return the cached results. check_first_access is the equivalent of specifying -2 for the argument.<br><br>The default is check_never. |
| unused | Currently unused and included for future enhancements of the method. |
| *dql_query* | Defines the DQL query to be executed. The query must be less than or equal to 255 characters in length if you are using DDE as the communications protocol between the external application and Content Server. |

## Return value

Query_cmd returns a collection identifier.

## Usage notes

Query_cmd is the recommended API to execute any DQL query through the API.

Setting the *persistent_cache* argument to TRUE instructs Content Server to store the query results in a persistent file. Query cache files are maintained within and across sessions. The collection identifier returned by the method points to that file. Setting the *persistent_cache* argument to T is only effective if query caching is enabled in the user's repository and environment. (For instructions on enabling and managing query caching,

refer to Managing persistent client caches, page 192 in *Content ServerAdministrator's Guide*.)

The *consistency_check* argument is effective only if the results are persistently cached. The argument defines how often the stored query results are checked for consistency against the repository. If you set *persistent_cache* to T, to cache the results, and do not include a consistency check value, the consistency check defaults to check_never, meaning the cached results are always used. (For details about how consistency checking is implemented, refer to Consistency checking, page 49, in *Content Server Fundamentals*.)

## Related methods

Cachequery, page 133

## Example

```
dmAPIGet("query_cmd,S0,T,T,check_first_access,,,,
select owner_name,object_name,title from dm_document
 where subject='SOP'")
```

# Query

## Purpose

Executes DQL statements.

## Syntax

```
dmAPIGet("query,session,dql_query")
```

## Arguments

**Table 2-111. Query method arguments**

| Argument | Description |
|---|---|
| *session* | Identifies an open repository session. |
| *dql_query* | Defines the DQL query that you want to execute. |

## Return value

The Query method returns a collection identifier representing the collection that contains the query results.

## Usage notes

Whenever you execute a DQL statement using the Query method, the results are returned as a collection. This feature lets you write generic code that can process any DQL statement whether it is a SELECT statement or not. Refer to Chapter 4, Using DQL, in *Content Server Fundamentals* for a discussion of query processing and a template for processing code.

## Related methods

Cachequery, page 133
Execquery, page 203
Readquery, page 342

## Example

This example:

- Queries the repository
- Loops through the resulting collection and prints each respondent
- Closes the query collection

```
cmd_str = "query," session ",select r_object_id from dm_document"
query_id = dmAPIGet( cmd_str )
if (query_id == NULL) {
err_msg = dmAPIGet("getmessage," session)
print "Error querying the repository"
print err_msg
exit
}

resp_cntr = 0
while( dmAPIExec("next," session "," query_id) > 0 ) {
obj_id = dmAPIGet("get," session "," query_id ",r_object_id")
if (obj_id == NULL) {
err_msg = dmAPIGet("getmessage," session)
print "Error retrieving object ID from the query collection"
print err_msg
exit
}
print ++resp_cntr " : " obj_id
}

err_flag = dmAPIExec("close," session "," query_id)
if (err_flag == 0) {
err_msg = dmAPIGet("getmessage," session)
print "Error closing the query collection"
print err_msg
exit
}
```

# Queue

## Purpose

Places an object in an inbox or posts an event to a workflow or for a work item.

## Syntax

To queue an object to a user:

```
dmAPIGet("queue,session,object_id,user_name,event,priority
[,send_mail],[due_date],message")
```

To queue an event to a workflow or work item:

```
dmAPIGet("queue,session,workflow_id|workitem_id,[recipient],
event_type,[priority],[send_mail],[due_date],[message]")
```

## Arguments

**Table 2-112. Queue method arguments**

| Argument | Description |
|---|---|
| *session* | Identifies an open repository session. |
| *object_id* | Identifies the object that you want to place on the queue. Use the object's ID.<br><br>You cannot specify an indirect reference (*@object_id*) for this argument. |
| *workflow_id* or *workitem_id* | Identifies the workflow or work item to which to post an event. If you are posting the event to a workflow, use the workflow's object ID. If you are posting the event for a work item, use the work item's object ID. |
| *user_name* | Identifies the user who is to receive the object. Use the user's user name. |

| Argument | Description |
|---|---|
| *recipient* | For events posted to a workflow or work item, identifies the recipient of the event. Use the recipient's user name.<br><br>If the event is posted to a workflow, the default recipient is the workflow supervisor.<br><br>If the event is posted to a work item, the default recipient is the work item performer |
| *event* | Provides information to be interpreted by the application about the specified object. Use a character string value for this argument. |
| *event_type* | The name of the event (a character string value). |
| *priority* | Defines an application- or user-interpreted priority level for the queued item. Specify an integer number for this argument. |
| *send_mail* | Directs the server to send an electronic message to the queue's owner. The message includes the text of the *message* argument.<br><br>This is an optional flag. The default is F (FALSE). |
| *due_date* | Specifies a date for the completion of the work represented by the queued object. |
| *message* | Defines a message to the owner of the queue on which you are placing the task.<br><br>If *send_mail* is set to T, the message is included in the notification. The message may also be retrieved using the Getevents method. |

## Return value

The Queue method returns the object ID of a dmi_queue_item object.

## Usage notes

If you queue an event to a workflow or for a work item, the method automatically sends an email notification to the user specified in the recipient argument. If the recipient argument is not included, the notification is sent to the workflow supervisor for workflows and to the work item's performer for work items.

## Related methods

Dequeue, page 179

## Example

This example places an object in John's queue.

```
stamp = dmAPIGet(sprintf("queue,%s,%s,%s,dm_fetch,
0,T,,%s",session,doc_id, "john","Please review
this document."));
```

# Readquery

## Purpose

Executes DQL queries that do not require database access during query result processing.

## Syntax

```
dmAPIGet("readquery,session,dql_query")
```

## Arguments

**Table 2-113. Readquery method arguments**

| Argument | Description |
|---|---|
| *session* | Identifies an open repository session. |
| *dql_query* | Defines the DQL query that you want to execute. The query must be less than or equal to 255 characters in length if you are using DDE as the communications protocol between the external application and Content Server. |

## Return value

The Readquery method returns a collection identifier.

## Usage notes

Use Readquery when you want to execute a SELECT statement whose results will be processed without any database changes occurring during the processing. For such SELECT statements, Readquery provides better performance than the Query method.

**Note:** Making changes in the repository while processing the results of a Readquery execution automatically closes the collection returned by the SELECT. Consequently, if you want to make changes in the repository while processing query results use the Query method rather than Readquery.

You can execute non-SELECT statements with Readquery also. However, there are no performance benefits to doing so.

If you want to send a query that is greater than 255 characters and you are using DDE, use the Execquery method instead of Readquery.

## Related methods

Cachequery, page 133
Execquery, page 203
Query, page 337

## Example

This example:

- Queries the repository
- Processes the query results
- Closes the collection that holds the query results

```
cmd_str = "readquery," session ",select r_object_id from dm_document"
query_id = dmAPIGet( cmd_str )
if (query_id == NULL) {
err_msg = dmAPIGet("getmessage," session)
print "Error querying the repository"
print err_msg
exit
}

resp_cntr = 0
while( dmAPIExec("next," session "," query_id) > 0 ) {
obj_id = dmAPIGet("get," session "," query_id ",r_object_id")
if (obj_id == NULL) {
err_msg = dmAPIGet("getmessage," sessi)
print "Error retrieving object ID from the query collection"
print err_msg
exit
}
print ++resp_cntr " : " obj_id
}

err_flag = dmAPIExec("close," session "," query_id)
if (err_flag == 0) {
err_msg = dmAPIGet("getmessage," session)
print "Error closing the query collection"
print err_msg
exit
}
```

# Refresh

## Purpose

Updates the attributes values of a mirror object in the repository.

## Syntax

```
dmAPIExec("refresh,session,object_id")
```

## Arguments

**Table 2-114. Refresh method arguments**

| Argument | Description |
|---|---|
| *session* | Identifies an open repository session. |
| *object_id* | Identifies the mirror object or replica you want to refresh. |

## Return value

The Refresh method returns TRUE if successful or FALSE if unsuccessful.

## Usage notes

To refresh a reference link requires at least Browse permission on both the reference link and the remote object to which the mirror object points.

To refresh a replica requires at least Browse permission on the replica.

Refreshing a reference link modifies the r_next_refresh attribute of the dm_reference object.

## Related methods

Dereference, page 181

## Example

```
err = dmAPIExec ("refresh," session "," mirror_obj_id)
```

```
if (err == 0) {
print "Error refreshing the object '" obj_id "'"
print dmAPIGet ("getmessage," session)
exit
}
```

# Register

## Purpose

Registers the current user to receive notification when the specified event occurs.

## Syntax

```
dmAPISet("register,session,object_id,event[,priority]
[,send_mail]","message")
```

## Arguments

**Table 2-115. Register method arguments**

| Argument | Description |
|---|---|
| *session* | Identifies an open repository session. |
| *object_id* | Identifies the object or object type for which you are registering the event.<br><br>To identify a particular object, use the object's object ID or an indirect reference (*@object_id*).<br><br>To identify an object type, use the object ID of the type's dm_type object. The object type must be SysObject or a SysObject subtype. Sysadmin or Superuser privileges are required to specify an object type.<br><br>Appendix B, System Events, lists the valid object types for system-defined events. |
| *event* | Defines the event for which you want to receive notification. Appendix B, System Events, lists the system-defined events. |
| *priority* | Defines a priority level for the event. Use an integer number for this argument. The priority is an optional argument whose value must be interpreted by the user or application. |

| Argument | Description |
|---|---|
| *send_mail* | Directs the server to send electronic mail to the user whenever an event is placed on the user's queue. TRUE means to send mail. FALSE means not to send mail. The default is TRUE. |
| *message* | Defines a message sent to the user receiving the event notification. If send_mail is set, the message is included in the email sent to the owner. Otherwise, the message can be retrieved using the Getevents method. |

## Return value

The Register method returns TRUE if successful or FALSE if unsuccessful.

## Usage notes

Any user can register to receive an event notice.

When you register for an event, the server puts a notification in your inbox when the event occurs. With two exceptions (dm_fetch and dm_acquire), you receive notification each time the event occurs in the same or different repository sessions. For dm_fetch and dm_acquire, you receive only one notification in a repository session even if the event occurs multiple times in the session.

If you register for an event on an object type, you receive a notification every time that event occurs for an object of the type.

Registering for an event doesn't create audit trail entries when the event occurs. To create audit trail entries for an event, you must use an Audit method to initiate auditing. You can both register for an event and audit the event. (Using Audit requires Config_audit privileges.)

With the exclusion of router events, the system-defined events for which you can register and those which you can audit are identical. They are listed in Appendix B, System Events.

You can register (but not audit) the following events with routers as the target of the event:

dm_acquire
dm_reassign
dm_end
dm_resume
dm_force
dm_reverse

dm_forward
dm_start
dm_halt
dm_status
dm_pause

You can register to receive notifications of router events for routers that are currently running. Note, however that router functionality is superseded in Documentum 4.0 and above by the workflow functionality. Documentum clients at version 4.0 and above won't allow you to start new routers.

For information about processing events and more information about how they work, refer to Chapter 11, Tasks, Events, and Inboxes, in *Content Server Fundamentals*.

## Related methods

Audit, page 115
Unaudit, page 466
Unregister, page 484

## Example

The following example registers the current user to receive notification when any type of event occurs to the document specified by *doc_id*:

```
stamp = dmAPISet(sprintf("register,%s,%s,all", session, doc_id),
"An event has occurred on my document")
```

# Reinit

## Purpose

Restarts the root Content Server thread (process).

## Syntax

```
dmAPIExec("reinit,session[,server_config_name]
[,send_to_docbroker]")
```

## Arguments

**Table 2-116. Reinit method arguments**

| Argument | Description |
|---|---|
| *session* | Identifies an open repository session. |
| *server_config _name* | Specifies the server config object that you want the restarted server to use. |
| *send_to_docbroker* | Indicates whether the server should send a checkpoint to the connection broker projection targets on reinitialization. Set the argument to T to direct the server to send the checkpoint. The default is F, which means that a checkpoint is not sent. |

## Return value

The Reinit method returns TRUE if successful or FALSE if unsuccessful.

## Usage notes

You have Sysadmin or Superuser user privileges to use the Reinit method. This method reinitializes the root server thread (process). The ability to reinitialize the server lets you change the server's configuration by allowing you to change the server's server config object and then restart the server. If you do not specify a server config object, the method assumes the server's current server config object.

Issuing a Reinit also affects the behavior of the repository session that issues the Reinit call.

## Related methods

Restart, page 386

## Example

```
status = dmAPIExec("reinit,c")
if (status != 1)
print dmAPIGet("getmessage,s0");
```

# Remove

## Purpose

Removes a value in a repeating attribute.

## Syntax

```
dmAPIExec("remove,session,object_id,attribute[[index]]")
```

## Arguments

**Table 2-117. Remove method arguments**

| Argument | Description |
|---|---|
| *session* | Identifies an open repository session. |
| *object_id* | Identifies the object that contains the specified attribute.<br><br>Use the object's ID or an indirect reference (*@object_id*) that points to the object.<br><br>If the object is the api config object, specify the keyword apiconfig instead of the object ID. |
| *attribute* | Specifies the repeating attribute that contains the value you want to remove. |
| [*index*] | Specifies the position (in the repeating attribute) that holds the value you want to remove. Note that you must enclose the index value in square brackets [ ]. If unspecified, the default is zero. |

## Return value

The Remove method returns TRUE if successful or FALSE if unsuccessful.

## Usage notes

Executing the Remove method removes the value at the specified position by removing the row at that position. The remaining rows in the attribute are renumbered.

## Related methods

Append, page 77
Insert, page 271

## Example

This example:

- Creates a document object
- Appends the names of the authors
- Removes an author's name

```
doc_id = dmAPIGet("create," session ",dm_document")
if (doc_id == NULL) {
err_msg = dmAPIGet("getmessage," session)
print "Error creating a document object"
print err_msg
exit
}

author_list = "W. Smith,J. Jones,R. Abnous,S. Ruppenthal"
count = split(author_list, author, ",")
cmd_str = "append," session "," doc_id ",authors"
for (i=1; i<=count; i++) {
err_flag = dmAPIExec( cmd_str, author[i] )
if (err_flag == 0) {
err_msg = dmAPIGet("getmessage," session)
print "Error appending authors' name to the 'authors' attribute"
print err_msg
exit
}
}

err_flag = dmAPIExec("remove," session "," doc_id ",authors[2]")
if (err_flag == 0) {
err_msg = dmAPIGet("getmessage," session)
print "Error saving the document object"
print err_msg
exit
}
```

# Removeactivity

## Purpose

Removes an activity from a workflow definition.

## Syntax

```
dmAPIExec("removeactivity,session,process_id
,activity_identifier")
```

## Arguments

**Table 2-118. Removeactivity method arguments**

| Argument | Description |
|---|---|
| *session* | Identifies an open repository session. |
| *process_id* | Identifies the process definition to remove the activity from. Use the object ID of the process. |
| *activity_identifier* | Uniquely identifies the activity to remove. |

## Return value

The Removeactivity method returns TRUE if successful or FALSE if unsuccessful.

## Usage notes

The Removeactivity method removes an activity, added with the Addactivity method, from an uninstalled workflow process definition.

You cannot remove an activity which is in use by any links. Use Removelink to remove the links before using Removeactivity.

## Related methods

Addactivity, page 44
Addlink, page 52
Removelink, page 357

## Example

```
status = dmexec("removeactivity," session ",4b00007b80000100,
original_request")
```

# Removecontent

## Purpose

Deletes a content file from an object.

## Syntax

```
dmAPIExec("removecontent,session,object_id[,page_number]")
```

## Arguments

**Table 2-119. Removecontent method arguments**

| Argument | Description |
|---|---|
| *session* | Identifies an open repository session. |
| *object_id* | Specifies the object from which you are removing the content file. Use the object's ID or an indirect reference (*@object_id*) that points to the object. |
| *page_number* | Identifies the content that you are removing. Use the content's page number. If unspecified, the default is zero (the first content). |

## Return value

The Removecontent method returns TRUE if successful or FALSE if unsuccessful.

## Usage notes

You must have at least Version permission on a document to remove its content.

A page number is a number assigned to each content in an object that indicates the content's position within the object's ordered contents. Page numbers begin with zero and increment by one for each additional content.

Removing a content renumbers any remaining content.

## Example

This example:

- Fetches a specified document
- Removes the first content (page number zero)
- Saves the document

```
err_flag = dmAPIExec("fetch," session "," doc_id)
if (err_flag == 0) {
err_msg = dmAPIGet("getmessage," session)
print "Error fetching the document"
print err_msg
exit
}

err_flag = dmAPIExec("removecontent," session "," doc_id ",0")
if (err_flag == 0) {
err_msg = dmAPIGet("getmessage," session)
print "Error removing the first content page"
print err_msg
exit
}

err_flag = dmAPIExec("save," session "," doc_id)
if (err_flag == 0) {
err_msg = dmAPIGet("getmessage," session)
print "Error saving the document"
print err_msg
exit
}
```

# Removelink

## Purpose

Removes a link from a workflow definition.

## Syntax

```
dmAPIExec("removelink,session,process_id,link_identifier")
```

## Arguments

**Table 2-120. Removelink method arguments**

| Argument | Description |
|---|---|
| *session* | Identifies an open repository session. |
| *object_id* | Identifies the process. |
| *link_identifier* | Uniquely identifies the link to remove. |

## Return value

The Removelink method returns TRUE if successful or FALSE if unsuccessful.

## Usage notes

The Removelink method removes a link, previously added with Addlink, from a process definition.

## Related methods

Addactivity, page 44
Addlink, page 52
Removeactivity, page 353

## Example

```
status = dmAPIExec("removelink,s0," proc_id ",link_0")
```

# Removenote

## Purpose

Detaches an annotation from a document or other SysObject or removes a note from a package.

## Syntax

```
dmAPIExec("removenote,session,annotation_id,object_id")
```

## Arguments

**Table 2-121. Removenote method arguments**

| Argument | Description |
|---|---|
| *session* | Identifies an open repository session. |
| *annotation_id* | Specifies which annotation you want to remove from the document. Use the annotation's object ID or an indirect reference (*@object_id*) that point to the annotation |
| *object_id* | Identifies the document or package from which you want to remove the annotation. Use the document or package's object ID. |

## Return value

The Removenote method returns TRUE if successful or FALSE if unsuccessful.

## Usage notes

The Removenote method destroys the relation object that links an annotation to a document. It does not destroy the annotation itself.

The Removenote method detaches a note from a package, or destroys the relation object that links an annotation to a document. It does not destroy the annotation itself.

To use Removenote to detach an annotation, you must have:

- At least RELATE permission on the object from which you are detaching the annotation, and

- At least Write permission on the note object.

You can only use Removenote to detach a note from a package if you added the note yourself.

## Related methods

Addnote, page 54

## Example

```
status = dmAPIExec("removenote,c,41000bb2800014a2,09000bb2800012a5")
if (status != 1)
print dmAPIGet("getmessage,s0");

status = dmAPIExec("save,c,41000bb2800014a2")
if (status != 1)
print dmAPIGet("getmessage,s0");
```

# Removepackage

## Purpose

Detaches a package from a specific port of a start activity or from a work item.

## Syntax

```
dmAPIExec("removepackage,session,object_id,
start_activity_name,port_name,package_name")

dmAPIExec("removepackage,session,object_id,package_name")
```

## Arguments

**Table 2-122. Removepackage method arguments**

| Argument | Description |
|---|---|
| *session* | Identifies an open repository session. |
| *object_id* | Identifies the workflow or work item from which to remove the package. |
| *start_activity_name* | Specifies the start activity from which to remove the package, if the workflow has more than one start activity. |
| *port_name* | Identifies the port from which to remove the package. |
| *package_name* | Identifies the package to remove |

## Return value

The Removepackage method returns TRUE if successful or FALSE if unsuccessful.

## Usage notes

The Removepackage method allows a workflow supervisor to detach a package from a specific port of a start activity, and enables work item performers to detach a package from a work item.

## Related methods

Addpackage, page 56

## Example

```
buff = "removepackageinfo," sess "," wfid "," actname ","
portname ",pack1"
    istatus = dmAPIExec(buff)
```

# Removepackageinfo

## Purpose

Removes a package definition from a port.

## Syntax

```
dmAPIExec("removepackageinfo,session,activity_id,port_name,
package_name")
```

## Arguments

**Table 2-123. Removepackageinfo method arguments**

| Argument | Description |
|---|---|
| *session* | Identifies an open repository session. |
| *activity_id* | Identifies the activity from which to remove the package information. |
| *port_name* | Identifies the port from which to remove the package information. |
| *package_name* | Identifies the package to remove |

## Return value

The Removepackageinfo method returns TRUE if successful or FALSE if unsuccessful.

## Usage notes

The Removepackageinfo method removes the specified package definition from the specified port.

## Related methods

Addpackageinfo, page 60

## Example

```
buff = "removepackageinfo,s0," act_id "," portname ",pack1"
istatus = dmAPIExec(buff)
```

# Removepart

## Purpose

Removes a component of a virtual document.

## Syntax

```
dmAPIExec("removepart,session,document_id,
containment_id|order_no[,orderno_flag]")
```

## Arguments

**Table 2-124. Removepart method arguments**

| Argument | Description |
|---|---|
| *session* | Identifies an open repository session. |
| *document_id* | Identifies the virtual document that contains the component you want to remove. Use the document's object ID or an indirect reference (*@object_id*) that points to the document. |
| *containment_id* | Identifies the component you want to remove. Use the component's containment ID. This is the object ID of the containment object that represents the link between the document and the component. If you use this, do not set orderno_flag. |
| *order_no* | Identifies the component you want to remove. Use this option only if your application defines and manages the order numbers for virtual document components itself. Specify an integer number for order_no. If you specify order_no, you must set the orderno_flag to T and you cannot provide a containment_id value. |
| *orderno_flag* | Indicates whether the preceding argument's value is a user-defined order number for the inserted component. You must set this to T (TRUE) if the preceding argument is the order_no option. The default value for this flag is F (FALSE), indicating that the preceding argument provides a containment ID value. |

## Return value

The Removepart method returns TRUE if successful or FALSE if unsuccessful.

## Usage notes

To specify which component you want to remove, use either the *containment_id* or the *order_no* argument. These arguments are mutually exclusive. Do not use both.

If you use the server's default numbering to order virtual document components, then use the *containment_id* argument to indicate which component you want to remove. The *containment_id* is the object ID of the containment object that links the component to the virtual document. You can obtain this object ID with the following query:

```
SELECT r_object_id, CONTAIN_ID FROM dm_sysobject
IN DOCUMENT ID('virtual_doc_id')
```

This returns a list of the object IDs and containment IDs for all the direct components of the virtual document.

If you are providing and managing your own order numbers for virtual document components, then use the *order_no* argument to indicate which component you want to remove. The *order_no* argument takes an integer value. When you use the *order_no* argument, you must also set the *orderno_flag* to T. This flag indicates to the server that you have given it a user-defined order number rather than a containment ID.

It is not necessary to set the *orderno_flag* when you are specifying a containment ID because the default value for the flag is F, which indicates that you have specified a containment ID.

## Related methods

Appendpart, page 84
Insertpart, page 280
Updatepart, page 486

## Example

```
component_id = dmget("create,s0,dm_document");
 if (component_id == NULL)
 {  err_mess = dmget("getmessage,s0")
    print err_mess
    exit
 }

status = dmset("set,s0,last,object_name", "Chapter1")

status = dmexec("save,s0,last")
 if (status != 1)
```

```
 {  err_mess = dmget("getmessage,s0")
    print err_mess
    exit
 }

parent_id = dmget("create,s0,dm_document");
 if (parent_id == NULL)
 {  err_mess = dmget("getmessage,s0")
    print err_mess
    exit
 }

status = dmset("set,s0,last,object_name", "Book")

containment_id = dmget("appendpart,s0,last," component_id)

status = dmexec("save,s0,last")
 if (status != 1)
 {  err_mess = dmget("getmessage,s0")
    print err_mess
    exit
 }

status = dmexec("removepart,s0,last," containment_id)
 if (status != 1)
 {  err_mess = dmget("getmessage,s0")
    print err_mess
    exit
 }

 status = dmexec("save,s0,last")
 if (status != 1)
 {  err_mess = dmget("getmessage,s0")
    print err_mess
    exit
 }
```

# Removeport

## Purpose

Removes a port added with the Addport method.

## Syntax

```
dmAPIExec("removeport,session,activity_id,port_name")
```

## Arguments

**Table 2-125. Removeport method arguments**

| Argument | Description |
|---|---|
| *session* | Identifies an open repository session. |
| *activity_id* | Identifies the activity definition that you want to remove the port from. Use the object ID of the activity definition. |
| *port_name* | Identifies the port to be removed |

## Return value

The Removeport method returns TRUE if successful or FALSE if unsuccessful.

## Usage notes

The Removeport method removes a port added with the Addport method.

## Related methods

Addport, page 64

## Example

```
status = dmexec("removeport," session ",
4c00007b80000100,inport1")
```

# Removerendition

## Purpose

Deletes a rendition from an object.

## Syntax

```
dmAPIExec("removerendition,session,object_id,format
[,page_number][,atomic][,page_modifier]")
```

## Arguments

**Table 2-126. Removerendition method arguments**

| Argument | Description |
|---|---|
| *session* | Identifies an open repository session. |
| *object_id* | Specifies the object whose rendition you are removing. Use the object's ID or an indirect reference (*@object_id*) that points to the object. |
| *format* | Defines the format of the file you are removing. You must specify a valid file format. |
| *page_number* | Identifies the content page with which the rendition is associated. The default is zero (the first content page). |
| *atomic* | Indicates whether you want the changes to be saved to the repository immediately. TRUE save the changes immediately. FALSE saves the changes when the object is saved. The default is FALSE. |
| *page_modifier* | Identifies a rendition. The page modifier is a character string defined when the rendition is created.<br><br>If not included, the method removes all renditions of the specified format for the specified content page. |

## Return value

The Removerendition method returns TRUE if successful or FALSE if unsuccessful.

## Usage notes

You must have at least Write permission on a document to remove one of its renditions.

The combination of the format, page_number, and page_modifier arguments identify which rendition ro remove. For example, suppose you have a document, 090000022364ec1, that has three content pages and that page three has a PDF rendition and two GIF renditions. To remove the PDF rendition, use the following statement:

```
dmAPIExec("removerendition,c,0900000223643ec1,pdf,3")
```

It isn't necessary to include a page_modifier because the PDF rendition has no page modifier.

The GIF renditions associated with the third page have page modifiers, to disambiguate them. The page modfier for the first GIF rendition is rend_1 and the modifier for the second is rend_2. To remove the first GIF rendition, use

```
dmAPIExec("removerendition,c,0900000223643ec1,pdf,2,,rend_1")
```

In this case, you must include the page modifier to identify which GIF rendition you want to remove.

If you wanted to remove both GIF renditions, you could leave out the page modifier:

```
dmAPIExec("removerendition,c,0900000223643ec1,pdf,2")
```

When a page modifier isn't included, the method removes all renditions in the specified format for the specified content page.

## Example

This example:

- Fetches a specified document
- Removes the first, perhaps only, page of content of the text format
- Saves the document

```
err_flag = dmAPIExec("fetch," session "," doc_id)
if (err_flag == 0) {
err_msg = dmAPIGet("getmessage," session
print "Error fetching the document"
print err_msg
exit
}

err_flag = dmAPIExec("removerendition," session "," doc_id ",text,0)
if (err_flag == 0) {
err_msg = dmAPIGet("getmessage," session)
print "Error removing the first text format content page"
print err_msg
exit
}

err_flag = dmAPIExec("save," session "," doc_id)
```

```
if (err_flag == 0) {
err_msg = dmAPIGet("getmessage," session
print "Error saving the document"
print err_msg
exit
}
```

# Removeroutecase

## Purpose

Removes a route case from the condition list.

## Syntax

```
dmAPIExec("removeroutecase,session,activity_id
,route_case_identifier")
```

## Arguments

**Table 2-127. Removeroutecase method arguments**

| Argument | Description |
|---|---|
| *session* | Identifies an open repository session. |
| *activity_id* | Identifies the activity definition from which to remove a route case. Use the object ID of the activity. |
| *route_case_identifier* | Identifies the route case. Use EXCEPTIONAL to remove the exceptional route case. |

## Return value

The Removeroutecase method returns TRUE if successful or FALSE if unsuccessful.

## Usage notes

The method removes the specified route case from the condition list. Use the keyword EXCEPTIONAL to remove the exception route case.

## Related methods

Addroutecase, page 72

## Examples

```
status = dmAPIExec("removeroutecase,s0," act_id1 ",route1")
```

The following example removes the exceptional route case:

```
status = dmAPIExec("removeroutecase,s0," act_id1
",EXCEPTIONAL")
```

# Removestate

## Purpose

Removes a state from a lifecycle.

## Syntax

```
dmAPIexec("removestate,session,object_id
[,position|state_name]")
```

## Arguments

**Table 2-128. Removestate method arguments**

| Argument | Description |
|---|---|
| *session* | Identifies an open repository session. |
| *object_id* | The object ID of the dm_policy object. |
| *position* | The position of the state to be removed. |
| *state_name* | The name of the state to be removed. |

## Return value

The Removestate method returns TRUE if successful or FALSE if unsuccessful.

## Usage notes

The Removestate method removes a state from a validated or draft lifecycle by position or by name. You cannot remove a state from an installed policy.

You must have Write permission to remove a state.

You can remove state by position or by state name. If you do not specify a state position or name, the base state is removed.

If you remove states from a policy that is attached to SysObjects, the server returns a warning message when you save the changed policy.

The Removestate method destroys the objects recorded in the state's entry_criteria_id, action_object_id, and type_override_id attributes.

## Related methods

Insertstate, page 287

## Example

```
status = dmAPIExec("removestate," session ",
   4600007b8000567f,0")
```

# Repeat

## Purpose

Allows a user to repeat a work item.

## Syntax

```
dmAPIGet("repeat,session,workitem_id{,user_name|group_name}")
```

## Arguments

**Table 2-129. Repeat method arguments**

| Argument | Description |
|---|---|
| *session* | Identifies an open repository session. |
| *workitem_id* | Identifies the work item to repeat. Use the object's ID or an indirect reference (*@object_id*) that points to the object. |
| *user_name* | Identifies the next user(s) for the work item. Use the user's name. |
| *group_name* | Identifies the next group(s) for the work item. Use the group's name. Only one member of a group can acquire the repeated work item. |

## Return value

The Repeat method returns TRUE if successful or FALSE if unsuccessful.

## Usage notes

The Repeat method extends a work item. The original performer, the workflow supervisor, or a user with Sysadmin or Superuser privileges can use Repeat to assign the work item to a second set of users when the original performer completes work item.

The work item must be in the aquired state. You cannot repeat a work item generated from an activity whose performer category is 9 (Some users in a group or some users in the repository, sequentially).

You can repeat a work item one time.

## Related methods

Acquire, page 42
Addpackage, page 56
Complete, page 153
Delegate, page 174

## Example

```
status = dmAPIExec("repeat," sess "," r_id
",john,mary,review_group")
```

# Repeating

## Purpose

Determines if an attribute is a repeating attribute or a single-valued attribute.

## Syntax

```
dmAPIGet("repeating,session,object_id,attribute")
```

## Arguments

**Table 2-130. Repeating method arguments**

| Argument | Description |
|---|---|
| *session* | Identifies an open repository session. |
| *object_id* | Identifies the object that contains the specified attribute. Use the object's ID or an indirect reference (*@object_id*) that points to the object. |
| *attribute* | Identifies the attribute. Use the attribute's name. |

## Return value

The Repeating method returns 1 for a repeating attribute or 0 for a single-valued attribute.

## Usage notes

The Repeating method determines if an attribute is a repeating attribute or a single-valued attribute.

## Example

This example:

- Fetches the specified object
- Determines the number of attributes
- Uses the Repeating method to determine if the attribute is single-valued or repeating and then gets the name, datatype, and value of each of the object's attributes

- Prints the name and value of each attribute (If the attribute is a repeating attribute, all values for the attribute are printed.)

```
err_flag = dmAPIExec("fetch," session "," obj_id)
if (err_flag == 0) {
err_msg = dmAPIGet("getmessage," session)
print "Error fetching the object"
print err_msg
exit
}

attr_cnt = dmAPIGet("count," session "," doc_id)
if (attr_cnt == NULL) {
err_msg = dmAPIGet("getmessage," session)
print "Error retrieving the number of attributes"
print err_msg
exit
}

for (i=0; i<attr_cnt; i++) {
cmd_str = "get," session "," obj_id ",_names[" i "]"
attr_name = dmAPIGet(cmd_str)
if (attr_name == NULL) {
err_msg = dmAPIGet("getmessage," session)
print "Error retrieving the attribute name : " i
print err_msg
exit
}

cmd_str = "datatype," session "," obj_id "," attr_name
d_type = dmAPIGet(cmd_str)
if (d_type == NULL) {
err_msg = dmAPIGet("getmessage," session)
print "Error retrieving the datatype for '" attr_name "'"
print err_msg
exit
}

if (d_type == 0) d_type_str = "boolean"
if (d_type == 1) d_type_str = "integer"
if (d_type == 2) d_type_str = "string"
if (d_type == 3) d_type_str = "ID"
if (d_type == 4) d_type_str = "time"
if (d_type == 5) d_type_str = "float"
#
cmd_str = "repeating," session "," obj_id "," attr_name
r_flag = dmAPIGet(cmd_str)
if (r_flag == NULL) {
err_msg = dmAPIGet("getmessage," session)
if (err_msg != NULL) {
print
"Error retrieving the repeating flag for '" attr_name "'"
print err_msg
exit
}
}

if (r_flag == 0) {
```

```
cmd_str = "get," session "," obj_id "," attr_name
value = dmAPIGet(cmd_str)
if (value == NULL) value = "NULL"
 out_str=sprintf("%2s)%19s:%s",i,attr_name, d_type_str, value)
}
else {
cnt = dmAPIGet("values," session "," obj_id)
if (cnt == NULL) {
err_msg = dmAPIGet("getmessage," sessio)
print
"Error retrieving number of values for'" attr_name "'"
print err_msg
exit
}
out_str =\
sprintf("%2s)%19s : ( %s value(s))", i, attr_name, d_type_str, cnt)
}
print out_str
if (r_flag != 0 && cnt > 0) {
for (j=0; jcnt; j++) {
cmd_str = "get," session "," obj_id "," attr_name "[" j "]"
value = dmAPIGet(cmd_str)
if (value == NULL) value = "NULL"
out_str = sprintf("%29s : %s", j, value)
print out_str
}
}
}
```

# Reset

## Purpose

Resets the _status attribute of an object.

## Syntax

```
dmAPIExec("reset,session,object_id")
```

## Arguments

**Table 2-131. Reset method arguments**

| Argument | Description |
|---|---|
| *session* | Identifies an open repository session. |
| *object_id* | Identifies the object whose status you wish to reset. Use the object's ID or an indirect reference (*@object_id*) that points to the object. |

## Return value

The Reset method returns TRUE if successful or FALSE if unsuccessful.

## Usage notes

When an error condition occurs during an operation on a document or its content, the document's special attribute _status is set to indicate the level of error. The server does not allow you to check in or save the object until the _status attribute is reset. The Reset method sets the _status attribute to 0.

## Example

The following example checks the _status attribute, retrieves any status messages, and resets _status if necessary.

```
status = dmAPIGet(sprintf("get,%s,%s,_status",session,id));
if (status != 0) {
msgs = dmAPIGet(sprintf("getmessage,%s",session));
if(msgs != NULL){
```

```
print msgs;
cmd = sprintf("reset,%s,%s",session,id);
dmAPIExec(cmd);
}
}
```

# Resolvealias

## Purpose

Resolves an alias.

## Syntax

```
dmAPIGet("resolvealias,session,[object_id],
%[alias_set_name.]alias_name")
```

## Arguments

**Table 2-132. Resolvealias method arguments**

| Argument | Description |
|---|---|
| *session* | Identifies an open repository session. |
| *object_id* | Identifies an object of type SysObject, a subtype of SysObject, a workflow object, or a work item object.<br><br>If this is supplied, the server attempts to resolve the alias using the scopes appropriate for the object. Refer to the Usage notes for a detailed explanation of the resolution algorithm. |
| *alias_set_name* | The name of the alias set object that contains the alias name you want to resolve. This is optional. |
| *alias_name* | The alias name that you want to resolve to a real user or group name or folder path. This is a value found in the alias_name attribute of the alias set object identified in *alias_set_name*. |

## Return value

The Resolvealias method returns a user or group name or a folder path.

## Usage notes

If you include the alias set name as a method argument, the server searches the alias set object identified by that argument to resolve the alias.

If you don't include the alias set name as a method argument, the server searches a set of scopes in a prescribed order to resolve the alias name.

### Resolution algorithm if the object is a SysObject

If you identify a SysObject or SysObject subtype in the method call:

- The server first searches the alias set defined in the object's lifecycle scope. This is the alias set defined in the object's r_alias_set_id attribute.
- If the alias is not found in the lifecycle scope, the server looks next at the alias set object defined for the session.
- If the session's default alias set has no match, the server looks at the alias set defined for the user issuing the method.
- Finally, if the server hasn't found a match in the alias sets of the lifecycle, session, user, or user's default group, it examines the alias set defined in the server config object.
- If the user's alias set has no match, the server looks at the alias set defined for the user's default group.

### Resolution algorithm if the object is a workflow or work item

If you identify a workflow or work item object in the method call:

- The server first searches the alias set defined in the workflow's r_alias_set_id attribute for a match.
- If the alias is not found in the workflow scope, the server looks next at the alias set object defined for the session.
- If the session's default alias set doesn't have a match, the server looks at the alias set defined in the user object for the user who completed the previous work item.
- If there is no match in the alias set defined for the user, the server next examines the alias set defined for the user's default group.
- Finally, if the server hasn't found a match in the workflow, session, user, or group alias sets, it examines the alias set defined in the server config object.

For details about these alias scopes, refer to Resolving aliases in workflows, page 315 in *Content Server Fundamentals*.

### If you do not identify an object or an alias set name

If you don't identify an object in the method call or an alias set name in the alias specification, the server begins the search with the alias set defined for the session, and continues with the user's alias set and then the alias set defined in the server config object.

## Example

```
BEGIN {
  # This script tests the resolvealias API as follows:
  # (1) without sysobject id, i.e., resolvealias,c,,<alias>
  #    The alias will be resolved based on the first match
  #     in the following sequence where it is defined:
```

```
  #       . Session Scope
  #       . User Scope
  #       . System Scope
  # (2) with sysobject id, i.e., #resolvealias,c,<sys_id>,<alias>
  #     The alias will be resolved based on the first match
  #     in the following sequence where it is defined:
  #       . Policy Scope
  #       . Session Scope
  #       . User Scope
  #       . System Scope
  #
  #
 session = dmget("connect,test_nt_1,tuser1,password1")
  if (session == NULL)
 {
    printf "Cannot connect\n"
    exit
   }
# Create first alias set, Project 1.
  alias_id_1 = dmget("create,c,dm_alias_set");
  dmset("set,c,l,object_name", "Project 1");
  dmset("set,c,l,alias_name", "Project Lead");
  dmset("set,c,l,alias_value", "test_usr_1");
  dmexec("save,c,l");
# Create second alias set, Project 2.
  alias_id_2 = dmget("create,c,dm_alias_set");
  dmset("set,c,l,object_name", "Project 2");
  dmset("set,c,l,alias_name", "Project Lead");
  dmset("set,c,l,alias_value", "test_usr_2");
  dmexec("save,c,l");
# Set the session scope for aliases to Project 1
  dmset("set,c,sessionconfig,alias_set", "Project 1");
# Resolve the alias %Project Lead based on session scope.
# The expected value is test_usr_1.
  real_value_1 = dmget("resolvealias,c,,%Project Lead");
  printf "The real value for alias %Project Lead under scope Project 1 is
  %s.\n", real_value_1;
# Create a policy object with alias_set_id set to Project 2.
  policy_id = dmget("create,c,dm_policy");
  dmset("set,c,l,object_name", "test_alias_set");
  dmset("set,c,l,included_type", "dm_document");
  dmset("set,c,l,alias_set_ids", alias_id_2);
  dmget("appendstate,c,l");
  dmset("set,c,l,state_name", "S0");
  dmset("set,c,l,allow_attach", "T");
  dmexec("save,c,l");
  dmexec("validate,c,l");
  dmexec("install,c,l");

# Create a document attached to the previous policy object
  doc_id = dmget("create,c,dm_document");
  dmset("set,c,l,object_name", "test_alias_set");
  dmexec("save,c,l");
  dmexec("attach,c,l," policy_id);

  # Resolve the alias %Project Lead against the document
  real_value_2 = dmget("resolvealias,c," doc_id ",%Project Lead");
  printf "The real value for alias %Project Lead under scope Project 2 is %s.\n",
```

```
real_value_2;
dmexec("disconnect," session);
exit;
}
```

# Restart

## Purpose

Reinitializes a session's server thread (process) or restarts a halted workflow.

## Syntax

```
dmAPIExec("restart,session[,server_config_name]
[,reset_client]")

dmAPIExec("restart,session,workflow_id[,activity_seq_no]")
```

## Arguments

**Table 2-133. Restart method arguments**

| Argument | Description |
|---|---|
| *session* | Identifies an open repository session. |
| *server_config _name* | Specifies the server that you want to restart. Use the object name of the server's associated server config object.<br><br>This argument is ignored if *reset_client_flag* is set to T (TRUE). |
| *reset_client* | Indicates you are only changing client-side parameters. (Refer to the Usage notes for details.) This is FALSE by default. |
| *activity_seq_no* | Identifies the activity instance to restart. If no activity instance is specified, restarts the workflow. |

## Return value

The Restart method returns TRUE if successful or FALSE if unsuccessful.

## Usage notes

Use the Restart method to change the server's configuration or to restart a halted workflow or a specific halted activity.

### Using restart to change the server's configuration

Use the Restart method when you want to change the server's configuration only for the current session or when you want to change the client's cache size or the location of the client's local area. The Restart method reinitializes the server thread (process) that is servicing your current session. It does not reinitialize the root server thread (process). You must have Sysadmin or Superuser user privileges to use Restart to reinitialize the server.

Any user can use Restart to change the size of the client cache or change the location of the client's local area. To do this, set the appropriate attributes in the session config object and then issue the Restart method with the *reset_client* flag set to T (TRUE). (This flag is FALSE by default.)

When the *reset_client* flag is TRUE, the *server_config_name* argument is ignored if it is set. Consequently, it is not useful or necessary to include *server_config_name* when you use the *reset_client* flag. However, because arguments to methods are positional, you must include the *server_config_name*'s comma as a place holder even when you do not include the argument itself.

### Using restart to restart a halted workflow

The Restart method can also restart a halted workflow or a specific halted activity instance. Only a workflow supervisor or a user with Sysadmin or Superuser privileges can use the Restart method to restart a workflow or activity.

When restarting a halted workflow, the Restart method removes all run-time activities, generated work items and involved packages, and then generates a run-time instance for each start activity with the r_act_state attribute set to dormant. For each activity that requires a pre-timer, Restart generates a run-time instance and sets the r_pre_timer value.

You cannot restart a workflow that is not halted. You cannot restart a workflow with any active automatic work items exist.

If you specify an activity_sequence_no, the Restart method restarts a specific activity instance whose r_act_state attribute value is failed.

## Related methods

Reinit, page 349

## Example

```
status = dmAPISet("set,c,sessionconfig,local_path", "/home/user")
if (status != 1)
print dmAPIGet("getmessage,s0");

status = dmAPIExec("restart,c,,T")
```

```
if (status != 1)
print dmAPIGet("getmessage,s0");
```

# Restore

## Purpose

Queues a request to restore a content file from the archives.

## Syntax

```
dmAPIGet("restore,session[,predicate][,dump_file][,operator]
[,priority][,send_mail][,due_date]")
```

## Arguments

**Table 2-134. Restore method arguments**

| Argument | Description |
|---|---|
| *session* | Identifies an open repository session. |
| *predicate* | Identifies the object or objects you want to restore. A valid predicate has the format<br>`type_name WHERE qualification`<br>The *type_name* must be a SysObject subtype, and the *qualification* must be a valid WHERE clause qualification.<br>Do not specify the (ALL) keyword with *type_name*. The dmarchive utility called to execute the restore request automatically adds (ALL) to the type name in the predicate.<br>If a predicate is not included, all objects in the target dump file are restored. |
| *dump_file* | Names a specific dump file as the archive to search. This defaults to the value in the set_file attribute of the content object representing the content you are restoring. Refer to the Usage notes for more information. |
| *operator* | Specifies who receives the restore request. Specify a valid user name. If you do not include this, request is queued to the user specified in the operator_name attribute of the server config object. (By default, this attribute contains the name of the repository owner.) |

| Argument | Description |
|---|---|
| *priority* | Specifies a priority level for the request. The server operator does not use this value. It is solely for the convenience of end users or user applications. The default value is zero. |
| *send_mail* | Directs the server to send an electronic mail notification about the restore request to the specified operator. The default is FALSE—mail is not sent. |
| *due_date* | Informs the operator when the restore request should be completed. The default for this is NULLDATE. |

## Return value

The Restore method returns the object ID of the dmi_queue_item for queued restore requests.

Because the query defined by the predicate can return more than one object, the ID returned by the Restore method is the ID of the last object queued for restoration. To find all the objects queued for restoration, run a query against the dm_queue view.

## Usage notes

Executing the Restore method runs the query specified in the predicate to ensure that the specified objects exist, are currently off-line, and are SysObject subtypes. If these conditions are met, the method queues a DM_RESTORE event to the inbox of the repository operator, sending the predicate as the event message.

**Note:** Do not include the keyword (ALL) in the type name in the predicate. The dmarchive utility, called to process the restore requests, adds (ALL) to the type name automatically. Consequently, if you include the keyword in the predicate, the utility fails when it attempts to execute the query generated from the predicate because (ALL) appears twice in the syntax.

The operator is responsible for running the dm_restore utility to actually restore the requested file or files. The utility will read the operator's inbox, extract the restore events, and perform that actual operations to restore the file or files. If you do not specify an operator, the event is queued to the default operator, who is the user specified in the operator_name attribute of the server's server config object.

Generally, it is not necessary to specify a dump file. When an object is archived, the system sets its associated content object's set_file attribute to the full path of the dump file that contains the archived object. When a restore is done, dmarchive reads that attribute to determine which dump file it should scan for the object.

If you do specify a dump file, the file you identify is used instead of the file identified in the set_file attribute.

Refer to Archiving and restoring documents, page 280 in the *Content Server Administrator's Guide* for more information about archiving and restoring content files.

## Related methods

Archive, page 98
Dequeue, page 179

## Example

```
event_id = dmAPIGet("restore,c,dm_document
where r_object_id = '09000bb2800012a8',,sysadmin,
1,T,12/12/94")

if (event_id == NULL)
print dmAPIGet("getmessage,s0");
```

# Resume

## Purpose

Restarts a halted workflow, halted activity, or paused work item, or moves a SysObject from a lifecycle exception state back to a normal state.

## Syntax

To resume halted workflow, halted activity, or paused work item:

```
dmAPIExec("resume,session,object_id[,activity_seq_no]")
```

To resume a SysObject from an exception state:

```
dmAPIExec("resume,session,sysobject_id
[,from_state |from_position,return_to_base_flag
,test_only_flag,override_flag]")
```

To resume a SysObject from an exception state at a scheduled time:

```
dmAPIExec("resume,session,sysobject_id,scheduled_date
,[pattern],[from_state |from_position |cancel
,return_to_base_flag,override_flag]")
```

## Arguments

**Table 2-135. Resume method arguments**

| Argument | Description |
|---|---|
| *session* | Identifies an open repository session. |
| *object_id* | Identifies the paused workflow, activity, work item, or SysObject to resume. Use the object's object ID.<br><br>If you are resuming an object from a lifecycle exception state, you can use an indirect reference (*@objectid*) or identify a replica in this argument. The operation affects the source object, not the local copy. |
| *activity_seq_no* | Identifies the unique sequence number of an activity run-time instance. |
| *from_state* | The state from which to resume the SysObject. You must specify *from_state* or *from_position* if you specify a *scheduled_date*. |

| Argument | Description |
|---|---|
| *from_position* | The position of the state from which to resume the SysObject. You must specify *from_state* or *from_position* if you specify a *scheduled_date* |
| *return_to_base_flag* | TRUE or FALSE. When set to TRUE, resumes the object to the base state instead of the normal state. Otherwise, resumes the object to the normal state. |
| *test_only_flag* | TRUE or FALSE. When set to TRUE, Resume evaluates the entry criteria to determine if the document can be promoted. The default value is FALSE. |
| *override_flag* | TRUE or FALSE. When set to TRUE, the transition proceeds even if the entry criteria of the destination state is not satisfied. Only users with Superuser user privileges or the lifecycle owner can specify this argument. The default is FALSE. |
| *scheduled_date* | The date and time for which this Resume is scheduled. |
| *pattern* | The date-time format for *scheduled_date.* The pattern defaults to client-side localized date-time format if not specified. |
| cancel | Cancels the scheduled Resume. To cancel a scheduled Resume, provide the original scheduled date. |

## Return value

The Resume method returns TRUE if successful or FALSE if unsuccessful.

## Usage notes

The Resume method restarts a halted workflow, halted activity, or paused work item, or resumes the lifecycle of a SysObject by moving from the SysObject from its current exception state to the normal state it had before suspension.

### Resuming workflows, activities, and work items

Using the Resume method has the following constraints:

- Only the workflow supervisor or a user with Sysadmin or Superuser privileges can resume a halted workflow or paused work item.
- By default, only the workflow supervisor or a user with Sysadmin or Superuser privileges can resume a halted activity. However, if workflow_security_disabled, a server.ini key, is set to T (TRUE), any user can resume the activity.
- To resume a halted activity instance, the workflow must be in an active state.

When you resume a halted activity, the activity's state changes to running and any related work items revert to the state they were in when the activity was halted.

When you resume a halted workflow:

- The workflow's state changes to running.
- The current activities' states change to running.
- Any work items revert to the state they were in when the workflow was halted.

When you resume a paused work item, the work item's state changes to dormant.

## Resuming lifecyles

If you are the object's owner or a superuser, you must have Write permission to resume an object in a lifecycle. Other users must have Write permission and Change State permission for the object. If you have only Change State permission, Content Server attempts to resume the object as the user defined in the a_bpaction_run_as attribute in the docbase config object. In those instances, that user must be either the owner or a superuser with Write permission or have Write and Change State permission on the object.

You can only resume an object that has been suspended.

The current state must be the exception state defined for the normal state described by *from_state* or *from_position*.

If the requirements for the normal state are not met, the method fails, unless you specify overrirde_flag= TRUE, which sets r_current_state to destination_state and commits the change without performing the actions associated with the specified state. You must be a lifecycle owner or have Superuser user privileges to set the override_flag to TRUE.

### Testing before Resume

If the test_flag is TRUE, the Resume method launches the bp_transition method to test the requirements of the next state.

### Scheduling Resume

If you specify a *scheduled_date* and do not specify cancel, the Resume method adds a new job object whose method_name is bp_transition and with proper values for bp_transition as arguments and due_time as scheduled_time. The object name for the job has the following format:

Bp*sysobj_id scheduled date*

The scheduled_date is in the *yyyymmddhhmiss* format where
*yyyy* is a 4-digit year
*mm* is a 2-digit month
*dd* is a 2-digit day
*hh* is a 2-digit hour
*mi* is a 2-digit minute
*ss* is a 2-digit second

Note that the same SysObject can not be scheduled for more than one state transition event at a specified time.

The bp_transition method is invoked by a job agent which periodically scans dm_job table to execute jobs which are overdue. The default scan time is 1 minute.

If you specify a scheduled_date and cancel is specified, the Resume method deletes the job with name equal to Bp*sysobj_id scheduled date.*

## Related methods

Abort, page 40
Pause, page 320
Suspend, page 453

## Examples

The following example places the specified workflow in the running state:

```
dmAPIExec(sprintf("resume,%s,%s",session,workflow_id));
```

The following example places task number 201 of the specified workflow in the running state:

```
dmAPIExec(sprintf("resume,%s,%s,%s",session,workflow_id,
"201"));
```

# Retrieve

## Purpose

Finds the object that satisfies a specified qualification.

## Syntax

```
dmAPIGet("retrieve,session,qualification")
```

## Arguments

**Table 2-136. Retrieve method arguments**

| Argument | Description |
|---|---|
| *session* | Identifies an open repository session. |
| *qualification* | Defines an object. The definition must take the form of that portion of a DQL SELECT statement that appears after the keyword FROM in the statement syntax. The first word in the qualification must be a type name. After this, the qualification can include a WHERE clause, an IN DOCUMENT clause, or a full-text SEARCH clause. |

## Return value

The Retrieve method returns the ID of the object that satisfies the specified qualification.

## Usage notes

The Retrieve method is a shortcut to issuing a full DQL query when you want to retrieve only one object. The system creates the object's ID and type as part of the query and fetches the object.

**Note:** To take advantage of persistent client caching for the returned object, use a Query_cmd method followed by a Fetch method instead of the Retrieve method.

## Related methods

Execquery, page 203
Execsql, page 205
Query, page 337
Readquery, page 342

## Example

The following example fetches the document that has the object name carrots and is located in the Vegetables cabinet.

```
cmd = "retrieve,%s,dm_sysobject where folder('%s',descend) and object_name='%s'";
dmAPIGet(sprintf(cmd, session,"/Vegatables","carrots"));
```

# Revert

## Purpose

Refetches an object from the repository or throws away unsaved permission changes.

## Syntax

```
dmAPIExec("revert,session,object_id [,acl_only]")
```

## Arguments

**Table 2-137. Revert method arguments**

| Argument | Description |
|---|---|
| *session* | Identifies an open repository session. |
| *object_id* | Specifies the object that you want to re-fetch from the repository. Use the object's ID or an indirect reference (*@object_id*) that points to the object. |
| *acl_only* | Indicates whether you want to throw away only the changes that you have made to the object's permissions. If set to TRUE, the server only throws away any unsaved permission changes. If set to FALSE, the server refetches the object from the repository.<br><br>This flag is only available if the specified object is a SysObject or SysObject subtype. |

## Return value

The Revert method returns TRUE if successful or FALSE if unsuccessful.

## Usage notes

The primary use of Revert is to provide a way to throw away unsaved changes that you have made to an object.

The Revert method searches the user's DMCL cache for a copy of the specified object. If the cached copy is different from the copy in the repository or if the object is not found in the cache, the method fetches the object from the repository.

Fetching an object using Revert is subject to the same rules and permissions as fetching an object using Fetch. (Refer to Fetch, page 209 for a description of Fetch and its use.)

If you use Revert without the *acl_only* flag, it compares the local (working copy) of a fetched object to the same object stored in the repository. If they are not the same, Revert refetches the object from the repository. If you specify the *acl_only* flag, Revert throws away only the changes that you have made to the object's permissions. It does not refresh the object from the repository. The *acl_only* flag is only available when *object_id* identifies a SysObject or SysObject subtype.

Reverting a load record object does not fetch the object from the repository, but instead, rolls back all the operations performed by the load. This means that all the objects that were loaded into the repository are removed from the repository.

**Note:** You cannot issue a Revert method for a dm_load_record object while an explicit transaction is open.

## Example

This example:

- Fetches the object
- Appends an author's name
- Reverts the object

```
err_flag = dmAPIExec("fetch," session "," doc_id)
if (err_flag == 0) {
err_msg = dmAPIGet("getmessage," session)
print "Error fetching the object"
print err_msg
exit
}

cmd_str = "append," session "," doc_id ",authors"
err_flag = dmAPISet( cmd_str, "J. Smith" )
if (err_flag == 0) {
err_msg = dmAPIGet("getmessage," session)
print "Error appending an author's name"
print err_msg
exit
}

err_flag = dmAPIExec("revert," session "," doc_id)
if (err_flag == 0) {
err_msg = dmAPIGet("getmessage," session)
print "Error reverting the object"
print err_msg
exit
}
```

# Revoke

## Purpose

Removes access control entries from an ACL object for a specified user or group.

## Syntax

```
dmAPIExec("revoke,session,object_id,accessor_name|alias
[,permit_type][,unused]
[,basic_permit|extended_permit|application_permit]")
```

## Arguments

**Table 2-138. Revoke method arguments**

| Argument | Description |
|---|---|
| *session* | Identifies an open repository session. |
| *object_id* | Directly or indirectly identifies the ACL from which you are removing the entry or entries. The *object_id* can be the object ID of the ACL itself, for a direct identification, or it can be the object ID of an object with which the ACL is associated, for an indirect identification.<br><br>If the object ID identifies a SysObject, you can use the object's ID or an indirect reference (*@object_id*) that points to the object.<br><br>**Note:** The behavior of the method varies depending on whether you use direct or indirect identification of the ACL. Refer to the Usage notes for details. |
| *accessor_name* | Identifies the user or group specified in the access control entry that you want to remove.<br><br>If *permit_type* is RequiredGroup or RequiredGroupSet, this argument must be set to a group name.<br><br>If you include this argument, do not include *alias*. |

| Argument | Description |
|---|---|
| *alias* | Defines an alias in place of *accessor_name*. Use this only when *object_id* is an ACL object ID.<br><br>Use this argument to remove aliased access control entries from template ACLs. The format of an alias is:<br><br>`%[alias_set_name.]alias_name`<br><br>*alias_set_name* is the name of an alias set object.<br><br>*alias_name* is a value in the alias_name attribute of the alias set object.<br><br>If *permit_type* is RequiredGroup or RequiredGroupSet, the alias must resolve to a group name. |
| *permit_type* | Identifies the type of entry that is being removed from the ACL. Valid values are:<br><br>AccessPermit<br>ExtendedPermit<br><br>With a Trusted Content Services license, the following values are also valid entries:<br><br>ApplicationPermit<br>AccessRestriction<br>ExtendedRestriction<br>ApplicationRestriction<br>RequiredGroup<br>RequiredGroupSet<br><br>The default is AccessPermit. |
| *basic_permit* | Identifies the base permission to be removed. Specify this as an integer value. Valid values are:<br><br>1, for None<br>2, for Browse<br>3, for Read<br>4, for Relate<br>5, for Version<br>6, for Write<br>7, for Delete<br><br>If this argument is included, *permit_type* must be AccessPermit or AccessRestriction and you may not include *extended_permit* or *application_permit*. |
| *extended_ permit* | Identifies the extended permission to remove. Valid values are:<br><br>change_location<br>change_owner<br>change_permit<br>change_state |

| Argument | Description |
|---|---|
| | delete_object<br>execute_proc<br><br>If this argument is included, *permit_type* must be ExtendedPermit or ExtendedRestriction and you may not include *basic_permit* or *application_permit*. |
| *application_permit* | A user-defined permission interpreted and enforced by an application.<br><br>If this argument is included, *permit_type* must be ApplicationPermit or ApplicationRestriction, and you may not include *basic_permit* or *extended_permit*. |

## Return value

The Revoke method returns TRUE if successful or FALSE if unsuccessful.

## Usage notes

The Revoke method removes access control entries from an ACL. Access control entries are recorded in a set of repeating attributes in the ACL. The values at corresponding index levels across the attributes constitute the individual access control entries in the ACL.(Managing ACLs, page 387, in *Content Server Administrator's Guide* describes ACLs and access control entries in detail.)

You can execute Revoke against a particular SysObject, to change the permissions for that object only, or you can execute Revoke against an ACL object, to change the permissions for all objects controlled by that ACL.

If you want to change the permissions only for one object, specify that object's ID in the *object_id* argument. Content Server copies the ACL associated with the object. The method applies your changes to the copy, and Content Server applies the new copy to the object. The original ACL is not affected.

If you want to change the ACL directly, specify the ACL's object ID in the *object_id* argument. The method then applies your changes to the ACL, and all objects referencing that ACL are affected by the changes.

### Permissions to use Revoke

To execute Revoke against a SysObject, you must own the SysObject, have change_permit permission on the SysObject, or have Sysadmin or Superuser privileges.

To execute Revoke against an ACL object, you must own the ACL or have Superuser privileges.

### Identifying the entry to remove

The arguments that you include determine which entry or entries are removed.

If you include only AccessPermit as the permit_type and the accessor name, without identifying a specific permit level, the method removes all AccessPermit entries for that user or group, which effectively removes the user or group's basic and extended permissions. This occurs because the basic and extended permissions are recorded in the same entry in the ACL. This models the behavior of prior releases and is backwards compatible. For example, the following method call removes the AccessPermit entry for JohnnyQ on a document:

```
dmAPIExec("revoke,s0,0900002356ac267f,AccessPermit,,JohnnyQ")
```

However, if you designate a specific permission for removal by including a *basic_permit* or *extended_permit* argument when the permit_type is AccessPermit, only that permission is removed. For example, suppose JohnnyQ's base permission level is 6 (Write permission) and his extended permissions are execute_procedure, change_location, and change_permit. The following method call removes his base permission but does not remove the entry because only the base permit is designated for removal. The AccesPermit entry remains in the ACL with the designated extended permissions for JohnnyQ.

```
dmAPIExec("revoke,s0,0900002356ac267f,AccessPermit,,JohnnyQ,6")
```

The specified permit level must match the value in the entry. For example, if JohnnyQ's base permission is 6 and you issued the following method, the method would fail:

```
dmAPIExec("revoke,s0,0900002356ac267f,AccessPermit,,JohnnyQ,5")
```

To remove application permissions, restricting entries, required groups, or a group set entry, you must issue separate Revoke methods. For example, suppose JohnnyQ has shred application permission and is restricted from the change_owner extended permission. To remove these entries, issue two Revoke methods, one for the application permission and one for the extended restriction:

```
dmAPIExec("revoke,s0,0900002356ac267f,ApplicationPermit,,JohnnyQ,shred")

dmAPIExec("revoke,s0,0900002356ac267f,ExtendedRestriction,,JohnnyQ,change_owner")
```

#### Specifying an alias

When you want to remove an entry in a template ACL that has an alias defined as the accessor_name, specify the alias as the accessor_name in the Revoke method.

Template ACLs are ACLs that have the acl_class attribute set to 1. Template ACLs typically have access control entries with one or more accessor names set to aliases in the format %[*alias_set_name*.]*alias_name*. Use the alias argument when you want to remove such an access control entry.

For more information about aliases, refer to Appendix A, Aliases of the *EMC Documentum Object Reference Manual*.

## Related methods

Grant, page 258

## Example

```
err = dmAPIExec ("fetch," session "," obj_id)
if (err == 0)
{
print "Error fetching the object '" obj_id "'"
print dmAPIGet ("getmessage," session)
exit
}

err = dmAPIExec ("revoke," session "," obj_id "," accessor_name)
if (err == 0)
{
print "Error revoking permit from accessor '"\
accessor_name "'"
print dmAPIGet ("getmessage," session)
exit
}

err = dmAPIExec ("save," session "," obj_id)
if (err == 0)
{
print "Error saving the object '" obj_id "'"
print dmAPIGet ("getmessage," session)
exit
}
```

# Save

## Purpose

Writes an object to the repository without creating a new version of the object.

## Syntax

```
dmAPIExec("save,session,object_id[,save_lock]
{,version_label}")
```

## Arguments

**Table 2-139. Save method arguments**

| Argument | Description |
|---|---|
| *session* | Identifies an open repository session. |
| *object_id* | Specifies the object that you want to save. Use the object's ID or an indirect reference (*@object_id*) that points to the object. |
| *save_lock* | Directs the server to hold the intention lock on the saved object. The default is F. Set this to T to hold the lock. You can only use this argument if the specified object was obtained from the repository using the Checkout method. |
| *version_label* | Defines an implicit and/or symbolic version label for the object. You can specify more than one label. If you are saving a new object (not yet in the repository), the default implicit label is 1.0 and the default symbolic label is CURRENT. If the object is currently in the repository, there are no default labels assigned if you do not specify any in the Save operation. |

## Return value

The Save method returns TRUE if successful or FALSE if unsuccessful.

## Usage notes

Using the Save method is subject to the following constraints:

- You must have Write permission on the object.

- The object cannot be immutable.
- You cannot use Save on existing objects of type dm_audittrail, dm_audittrail_acl, or dm_audittrail_group.

The Save method is generally used to save a newly created object in the repository. You can also use it when you want to save changes to an object without creating a new version. The Save method overwrites the version identified by *object_id* with the local copy of the object. If you have changed the content of the object, it also sets the object's a_archive attribute to FALSE.

When you save a new object, its lifecycle is set to null. Use the Attach command to set up policy-related attributes.

Note that if the object's special attribute _status indicates an error condition, the Save method will not succeed. You must reset the object's state before saving in such cases (refer to the Reset, page 380 method description).

Additionally, a save operation might fail if you fetched the object from the repository instead of checking it out. This happens because fetching an object does not place a lock on the object and other users may have checked out or fetched and saved the object while you were working on it. In such circumstances, when you try to save the object, the operation fails.

**Note:** You cannot Save a load record object while you are in an explicit transaction (a transaction started with a Begintran method).

### Saving process definitions

When you save a process definition, the Save method enforces uniqueness on identifiers of activities and links within the process definition.

### Saving activity definitions

Saving a new activity definition provides default values for the repeatable_invoke and r_definition_state attributes.

Saving an activity definition validates the following:

- Execution-related attributes
- Performer-related attributes
- Timer-related attributes
- The trigger_threshold value

The Save method enforces a unique combination of port_name and package_name.

If you change any critical attributes in an existing activity definition, the Save method verifies that the activity definition is in the draft state.

### Saving workflows

To save a workflow (a dm_workflow object), you must have RELATE permission on the process definition that the workflow references, or have Superuser or Sysadmin

privileges. The workflow supervisor must also have RELATE permission on the process definition that the workflow references, or have Superuser or Sysadmin privileges.

When you save a workflow, the server validates the supervisor_name (which is the creator_name unless set to a different name) and sets the r_runtime_state to dormant.

### Saving lifecycles (dm_policy objects)

To save a lifecycle:

- The user must have Write permission.
- The included_type attribute must be parallel to the include_subtypes attribute.
- The state-related attributes must be parallel to each other.
- The object name and the state names within the policy must be unique.

  Therefore, you can version an existing lifecycle, but you cannot create another lifecyle with the same name.

Saving a new lifecycle sets the r_definition_state to DRAFT.

## Related methods

Branch, page 131
Checkin, page 138
Saveasnew, page 409
Unlock, page 477

## Example

This example:

- Creates a document object
- Associates content of the specified file with the document
- Saves the document object

```
doc_id = dmAPIGet("create," session ",dm_document")
if (doc_id == NULL) {
err_msg = dmAPIGet("getmessage," session)
print "Error creating a document object"
print err_msg
exit
}

the_file = "chap_1.wp5"
cmd_str = "setfile," session "," doc_id "," the_file ",wp5"
err_flag = dmAPIExec( cmd_str )
if (err_flag == 0) {
err_msg = dmAPIGet("getmessage," session)
print "Error setting file contents"
```

```
print err_msg
exit
}

err_flag = dmAPIExec("save," session "," doc_id)
if (err_flag == 0) {
err_msg = dmAPIGet("getmessage," session)
print "Error saving the document object"
print err_msg
exit
}
```

# Saveasnew

## Purpose

Copies an object.

## Syntax

```
dmAPIGet("saveasnew,session,object_id,share_content[,keep_
storage_areas]")
```

## Arguments

**Table 2-140. Saveasnew method arguments**

| Argument | Description |
|---|---|
| *session* | Identifies an open repository session. |
| *object_id* | Identifies the object that you want to copy. This must be an object of type SysObject or a SysObject subtype.<br><br>Use the object's ID or an indirect reference (*@object_id*) that points to the object. |
| *share_content* | Determines if the content of the source object is shared with the new object or if the saveasnew operation creates a new copy of the content for the new object.<br><br>TRUE indicates that the new object and its source share the same content file. FALSE directs the server to create a new content file for the new copy.<br><br>The defaults depend on the scope of the operation. Refer to the Usage notes for details.<br><br>This flag is FALSE by default, indicating that the saveasnew operation creates a new content for the new copy. |
| *keep_storage_areas* | Whether to store the content files of the new copy in the same storage area or areas as original or in the default storage area of the new copy. This argument is only effective if share_content is FALSE.<br><br>*keep_storage_areas* is F (FALSE) by default, meaning to store the copies in the default storage area of the new copy. |

For a complete description of the behavior of this argument, refer to the Usage notes.

## Return value

The Saveasnew method returns the ID of the new copy.

## Usage notes

If the object you are copying has content, you must have at least Read permission for the object. If the object has no content, only Browse permission is required to copy it. If the repository is running under folder security, you must also have at least Write permission for every folder or cabinet to which the object is linked.

**Note:** The Saveasnew method uses the client local area to hold a temporary copy of the object being saved as new. (If client local areas become full, they can be cleaned up using Purgelocal method.)

### Saveasnew and retention policies

Retention policy assignments are not carried over to the new copy created by Saveasnew. If you want the new copy to be associated with a retention policy, you must explicitly assign the new copy to the retention policy.

### Saveasnew and virtual documents

Because the copy that you are creating is a new object, not a version of the specified object, the Saveasnew method does not copy virtual-document containment or version information to the new copy. This means that the new copy is not contained in any virtual document even if the original object belongs to one or more virtual documents.

However, if you create a virtual document using Saveasnew, that virtual document contains the same components as the document from which it was copied.

### Sharing content

By default, the share_content flag is FALSE, which means that the server will make a copy of the content file for the new document.

If the document you are copying is in the current repository (the repository to which you are currently connected), you can set the share_content flag to TRUE. If you do, the new object shares the same content as the source object. The operation creates a new object and then sets its content to be the same content as that of the source object.

If two or more objects share content, when you replicate or dump and load one of the objects, all the objects that share the content are replicated or dumped and loaded. For

example, suppose Doc A and Doc B share a content file in repository_1. If you replicate Doc A to repository_3, the replication job will replicate Doc B to repository_3 also.

You cannot set the share_content flag to TRUE if you are using Saveasnew to copy an object across repositories or if you are using it to copy a replica object.

When you use an indirect reference to identify a remote object, the new copy is created in the repository defined in the docbase_scope attribute of the session config object. In such cases, the *share_content* flag is forced to FALSE.

## Specifying the storage area for new content

If the share_content argument is FALSE, the method must determine where to store the new copies of the content and renditions. Where the content (primary or renditions) associated with the new object is stored is dependent first on the setting of the keep_storage_areas argument and secondly, on the values in the following two attributes:

- The default-_torage attribute of the object's type
- The a_storage_type attribute in the object

### If keep_storage_areas is FALSE

The keep_storage_areas argument is FALSE by default. In such cases, by default, any content associated with the new object is saved in the storage area defined by the default_storage attribute of the object's type. If you want to store it in an alternate location, you must explicitly set the object's a_storage_type to the desired storage area before you issue the Saveasnew method.

If an object is copied from a non-content addressed storage area to a content-addressed storage area and a retention period is required by the content-addressed storage, you must issue a Setcontentattrs method to define the retention period before issuing the Saveasnew method.

### If keep_storage_areas is TRUE

If you include the argument set to TRUE, where the content is stored depends on whether the object is being copied within a repository or across repositories.

If the copy is within a repository, the content is saved in the same storage area as the source of the copy operation. For example, if a document has a primary rendition in StoreA and secondary rendition in StoreB, the new copy of the primary rendition is placed in StoreA and the copy of the secondary rendition is placed in StoreB.

If the copy is across repositories, the method checks for the existence in the target repository of a storage area whose name and store type matches the storage area of the original content. If such a storage area exists, the new content is saved to that storage area. If no match exists, an error is returned.

## Default group

Which default group is assigned to the new object is controlled by a server.ini key called saveasnew_retain_source_group. This is a Boolean key. If the key is set to T (TRUE), the

new object is assigned the same default group as the original object from which it was copied. If the key is set to F (FALSE), the new object is assigned the default group of the user who issues the Saveasnew method.

### Copying folders and cabinets

If you are using Saveasnew to copy a folder or cabinet, you must change its object name before you execute the Saveasnew method. (Content Server does not allow you to create two folders or cabinets in the same place with the same name.)

### Copying lifecyle definitions

When you use Saveasnew to copy a lifecycle definition (dm_policy object), the Saveasnew method sets the r_definition_state to draft on the new copy and clones the following attributes:

- entry_criteria_id
- user_criteria_id
- action_object_id
- user_action_id
- type_override_id

## Related methods

Branch, page 131
Checkin, page 138
Save, page 405
Unlock, page 477

## Example

```
cmd = sprintf("saveasnew,%s,%s",session,obj_id);
new_id = dmAPIGet(cmd);
```

# Seek

## Purpose

Defines the starting position (byte range) for the next reading of a file retrieved using a Getcontent method.

## Syntax

```
dmAPIExec("seek,session,collection,position[,direction]")
```

## Arguments

**Table 2-141. Seek method arguments**

| Argument | Description |
|---|---|
| *session* | Identifies an open repository session. |
| *collection* | Identifies a collection returned by the Getcontent method. |
| *position* | Defines the byte position within the content file where you want to start reading the file. The position can be a positive or negative integer. |
| *direction* | Specifies the direction of the read. 'B' for Beginning (default), 'C' for Current, 'E' for End.<br><br>If this is defined, the position value is treated as an offset value. Refer to the Usage notes for details. |

## Return value

The Seek method returns TRUE if successful and FALSE if unsuccessful.

## Usage notes

You can only use the Seek method if the content file is stored in a file store storage area or content-addressed storage area. You cannot use Seek if the content is stored in turbo storage, blob storage, or external storage. Additionally, you may not use Seek if the storage area is configured to use compression or encryption.

The method is useful in applications that need to read specific portions of a content file. The method allows you to transfer only the needed portions to the application. The Seek method must be used in conjunction with the Getcontent and Next methods.

Issue the Getcontent content method first, which returns a collection identifier that contains one result object. The result object has an attribute (buffer_size) that indicates the size of the buffer used by the Next method when returning content. Each execution of Next reads a portion of the content file into the attribute. To obtain only the needed portion of the content, issue Getcontent followed by Seek, then issue the Next method. When the Seek method is issued before the Next method, the Next method will read into the buffer the portion of the content starting at the position defined by the Seek method. (Be sure to copy the contents of the buffer to an external file or location between Next methods. Each time you issue a Next method, the current content of the buffer is overwritten.)

**Note:** The maximum buffer size is 63000 bytes.

The starting position in the Seek method is defined by the *position* argument and optionally, the *direction* argument. The value in *position* is an offset value, interpreted as a number of bytes. The value in *direction* defines the starting point of the offset. The system counts the specified number of bytes forward or backward in the file, depending on the setting of direction and the value in position.

If *direction* is undefined or is defined as B, for beginning, the position value must be a positive integer. The starting position of the next read is the offset value counted from the beginning of the file.

If *direction* is defined as E, for end, the position value must be a negative integer. The starting position of the next read is the offset value counted back from the end of the file.

If *direction* is defined as C, for current, the position value may be either positive or negative. The system counts forward or backward, respectively, from the current position within the file, to determine the starting position for the next read.

**Note:** After reading beyond the end of the content, it may no longer be possible to seek to a different position in the content.

## Related methods

Getcontent, page 231

## Example

```
q0 = dmAPIGet("getcontent,c,<object id>");
dmAPIExec("next,c,q0");
seek = dmAPIGet("get,c,q0,_seekable");
if(seek){
```

```
dmAPIExec("seek,c,q0,10000");
pos = dmAPIGet("get,c,q0,_byte_position");
/* The value of pos is 10000 here */
dmAPIExec("next,c,q0");
buf = dmAPIGet("get,c,q0,_content_buffer");
/* The first character in buf is the character
 at position 10000 in the content */
dmAPIExec("seek,c,q0,0,B");
/* The first character in buf is now the first
character of the content */
dmAPIExec("seek,c,q0,-1,E");
dmAPIExec("next,c,q0);
/* The first character in buf is now the last
character of the content */
}
```

# Set

## Purpose

Sets the value of an object's attribute.

## Syntax

```
dmAPISet("set,session,object_
id,attribute[[index]][,pattern]","value|alias")
```

## Arguments

**Table 2-142. Set method arguments**

| Argument | Description |
|---|---|
| *session* | Identifies an open repository session. |
| *object_id* | Specifies the object that contains the attribute you want to set. Use the object's ID or an indirect reference (*@object_id*) that points to the object. For the configuration objects, use one of the following keywords:<br><br>apiconfig<br>sessionconfig<br>connectionconfig<br>docbaseconfig<br>serverconfig<br><br>Refer to the Usage notes for a description of the objects represented by the keywords. |
| *attribute* | Identifies the attribute you want to set. Use the attribute's name. |
| [*index*] | Specifies the row in the attribute at which you want to set the value. Use this only if the specified attribute is a repeating attribute. Note that you must enclose the index value in square brackets [ ]. |

| Argument | Description |
|---|---|
| *pattern* | For date attributes, defines the pattern that you want the server to use to interpret the value. Refer to the Usage notes for a list of accepted patterns.<br><br>If a pattern is not specified, the value is interpreted using the client's short date format if that is defined. Otherwise, the server uses one of the accepted character string date formats for date literals or the formats defined by the date keywords. |
| *value* | Defines the value to which you are setting the specified attribute.<br><br>If you include *value*, do not include *alias*. |
| *alias* | Defines an alias as the value to which you are setting the specified attribute. This argument is intended for use in applications. The format of an alias is:<br><br>`%[alias_set_name.]alias_name`<br><br>*alias_set_name* is the name of an alias set object.<br><br>*alias_name* is a value in the alias_name attribute of the alias set object. |

## Return value

The Set method returns TRUE if successful or FALSE if unsuccessful.

## Usage notes

The Set method sets the value of a specified attribute of an object. Each execution of Set sets one attribute value.

### Identifying the object

To identify the object containing the attribute, specify the object's ID or one of the following keywords:

- apiconfig

  This keyword represents the api config object, a non-persistent object that is constructed when the client issues a dmAPIInit call and is destroyed when the client issues the dmAPIDeInit call. This object's attributes reflect values in the client's dmcl.ini file. By setting these attributes, you can override the dmcl.ini values. Setting the connection broker-related attributes of this object affects your current session and

any other sessions started by Connect calls from your session. Setting the attributes that are not connection broker-related affects only sessions started by Connect calls from your session. Refer to Table 2–6, page 68 in the *EMC Documentum Object Reference Manual* for a list of its attributes.

Any user can set these attributes.

- sessionconfig

  This keyword represents the session config object, a non-persistent object that is constructed when the client connects to a repository and is destroyed when the client explicitly disconnects from the repository or when the application terminates. This object's attributes represent the operating parameters for that client's repository session. They are a merging of some attribute values from the client's api config and server config objects. Setting the server config object's attributes affects only your current session. It will not affect subsequent sessions opened with the same or another repository. It is not necessary to issue a Save method after setting a session config attribute. Refer to Session Config, page 441 in the *EMC Documentum Object Reference Manual* for a list of its attributes.

  Any user can set these attributes.

- connectionconfig

  This keyword represents a connection config object, a non-persistent object that is constructed when the client establishes a subconnection within a session. It is destroyed when the client closes the subconnection. The object's attributes represent the operating parameters for that repository connection. Refer to Table 2–34, page 140 in the *EMC Documentum Object Reference Manual* for a list of its attributes.

- docbaseconfig

  This keyword represents the docbase config object, a persistent object that is part of the repository's configuration definition. The attributes in this object control the repository's security level, the location of its events directory, and the location of the tablespace for type indexes. (Refer to Table 2–46, page 181 in the *EMC Documentum Object Reference Manual* for a list of its attributes.)

  You must have Sysadmin or Superuser user privileges to set the attributes of the docbase config object. After you modify an attribute and save the object, you must execute either a Reinit or Restart to make the changes visible to the server. Reinit reinitializes the root process server, affecting the current session and subsequent sessions with the server, and Restart reinitializes the current session's server.

- serverconfig

  This keyword represents a server config object, a persistent object whose attributes define the configuration for a server. Each server has a server config object.

  You must have Sysadmin or Superuser user privileges to change the attributes of a server config object. To make the changes visible to the server, execute the Reinit or Restart method after you save the server config object. The Reinit method reinitializes

the root process server, affecting the current session and subsequent sessions with the server. The Restart method reinitializes only the current sessions server.

### Setting repeating attributes

If you are setting a repeating attribute, you must include the index with the attribute's name in the arguments. The index value identifies the new value's position in the attribute's value list. Enclose the index value in square brackets. For example, the following Set method sets the first value in the keywords attribute for a document:

```
dmAPISet("set,s0,doc_id_1,keywords[0]","evaluation")
```

Index values begin with zero and increment by one with each addition to the attribute. You cannot skip an index value. For example, the following methods add two more keywords to the previous example:

```
dmAPISet("set,s0,doc_id_1,keywords[1]","hotel")
```

```
dmAPISet("set,s0,doc_id_1,keywords[2]","new_proposal")
```

The Set method does not insert values into repeating attributes. If you specify an index value for an attribute that has an existing value at that index level, the existing value is replaced by the new value. For example, the following statement replaces the hotel keyword with the new keyword mall:

```
dmAPISet("set,s0,doc_id_1,keywords[1]","mall")
```

### Specifying a pattern for dates

When the attribute you are setting is a date, you may need to ensure that the server interprets the value correctly. Some values can be interpreted as more than one date. For example, the value 04/05/95 could be interpreted as April 5, 1995 or as May 4, 1995. To ensure that the server interprets the date values in the Set method correctly, specify the pattern used for the value.

Valid patterns are:

*mm*/*dd*/[*yy*]*yy*
*dd*-*mon*-[*yy*]*yy*
*month dd*[,] [*yy*]*yy*
*mon dd* [*yy*]*yy*
[*dd*/]*mm*/[*yy*]*yy* [*hh*:*mi*:*ss*]
[*yy*]*yy*/*mm*[/*dd*] [*hh*:*mi*:*ss*]
[*mon*-][*yy*]*yy* [*hh*:*mi*:*ss*]
*month*[,] [*yy*]*yy* [*hh*:*mi*:*ss*]

For example, the following method call sets the plan_end_date attribute to June 4, 1996:

```
set,s0,1,plan_end_date,dd/mm/yy,04/06/96
```

Without the pattern specification, the server would interpret 04/06/96 as April 6, 1996.

If the pattern contains a comma, you must enclose the pattern in single quotes.

### Using aliases

Aliases are a feature that support client applications designed to execute in a variety of situations. For example, you can create a workflow definition that contains aliases in place of user names for activity performers. The aliases are resolved when the workflow is instantiated or during its runtime.

You can also write an application that creates a document or other SysObject and by setting the document's owner_name, acl_name, or acl_domain to an alias, ensure that the object will have appropriate values for these attributes when the application executes and the object is saved to the repository.

Aliases are resolved by a search through an ordered series of scopes. When the application executes in different situations, the possible resolutions in each scope can differ. (For a detailed explanation of how aliases are resolved, refer to Appendix A, Aliases of *Content Server Fundamentals*.)

You can use Set to place aliases in the owner_name, acl_domain, or acl_name attributes of a SysObject or the performer_name attribute of an activity object.

## Related methods

Append, page 77
Insert, page 271
Remove, page 351
Truncate, page 462

## Example

This example:

- Creates a document object
- Associates the specified file content with the document
- Set the author's name and the document's title
- Saves the document object

```
doc_id = dmAPIGet("create," session ",dm_document")
if (doc_id == NULL) {
err_msg = dmAPIGet("getmessage," session)
print "Error creating a document object"
print err_msg
exit
}

the_file = "chap_1.wp5"
cmd_str = "setfile," session "," doc_id "," the_file ",wp5"
err_flag = dmAPIExec( cmd_str )
if (err_flag == 0) {
err_msg = dmAPIGet("getmessage," session)
```

```
print "Error setting file contents"
print err_msg
exit
}

cmd_str = "set," session "," doc_id ",title"
err_flag = dmAPISet( cmd_str, "Reference Manual - Chapter One" )
if (err_flag == 0) {
err_msg = dmAPIGet("getmessage," session)
print "Error setting document title"
print err_msg
exit
}

cmd_str = "set," session "," doc_id ",authors"
err_flag = dmAPISet( cmd_str, "S. Ruppenthal" )
if (err_flag == 0) {
err_msg = dmAPIGet("getmessage," session)
print "Error setting authors' name"
print err_msg
exit
}

cmd_str = "save," session "," doc_id
err_flag = dmAPIExec( cmd_str )
if (err_flag == 0) {
err_msg = dmAPIGet("getmessage," session)
print "Error saving the document object"
print err_msg
exit
}
```

# Setbatchhint

## Purpose

Defines a maximum number of rows that can be returned to the server in each call to the underlying RDBMS.

## Syntax

```
dmAPIExec("setbatchhint,session,size")
```

## Arguments

**Table 2-143. Setbatchhint method arguments**

| Argument | Description |
|---|---|
| *session* | Identifies an open repository session. |
| *size* | Defines the maximum rows that you want returned to the server in each call to the underlying RDBMS. |

## Return value

The Setbatchhint method returns TRUE if successful or FALSE if unsuccessful.

## Usage notes

The Setbatchhint method allows you to performance tune your session. Each query that you make to the repository effectively queries the underlying RDBMS. If you know that the information coming back from the RDBMS for each row is relatively small, use Setbatchhint to set the allowed maximum number of returned rows for each RDBMS call to a higher number, which reduces calls to the RDBMS and consequently, provides better performance.

The server may choose to return fewer rows than the specified maximum. It will not return more than the number set with Setbatchhint. The default batch size is 20.

## Example

This example sets 100 as the proposed number of rows to be returned on each call to the database.

```
dmAPIExec("setbatchhint,s0,100")
```

# Setcontent

## Purpose

Sets the content of an object.

## Syntax

```
dmAPISet("setcontent,session,object_id[,format][,page_number]
[,length][,set_resource]","content")
```

## Arguments

**Table 2-144. Setcontent method arguments**

| Argument | Description |
|---|---|
| *session* | Identifies an open repository session. |
| *object_id* | Identifies the object whose content you are setting. Use the object's ID or an indirect reference (*@object_id*) that points to the object. |
| *format* | Specifies the file format of the content. Use any file format recognized by Content Server.<br><br>You must specify this argument if the content is the first content added to the object. If there is already existing content for the object, do not specify this argument; use the page_number argument instead. |
| *page_number* | Defines the position of the new content in the ordered list of the object's content. If unspecified, this defaults to zero, the first position. |
| *length* | Specifies the length, in bytes, of the data that you are adding as content. |
| *set_resource* | For files created on Macintosh systems, indicates whether you want the method to add the data fork (the file containing the content) or the resource fork. FALSE (the default) adds the data fork (the content file). TRUE adds the resource fork. |
| *content* | Specifies the location of the data in working memory. |

## Return value

The Setcontent method returns TRUE if successful or FALSE if unsuccessful.

## Usage notes

Use the Setcontent method in an application when you want to add data residing in working memory as content to an object. This method is not intended for direct use by a user. Instead, in an application, the method should interact with the user's system to take data residing in working memory and add it as content to a specified object.

If you are adding content created on a Macintosh machine, you must issue the Setcontent twice before saving the object. Issue the method once to add the data fork (set_resource=F) and then again to add the resource fork (set_resource=T). When you add the data fork, the content argument must identify the location in memory of the data fork. When you add the resource fork, the content argument must identify the memory location of the resource fork.

The data in working memory is not copied to the repository until a Save or Checkin is executed.

You cannot use the Setcontent method with objects stored in external storage areas.

## Related methods

Addrendition, page 66
Appendcontent, page 80
Appendfile, page 82
Bindfile, page 129
Insertcontent, page 274
Insertfile, page 277
Setfile, page 434
Setpath, page 439

## Example

```
doc_id = dmget("create,s0,dm_document");
if (doc_id == NULL)
{err_mess = dmget("getmessage,s0")
print err_mess
exit
}

status = dmset("set,s0,last,object_name", "my_document")
content = "This manual is for use with the B1.1 release"

status = dmset("setcontent,s0,last,text", content)
```

```
if (status != 1)
{  err_mess = dmget("getmessage,s0")
print err_mess
exit
}
status = dmexec("save,s0,last")
if (status != 1)
{  err_mess = dmget("getmessage,s0")
print err_mess
exit
}
```

# Setcontentattrs

## Purpose

Sets the content-related attributes in a content object.

## Syntax

```
dmAPIExec("setcontentattrs,session,object_id,format[,page_
number][,page_modifier],parameters")
```

## Arguments

**Table 2-145. Setcontentattrs method arguments**

| Argument | Description |
|---|---|
| *session* | Identifies an open repository session. |
| *object_id* | Object ID of the document that contains the content file represented by the content object. |
| *format* | File format of the content file. Use the name of the format object representing the format. |
| *page_number* | Page number of the content in the document's set of content files. The default is 0. |
| *page_modifier* | Identifies a rendition. Refer to the Usage notes for a detailed description of the purpose of the page modifier. |
| *parameters* | List of attribute name and value pairs. The entire list must be enclosed in single quotes. The list has the format<br>`'name=value{,name=value}'`<br>*value* is one of:<br>"*character_string*"<br>FLOAT(*number*)<br>DATE(*date_value*)<br>Seperate name and value pairs with commas. Refer to the Usage notes for examples. |

## Return value

Setcontentattrs returns TRUE if successful or FALSE if unsuccessful.

## Usage notes

Use Setcontentattrs if you want to set the content-related attributes of an object's content object when you add or modify the object's content.

**Note:** If you only want to change the content-related attributes without modifying the actual content in some manner, use the SET_CONTENT_ATTRS and PUSH_CONTENT_ATTRS administration methods. Setcontentattrs is only effective if the actual content is also changed.

The Setcontentattrs method sets five attributes of a content object:

- content_attr_name
- content_attr_value
- content_attr_num_value
- content_attr_date_value
- content_attr_data_type

These are repeating attributes. Setcontentattrs sets them once for each name and value pair specified in the parameters argument. The method sets content_attr_name to the name value and content_attr_data_type to the datatype of the value. The remaining attributes are set in accordance with the datatype of the value:

- If the value is a character string, the content_attr_value is set to the value and content_attr_date_value and content_attr_num_value are set to the default NULL values (NULLDATE and NULLINT).
- If the value is a numeric value, the content_attr_num_value attribute is set to the value and content_attr_date_value and content_attr_ value are set to the default NULL values (NULLDATE and NULLSTRING).
- If the value is a date value, the content_attr_date_value attribute is set to the value and content_attr_num_value and content_attr_ value are set to the default NULL values (NULLINT and NULLSTRING).

You must issue a Save after Setcontentattrs to commit the changes to the repository. If the content file represented by the content object is stored in a content-addressed storage system, saving the changes also saves the attribute values to the storage system. Content Server calls the plugin library to save the attribute values to the content-addressed storage area.

**Note:** The attributes and values whose values are actually saved to the storage area are those that are identified in the a_content_attr_name attribute of the ca store object. If an attribute named in the parameters argument is not also recorded in the

a_content_attr_name attribute of the ca store object, its value is not passed to the storage system.

### Using the parameters argument

The parameters argument identifies the content metadata that you want to set. The metadata is specified as a comma-separated list of name and value pairs in the parameters argument. The format is:

```
'name=value{,name=value}'
```

Because parameters is a string argument, enclose the full string in single quotes. Additionally, if *value* is a character string, enclose the individual value in double quotes. For example:

```
setcontentattrs,S0,09000001003612ca,text,0,,'name="Engr_archive"'
```

If the value is a numeric value, use the FLOAT function to specify the value:

```
setcontentattrs,S0,09000001003612ca,text,0,,'deptno=FLOAT(0012)'
```

If the value is a date, use the DATE function to specify the value:

```
setcontentattrs,S0,09000001003612ca,text,0,,'a_retention_date=DATE(09/30/2005)'
```

The values are set in the content object attributes in the order in which they are listed in the parameters argument. If a name specified in the parameters argument matches a name already recorded in content_attr_name, the existing value for that name is overwritten. If the name is not found in content_attr_name, it is appended to the list of names in content_attr_name and its value is appended to the appropriate attribute recording values.

### Using page_modifier

Use the page_modifier argument if the content file represents a rendition of the document. The page_modifier is an identifier that distinguishes the rendition from any other rendition in the same format associated with a particular content page.

There are no constraints on the number of renditions that you can create for a document. Additionally, you can create multiple renditions in the same format for a particular document. To allow users or applications to distinguish between multiple renditions in the same format for a particular document, define a page modifier.

Including the page_modifier argument in Setcontentattrs sets the page_modifier attribute of the content object. This attribute, along with three others (parent_id, page, i_format) uniquely identifies a rendition. Applications that query renditions can use the modifier to ensure that they return the correct renditions.

### Executing on an existing content object

You can execute Setcontentattrs on an existing content object. If you do, successful completion of the method generates a new content address for the content file. The new address is stored in the i_contents attribute of a subcontent object. If there already exists

a subcontent object that records the content addresses associated with the content file, the new address is appended to the i_contents list in that subcontent object.

## Related methods

There are no related DMCL API methods. However, there are two related administration methods:
PUSH_CONTENT_ATTRS, page 285
SET_CONTENT_ATTRS, page 307

## Example

The following excerpt from a script creates a document whose content is to be stored in a content-addressed storage area. The Setcontentattrs method call is shown in boldface.

```
Function create_doc(ByVal sess As String,
ByVal store_name As String, _
    ByVal filepath As String, ByVal format_str
 As String) As String

    Dim buff    As String
    Dim status  As Integer

    buff = "create," & sess & ",dm_document"
    id$ = dmAPIGet(buff)
    If id = "" Then
      Call err_handler2(sess,"create_doc",buff,"09...",id)
    End If
    buff = "set," & sess & "," & id & ",a_storage_type"
    status = dmAPISet(buff,store_name)
    If status <> 1 Then
        Call err_handler2(sess,"create_doc",buuff,1,status)
    End If
    buff = "set," & sess & "," & id & ",object_name"
    status = dmAPISet(buff, store_name)
    If status <> 1 Then
        Call err_handler2(sess,"create_doc",buff,1,status)
    End If

    buff = "setfile," & sess & "," & id & "," & f
ilepath & "," & format_str
    status = dmAPIExec(buff)
    If status <> 1 Then
        Call err_handler2(sess,"create_doc",buff,1,status)
    End If

    buff = "setcontentattrs," & sess & "," & id & ","
& format_str & ",0,," & "'description=""description"",
title=""title""'"

    status = dmAPIExec(buff)
    If status <> 1 Then
```

```
    Call err_handler2(sess,"create_doc",buff,1,status)
End If
```

# Setdoc

## Purpose

Sets the r_is_virtual_doc attribute of a SysObject.

## Syntax

```
dmAPIExec("setdoc,session,object_id,is_virtual_doc")
```

## Arguments

**Table 2-146. Setdoc method arguments**

| Argument | Description |
|---|---|
| *session* | Identifies an open repository session. |
| *object_id* | Identifies the object whose r_is_virtual_doc attribute you are setting. Use the object's ID or an indirect reference (@*object_id*) that points to the object. |
| *is_virtual_doc* | Identifies the document as a simple document or a virtual document (virtual documents are opened using the Virtual Document Manager).<br><br>Set this to T (TRUE) to identify the object as a virtual document. Set this to F (FALSE) to identify the object as a simple document. |

## Return value

The Setdoc method returns TRUE if successful or FALSE if unsuccessful.

## Usage notes

To use the Setdoc method, you must have either Version or Write permission on the specified object. You need Version permission if you want to version the object. Saving the object without versioning requires Write permission.

If you set an object's r_is_virtual_doc attribute to T (TRUE), when users open the document, the system opens the document in the Virtual Document Manager (VDM). This happens regardless of whether the object has components.

You can set an existing virtual document's r_is_virtual_doc attribute to FALSE. If you do, the system continues to open the object in VDM until all the components are removed from the object. After the object's components are removed, the system treats the object as a simple document.

## Example

```
err = dmAPIExec ("fetch," session "," obj_id)
if (err == 0)
{
print "Error fetching the object '" obj_id "'"
print dmAPIGet ("getmessage," session)
exit
}

err = dmAPIExec ("setdoc," session "," obj_id ",T")
if (err == 0)
{
print "Error setting r_is_virtual_doc attribute"
print dmAPIGet ("getmessage," session)
exit
}

err = dmAPIExec ("save," session "," obj_id)
if (err == 0)
{
print "Error saving the object '" obj_id "'"
print dmAPIGet ("getmessage," session)
exit
}
```

**Note:** obj_id is a document object.

# Setfile

## Purpose

Adds a content file to an object or replaces an existing content.

## Syntax

```
dmAPIExec("setfile,session,object_id,file_name
[,page_number|format][,other_file]")
```

## Arguments

**Table 2-147. Setfile method arguments**

| Argument | Description |
|---|---|
| *session* | Identifies an open repository session. |
| *object_id* | Identifies the object whose content you are setting. Use the object's ID or an indirect reference (*@object_id*) that points to the object. |
| *file_name* | Specifies the file that contains the content. Use either an absolute or relative path to specify the file.<br><br>The file must be located on the machine from which the Setfile is issued or in a directory visible to that machine. |
| *page_number* | Defines the position where you want to place the content within the ordered contents of the object. Use this argument when you are adding second or subsequent contents to an object or when you want to replace an existing content. The default is zero. |
| *format* | Specifies the content's file format. The format must be a valid file format. You must specify this argument if you are setting an object's first content. Do not use this argument when you are setting second or subsequent content for an object. |
| *other_file* | Specifies the file that contains the resource fork for a Macintosh document. You can use either an absolute or a relative path. If this is set, the path must be valid or the Setfile will fail.<br><br>If the content is stored in an external storage area, the other_file argument is ignored. |

## Return value

The Setfile method returns TRUE if successful or FALSE if unsuccessful.

## Usage notes

If you specify the *page number* of an existing content for the object, the existing content is replaced in the object by the content specified in *file_name*. When you use Setfile to replace an existing content, any renditions associated with the content you are replacing are removed also.

The optional argument, *other_file*, allows you to add content files created on a Macintosh to a document. Macintosh-created files have two parts, a data fork and a resource fork. When you add such a file to a document, you must add both parts. The *other_file* argument specifies the resource fork file. You can use either a relative path or an absolute path. You cannot use *other_file* with external storage.

If a user's session code page is UTF-8 and the client_os_codepage is Shift_JIS or EUC-KR, the server substitutes a hyphen (-) for any character in the file name that the server cannot map to an ASCII character.

Although no special permission is needed to issue a Setfile method call, you must have at least Version permission on the object to check it in after you use Setfile to set its content.

## Related methods

Addrendition, page 66
Appendcontent, page 80
Appendfile, page 82
Bindfile, page 129
Insertcontent, page 274
Insertfile, page 277
Setcontent, page 424
Setpath, page 439

## Example

This example:

- Creates a document object
- Associates the specified file content with the document
- Saves the document object

```
doc_id = dmAPIGet("create," session ",dm_document")
if (doc_id == NULL) {
err_msg = dmAPIGet("getmessage," session)
```

```
print "Error creating a document object"
print err_msg
exit
}

the_file = "chap_1.wp5"
cmd_str = "setfile," session "," doc_id "," the_file ",wp5"
err_flag = dmAPIExec( cmd_str )
if (err_flag == 0) {
err_msg = dmAPIGet("getmessage," session)
print "Error setting file contents"
print err_msg
exit
}

err_flag = dmAPIExec("save," session "," doc_id)
if (err_flag == 0) {
err_msg = dmAPIGet("getmessage," session)
print "Error saving the document object"
print err_msg
exit
}
```

# Setoutput

## Purpose

Specifies the output port for manual transition of a work item.

## Syntax

```
dmAPIExec("setoutput,session,workitem_id{,output_port_name}")
```

## Arguments

**Table 2-148. Setoutput method arguments**

| Argument | Description |
|---|---|
| *session* | Identifies an open repository session. |
| *workitem_id* | Identifies the work item. Use the object's ID or an indirect reference (*@object_id*) that points to the object. |
| *output_port_name* | Specifies one or more output port to receive packages. |

## Return value

The Setoutput method returns TRUE if successful or FALSE if unsuccessful.

## Usage notes

The Setoutput method identifies one or more output ports selected for use in a manual transition of an activity. If an activity's transition type is "manual", the work item performer chooses the output ports that receive packages when the activity is completed. The application must provide users with a means to identify the output choices and pass those choices to Setoutput as arguments.

The number of choices a particular performer can make may be limited by the value in the activity's transition_max_output_cnt attribute. This attribute defines a maximum number of outports that the work item performer can choose for the activity. (For more information about this attribute, refer to Limiting output choices in manual transitions, page 220 in *Content Server Fundamentals*.)

The work item must be in the acquired state when you use Setoutput.

Only the work item performer, the workflow supervisor, or a Superuser can use Setoutput.

## Related methods

Complete, page 153

## Example

```
status = dmexec("setoutput," session ",4a00007b80000101,
final_review")
```

# Setpath

## Purpose

Adds a file in an external storage area to a document as primary content.

## Syntax

```
dmAPIExec("setpath,session,object_id,file_name
[,page_number|format[,other_file]]")
```

## Arguments

**Table 2-149. Setpath method arguments**

| Argument | Description |
|---|---|
| *session* | Identifies an open repository session. |
| *object_id* | The object to which you are adding the content. Use the object's object ID. |
| *file_name* | The content file you are adding to the object. Use either an absolute or relative path to identify the file.<br><br>The file must be located on the machine from which the Setpath is issued or in a directory visible to that machine. |
| *page_number* | Defines the position in which to place the content within the ordered contents of the object. Use this argument when you are adding second or subsequent contents to an object or when you want to replace an existing content. The default is zero. |
| *format* | Specifies the content's file format. The format must be a valid file format. You must include this argument if you are adding an object's first content. Do not use this argument when you are setting second or subsequent content for an object. |
| *other_file* | Specifies the file that contains the resource fork for a Macintosh document. You can use either an absolute or a relative path. If this is set, the path must be valid or Setpath fails.<br><br>The other_file value is ignored when the content is stored in an external storage area. |

## Return value

The Setpath method returns TRUE if successful and FALSE if unsuccessful.

## Usage notes

Use Setpath when you want to add a content file to a document and that file is stored in an external storage area.

## Related methods

Getpath, page 253
Mount, page 312
Setfile, page 434

## Example

```
status = dmexec("setpath," session ",0900007b80005666,
c:\\dm\\dmcl.ini,text")
```

Object using the dm_filestore object type for storage:

```
dmAPIExec('create,c,dm_document')
dmAPIExec('set,c,l,object_name,test4')
dmAPIExec('setpath,c,l,c:\temp\out1,text')
dmAPIExec('save,c,l')
```

Object using the dm_extern_file object type for storage:

```
dmAPIExec('create,c,dm_document')
dmAPIExec('set,c,l,object_name,test1')
dmAPIExec('set,c,l,a_storage_type,external_store_01')
dmAPIExec('setpath,c,l,c:\temp\out1,text')
dmAPIExec('save,c,l')
```

# Setperformers

## Purpose

Sets the performer or performers for an activity.

## Syntax

```
dmAPIExec("setperformers,session,object_id,activity_name
[,performer_list]")
```

## Arguments

**Table 2-150. Setperformers method arguments**

| Argument | Description |
|---|---|
| *session* | Identifies an open repository session. |
| *object_id* | The object ID of either:<br>• The workflow that contains the activity identified in *activity_name*<br>• The work item whose performer is choosing the performers |
| *activity_name* | Identifies the activity for which performers are being chosen. |
| *performer_list* | A comma-separated list of user names, group names, or both chosen as performers for the activity. |

## Return value

The Setperformers method returns TRUE if successful or FALSE if unsuccessful.

## Usage notes

To specify the object ID of a workflow in the Setperformers method, you must have Sysadmin or Superuser privileges, be the workflow creator, or the workflow supervisor.

To specify the object ID of a work item in the method, you must be either:

- A performer of the activity whose completion is invoking the Setperformer method
- A user with Sysadmin or Superuser privileges

The Setperformers method is used in an application when an activity's performer or performers are chosen at workflow initiation or upon completion of a previous activity.

The application must present the workflow initiator or the activity performer with a list of the possible performers. After the user chooses the performers from the list, the application calls Setperformers to add the performers to the workflow.

Executing Setperformers sets the r_performers and r_perf_act_name attribute of the workflow object. These are repeating attributes. The r_performers attribute records the names of the performers and r_perf_act_name records the activity for which the performer is chosen. For example, suppose you execute the following Setperformers method:

```
setperformers,s0,workflow_id,review_activity,
'John Kirsh',helenp,maryw
```

Assuming that r_performers and r_perf_act_name were empty when the method executed, the method sets the following values:

| | |
|---|---|
| r_performers[0]=John Kirsch | r_perf_act_name[0]=review_activity |
| r_performers[1]=helenp | r_perf_act_name[1]=review_activity |
| r_performers[2]=maryw | r_perf_act_name[2]=review_activity |

If you execute Setperformers more than once specifying the same *object_id* and *activity_name*, the values set by previous executions are overwritten.

If you don't include performer_list in the method or if the method isn't executed, when the activity identified in review_activity executes, Content Server queues the work item to the workflow supervisor and sends the supervisor a message saying that no performers for the activity were found.

## Related methods

None

## Example

The following example sets performers of the activity named Activity_1 to userA and userB:

```
status=dmAPIExec("setperformers,s0," & wflow_id & ",Activity_1,userA,userB")
```

# Setpriority

## Purpose

Sets the priority of a work item in a workflow.

## Syntax

```
dmAPIExec("setpriority,session,workitem_id,priority")
```

## Arguments

**Table 2-151. Setpriority method arguments**

| Argument | Description |
|---|---|
| *session* | Identifies an open repository session |
| *workitem_id* | Object ID of the work item<br><br>The work item cannot be in the finished state and the workflow that generated the work item must be in the running state. |
| *priority* | New priority value for the work item. |

## Return value

The Setpriority method returns TRUE if successful or FALSE if unsuccessful.

## Usage notes

Use the Setpriority method to change the priority of a work item at runtime.

The priority value of a work item is derived from the priority value assigned to the activity that generated the work item. Priority values assigned to manual activities are ignored by Content Server. However, Content Server uses the priority values assigned to automatic activities to determine an order of execution for the generated work items.

The work item cannot be in the finished state, and the workflow must be in the running state.

### Permissions

By default, only the workflow supervisor or a user with Sysadmin or Superuser privileges can change a work item's priority. A work item performer cannot change the priority of a work item that he or she is performing. However, to allow any user, including a work item's performer, to change a work item's priority, set the workflow_security_disabled key in the server.ini file to T (TRUE).

**Note:** Setting workflow_security_disabled to TRUE also affects the ability to acquire and delegate work items and the ability to halt or resume a running activity. For details, refer to The enable_workitem_mgmt key, page 234, in *Content Server Fundamentals*.

### Effects of changing the priority

Changing a work item's priority has the following effects:

- Changes the priority of the work item's associated queue item, if one exists
- Generates a dm_changeprioritywork item event

  You can audit the dm_changeprioritywork item event.

## Related methods

None

## Example

```
status=dmAPIExec("setpriority,s0,4a00002211c592ac,5")
```

# Setsupervisor

## Purpose

Changes the supervisor of a workflow.

## Syntax

```
dmAPIExec("setsupervisor,session,workflow_id,new_supervisor")
```

## Arguments

**Table 2-152. Setsupervisor method arguments**

| Argument | Description |
|---|---|
| *session* | Identifies an open repository session |
| *workflow_id* | Object ID of the workflow<br><br>The workflow must be in a dormant, running, or halted state. |
| *new_supervisor* | Name of the new supervisor for the workflow.<br><br>This can be either an individual user or a group. The user or group must have at least Relate permission on the process object used to create the workflow. |

## Return value

Setsupervisor returns TRUE if successful or FALSE if unsuccessful.

## Usage notes

To use the Setsupervisor method, you must be the user who created the workflow, the workflow's current supervisor, or a user with Sysadmin or Superuser privileges.

Executing the method sends a dm_changeworkflowsupervisor event to the new supervisor's inbox.

It isn't necessary to issue a Save method after the Setsupervisor method.

## Related methods

None

## Example

```
dmAPIExec("setsupervisor,s0,4d000231000083cf,griselda")
```

# Shutdown

## Purpose

Shuts down Content Server.

## Syntax

```
dmAPIExec("shutdown,session[,immediate][,delete_entry]")
```

## Arguments

**Table 2-153. Shutdown method arguments**

| Argument | Description |
|---|---|
| *session* | Identifies an open repository session. |
| *immediate* | Directs the server to shut down either immediately, without waiting for open transactions to finish, or when all open transactions are finished. Setting the argument to T specifies an immediate shutdown. The default is F, shut down after open transactions are finished. |
| *delete_entry* | Indicates if the connection broker should delete its entry for the server. The default value is TRUE, which directs the connection broker to delete its entry for the server when the server shuts down. If this is set to FALSE, the connection broker does not delete its entry for the server. |

## Return value

The Shutdown method returns TRUE if successful or FALSE if unsuccessful.

## Usage notes

When you execute the Shutdown method, the session's server no longer accepts connection requests. However, if you shut down the server gracefully, the client processes continue to accept RPC calls until either:

- Any open transactions are completed, or
- The client process times out from lack of activity.

Shutting down gracefully only allows a client that has an open transaction with the server to complete the transaction. The client cannot initiate a new transaction.

An immediate shutdown stops the server immediately without allowing client processes to complete any open transactions.

Whenever you use shutdown to stop a server, the server sends a message to the connection broker, notifying the connection broker that the server is going down. When the connection broker receives the message, by default, it removes the server's entry. However, if you set the *delete_entry* flag to FALSE, the connection broker does not remove its entry for the server, but simply sets the server's status to down.

**Caution:** On Windows platforms, the Shutdown method does not use the Window service manager to shutdown the server. Consequently, all of the associated processes may not be shut down appropriately. Therefore, on Windows platforms, it is recommended that you shut down a server through the Windows service manager rather than using the Shutdown method.

## Example

This example shuts down the server connected to the specified session immediately.

```
dmAPIExec(sprintf("shutdown,%s,T", session_id));
```

# Signoff

## Purpose

Creates an audit trail entry of signoff information for an object.

## Syntax

```
dmAPIExec("signoff,session,object_id,[user_auth_name],
password[,reason]")
```

## Arguments

**Table 2-154. Signoff method arguments**

| Argument | Description |
|---|---|
| *session* | Identifies an open repository session. |
| *object_id* | Specifies the object to sign off. Use the object's ID or an indirect reference (*@object_id*) that points to the object. |
| *user_auth_name* | Identifies the signer. This is the value in the user's user_os_name attribute.<br><br>On Windows platforms, *user_auth_name* is specified with a domain name, in the following format:<br><br>`domain name\user_auth_name`<br><br>The default is the current user's user_os_name value. |
| *password* | Specifies the user's password. Valid password formats are:<br><br>*password*<br>DM_PLUGIN=*plugin_id/password*<br><br>where *plugin_id* is the plugin identifier for an authentication plugin.<br><br>If the password is encrypted, the format for the password argument is:<br><br>DM_ENCR_PASS=*encrypted_password*<br><br>where *encrypted_password* is one of<br><br>*password*<br>DM_PLUGIN=*plugin_id/password*<br><br>Refer to the Usage notes for details. |

| Argument | Description |
|---|---|
| | A password may not exceed 8,192 characters in length. |
| *reason* | Describes the purpose for which the signature is issued. |

## Return value

The Signoff method returns TRUE if successful or FALSE if unsuccessful.

## Usage notes

You must have Read permission on an object to sign it off. The Signoff method operates on the CURRENT version of an object.

Executing the Signoff method creates an audit trail entry with the following information:

**Table 2-155. Audit trail entry values for Signoff events**

| Attribute Name | Description |
|---|---|
| user_name | Signer's name |
| string_1 | The Windows OS domain and user name, used to validate the signature |
| string_2 | Reason for the signature |
| audited_obj_id | The ID of the object which has been signed |

Lifecycle designers can require a signature to fulfill promotion requirements. When a user promotes an object to the next state, the server evaluates the user-defined entry criteria to check for the required signatures by querying the dm_audittrail object tables.

Workflow designers can specify signoff of work items before the work item can be marked complete. The server enforces the requirement by not allowing a work item to be marked complete if it has not been signed off and the activity definition associated with the work item requires a sign-off.

### Authenticating the user

Content Server uses the values in the *user_auth_name* and *password* arguments to authenticate the user wishing to assume the connection. If the authentication is to be performed by an authentication plug-in module and the plug-in is not specified in the user's user_source attribute, you must prefix the password with the plug-in module's identifier. (The identifier is defined by the plugin module.) The format of a password with a plugin identifier is:

```
DM_PLUGIN=plugin_id/password
```

The plug-in identifier tells Content Server which authentication plug-in to use to authenticate the user. If Content Server cannot find the module, the method fails.

If a plug-in module is not identified in the arguments, Content Server authenticates the user by the mechanism defined in the user_source attribute of the user's dm_user object.

- If the attribute indicates that the user is authenticated against the operating system using the default mechanism, the user is authenticated using the default mechanism.
- If the attribute is set to LDAP, Content Server authenticates the user against the LDAP server.
- If the attribute is set to a plug-in identifier, Content Server authenticates the user using the plug-in module. If Content Server cannot locate the plug-in, the server authenticates the user using the platform's default mechanism.

### Using encrypted passwords

If you are using encrypted passwords, the *password* argument must specified as

```
DM_ENCR_PASS=encrypted_password
```

*encrypted_password* is the encrypted form of either of the valid password formats. The password must have been encrypted using the Encryptpass method.

When the DMCL receives the method, it decrypts the string using the AEK. The AEK must be initialized within the API session before the DMCL can execute the method. The recommended way to initialize the AEK is to use an Initcrypto method. If you do not use an Initcrypto method, the Signoff method looks for the AEK in the location identified in the environment variable DM_CRYPTO_FILE. If the AEK location is not identified in DM_CRYPTO_FILE, the method assumes it is in %DOCUMENTUM%\dba\secure\aek.key ($DOCUMENTUM/dba/secure/aek.key). When the key is found, the method initializes the AEK before proceding.

### Commas in user_auth_name and password

The value in the user_auth_name or password argument can contain commas. However, you must enclose the argument in single quotes if the value contains a comma. For example, suppose the user JaneDoe with the password "violet,eyes" is signing off the document represented by 09000002523a13c. The Signoff method would be:

```
dmAPIExec("signoff,s0,09000002523a13c,JaneDoe,'violet,eyes'")
```

## Related methods

Initcrypto, page 269

## Example

```
dmAPIExec(sprintf("signoff,%s,%s,%s",
session,workitem_id,password));
```

# Suspend

## Purpose

Suspends the lifecycle of an object.

## Syntax

```
dmAPIExec("suspend,session,sysobject_id
[,from_state|from_position,test_only_flag,override_flag]")
dmAPIExec("suspend,session,sysobject_id,scheduled_date
 ,[pattern],from_state|from_position|cancel[,override_flag]")
```

## Arguments

**Table 2-156. Suspend method arguments**

| Argument | Description |
|---|---|
| *session* | Identifies an open repository session. |
| *sysobject_id* | Identifies the object to suspend. Use the object's ID or an indirect reference (*@object_id*) that points to the object.<br><br>If the object ID identifies a replica object or you use an indirect reference, the operation affects the source object, not the local copy. |
| *from_state* | Identifies the state from which to suspend the SysObject. It is required for a scheduled suspend. |
| *from_position* | Identifies the position of the state from which to suspend the SysObject. |
| *test_only_flag* | TRUE or FALSE. When set to TRUE, Suspend evaluates the entry_criteria to determine if the document can be suspended. Default value is FALSE. |
| *override_flag* | TRUE or FALSE. When set to TRUE, Suspend proceeds even if the requirements of the destination state are not met. Only users with Superuser user privileges or the lifecycle owner can specify this argument. Default is FALSE. |
| *scheduled_date* | The date and time that the Suspend is scheduled for. |

| Argument | Description |
|---|---|
| *pattern* | The date-time format for *scheduled_date.* The pattern is defaulted to client-side localized date-time format, if not specified. |
| cancel | Cancel the scheduled Suspend. To cancel a scheduled Suspend, provide the original scheduled date. |

## Return value

The Suspend method returns TRUE if successful or FALSE if unsuccessful.

## Usage notes

The Suspend method suspends the lifecycle of an object by moving the object from its current normal state to the exception state of its current state.

If you are the object's owner or a superuser, you must have Write permission to suspend an object in a lifecycle. Other users must have Write permission and Change State permission for the object. If you have only Change State permission, Content Server attempts to suspend the object as the user defined in the a_bpaction_run_as attribute in the docbase config object. In those instances, that user must be either the owner or a superuser with Write permission or have Write and Change State permission on the object.

You cannot suspend an object whose current state is a normal state with no exception state, or if the object is already suspended.

The from_state or from_position must be the current state for a scheduled Suspend to succeed. The from_state and from_position attributes are mutually exclusive, and if you do not specify either one, the default is the current state. You must specify a from_state or from_position for a scheduled suspend.

If the requirements for the next state are not met, the method fails, unless you specify overrirde_flag=TRUE. If the override_flag is set to TRUE and if the user is a lifecycle owner or has Superuser user privileges, the method sets r_current_state to the exception state and commits the change without performing the actions associated with the exception state.

### Testing before promotion

If the test_flag is TRUE, the Suspend method launches the bp_transition method to see if the requirements of the exception state are met.

### Scheduling promotion

If you specify a scheduled_date, the Suspend method adds a new job object whose method_name is bp_transition. The object name for the job has the following format:

Bp*sysobj_id scheduled date*

The scheduled_date is in *yyyymmddhhmiss* format where

*yyyy* is a 4-digit year
*mm* is a 2-digit month
*dd* is a 2-digit day
*hh* is a 2-digit hour
*mi* is a 2-digit minute
*ss* is a 2-digit second

You cannot schedule the same SysObject for more than one state transition event at a specified time.

If you specify a scheduled_date and cancel is specified, the Suspend method deletes the job with name equal to Bp*sysobj_id scheduled date*.

## Related methods

Resume, page 392

## Example

```
status = dmAPIExec("suspend," sess "," doc_id ","
 dat_tim ",,state1")
```

# Trace

## Purpose

Turns on tracing for the session or returns information about tracing options.

## Syntax

```
dmAPIExec("trace,session,severity_level[,logfile],
option|facility|all")
```

## Arguments

**Table 2-157. Trace method arguments**

| Argument | Description |
|---|---|
| *session* | Identifies an open repository session. |
| *severity_level* | Defines a minimum severity level for the trace messages sent to the log file. The server sends all messages at or below the specified level to a session log file. The severity_level is specified as an integer value. Valid values and their corresponding severity levels are:<br><br>0, tracing is turned off<br>1, Level 1 trace messages<br>2, Level 2 trace messages<br>3, Level 3 trace messages<br>4, Level 4 trace messages<br>8, Dump and load object information<br>10, Timing information; commits after every loaded object<br>11, Load operation information<br><br>Turning on tracing also turns on tracing to the log file.<br><br>Informational, warning, error, and fatal error messages are always written to the session log file, even if tracing is turned off. |
| *logfile* | Specifies the location of the log file. If unspecified, the default is the api.log file in the current directory. |
| *option* | Returns information about the specified trace option. See Usage notes, page 457 for a list of the options. |

| Argument | Description |
|---|---|
| *facility* | Returns information about the specified trace facility. See Usage notes, page 457 for a list of the facilities. |
| all | Returns information about all available trace options and facilities. |

## Return value

The Trace method returns TRUE if successful or FALSE if unsuccessful.

## Usage notes

The Trace method turns on tracing. Trace messages are sent to two types of log files:

- The session log file records messages determined by the specified severity level. Informational, warning, error, and fatal error messages are always written to the session log file, even if tracing is turned off.
- The api.log file, on the client, records all user API method calls and the responses generated by the API.

The *severity_level* only affects the content of the session log file. For client log files, the *severity_level* must be greater than 0 to turn on tracing, but has no other effect.

The session log file is 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. A separate session log is created for each new connection.

To stop tracing, issue the Trace method and specify zero for the *severity_level*.

### Tracing severity levels

Severity levels from 1 through 10 are additive. In addition to the messages for the level you specify, you also receive all the other messages specified by the lower levels. 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.

#### Severity level 8

If you set the tracing level to 8, Content Server traces every dumped and loaded object.

#### Severity level 10

If you set the tracing level to 10, you receive information about how long each API method takes to execute. When you turn off level 10 tracing, you receive information about the total time of execution.

To illustrate, here are a few sample commands and the timing trace information they generate if the severity level is set to 10:

```
# Time0.00 sect 'OK'
create,c,dm_document
# Time0.024 sec '090007d080037572'
setfile,c,l,api.c,text
# Time1.924 sec 'OK'
save,c,l
# Time0.872 sec 'OK'
```

#### Severity level 11

The severity level 11 returns information about the available trace facilities and options.

For example, to see a list of the available trace facilities and options:

```
dmAPIExec("trace,c,11")
```

To see whether the API tracing facility is on and to get instructions on how to turn on the API tracing facility:

```
dmAPIExec("trace,c,11,,DM_API")
```

You can use severity level 11 in debugging if you are having a problem with a load operation. Generally, load operations return status messages to the server log file after every 100 objects are loaded. If you are having a problem with a load, you can set the trace level to 11 and the operation will return status messages for each object loaded. However, this does severely impact the performance of the load operation.

### Tracing options

Options are server-based, and capture tracing information in the server log. Turn trace options on through the Apply method.

The following table lists the tracing options.

**Table 2-158. Tracing options for Trace method**

| Facility | Meaning |
|---|---|
| debug | Records session shutdown information, change check information, launch and fork information in the server log. |
| lock_trace | Records lock information for shared access in the server log. This flag is for NT only. |
| sqltrace | Records all SQL commands generated from DQL commands in the server log. This flag affects subsequent sessions b ut it does not affect the current session. |

| Facility | Meaning |
|---|---|
| nettrace | Records the connection ID, client host address, client host name, and all Netwise calls in the server log, To turn this flag off, disconnect all client sessions. The change will take effect about 1 minute after the clients are disconnected. |
| trace_authentication | Records the authentication status for each authentication call in the server log |
| net_ip_addr | Records the client and server IP addresses in the server log during back-door authentication |
| trace_complete_launch | Records the process launch information for the parent process in the server log. |
| docbroker_trace | Enables connection broker trace for all sessions |

### Tracing facilities

Facilities are session-based, and capture tracing information in the session log. Turn trace facilities on through the Trace method.

The following table lists the tracing facilities.

**Table 2-159. Tracing facilities for Trace method**

| Option | Meaning |
|---|---|
| DM_API | Enables client API tracing. When this flag is on, the client API method calls are traced in the api.log file or a user specified log file. |
| DM_CONTENT | Enables content tracing in the session log. |
| DM_DUMP | Enables dump tracing in the session log. |
| DM_FULLTEXT | Enables full-text tracing in the server log. |
| DM_LOAD | Enables load tracing in the session log. |
| DM_QUERY | Enables query tracing in the session log. |
| SQL_TRACE | Enables session based SQL trace. When this flag is on, all SQL commands to the underlying RDBMS are stored in the session log. |
| DM_SYSOBJECT | Enables system object tracing in the session log. |

## Related methods

Getmessage, page 251
Listmessage, page 302

## Example

This example starts Level 10 tracing for client API methods in the api.log file.

```
dmAPIExec('trace,c,10,,DM_API
...
OK
API> retrieve,c,dm_user
...
110007b680000100
API> retrieve,c,dm_server_config
...
3d0007b680000101
API> trace,c,0,,DM_API
…
OK
```

Here is the corresponding API log file:

```
# Time:        0.000 sec  'OK'
API> retrieve,c,dm_user
# Network Requests: 10
# Time:        2.244 sec  '110007b680000100'
API> retrieve,c,dm_server_config
# Network Requests: 17
# Time:        0.782 sec  '3d0007b680000101'
API> trace,c,0,,DM_API
# Network Requests: 21
# Total Time:       38.686 sec  ' OK '
```

This example shows SQL tracing and the information returned by severity level 11.

```
API> trace,c,11,,SQL_TRACE
...
OK
API> getmessage,c
...
[DM_API_I_SQL_TRACE_USAGE]info:  "SQL Trace is
currently (off). The api command to turn on[/off]
 SQL trace is: trace,<session>,1[/0],,SQL_TRACE"

API> trace,c,1,,SQL_TRACE
...
OK
API> retrieve,c,dm_user
...
110007b680000100
API> trace,c,11,,SQL_TRACE
...
OK
API> getmessage,c
...
```

```
[DM_API_I_SQL_TRACE_USAGE]info:  "SQL Trace is
currently (on). The api command to turn on[/off]
SQL trace is: trace,<session>,1[/0],,SQL_TRACE"

API> trace,c,0,,SQL_TRACE
...
OK
API> trace,c,11,,SQL_TRACE
...
OK
API> getmessage,c
...
[DM_API_I_SQL_TRACE_USAGE]info:  "SQL Trace is
currently (off). The api command to turn on[/off]
SQL trace is: trace,<session>,1[/0],,SQL_TRACE"

API> retrieve,c,dm_server_config
...
3d0007b680000101
```

Here is the corresponding session log:

```
Tue Aug 11 11:36:52 1998 277000
[DM_SESSION_I_SESSION_START]info:
"Session 010007b680000560 started for user zhan."
select all dm_user.r_object_id from dm_user_sp dm_user
SELECT G_.I_TYPE FROM zhan.dmi_object_type G_
WHERE G_.R_OBJECT_ID=:
handle
:handle = 110007b680000100
SELECT * FROM dm_user_s OB_  WHERE
OB_.R_OBJECT_ID=:handle
:handle = 110007b680000100
Tue Aug 11 11:47:28 1998 001000
[DM_SESSION_I_SESSION_PAUSE]info:
"Session 010007b680000560 paused."
Tue Aug 11 11:48:47 1998 475000
[DM_SESSION_I_SESSION_START]info:
"Session 010007b680000560 started for user zhan."
```

Here is a sample of the Trace method:

```
status = dmexec("trace," session ",10,c:\trace.api")
```

# Truncate

## Purpose

Removes all the values in a repeating attribute.

## Syntax

```
dmAPIExec("truncate,session,object_id,attribute")
```

## Arguments

**Table 2-160. Truncate method arguments**

| Argument | Description |
|---|---|
| *session* | Identifies an open repository session. |
| *object_id* | Identifies the object that contains the repeating attribute. Use the object's ID or an indirect reference (*@object_id*) that points to the object. |
| *attribute* | Specifies which repeating attribute you want to truncate. Use the attribute's name. |

## Return value

The Truncate method returns TRUE if successful or FALSE if unsuccessful.

## Usage notes

In addition to removing all values in the specified repeating attribute, the Truncate method sets the attribute's count to zero. (The attribute's count is stored in the repeating special attribute _count, in the row corresponding to that attribute.)

## Related methods

Remove, page 351

## Example

This example:

- Creates a document object
- Appends the authors' names
- Truncates the list of authors

```
doc_id = dmAPIGet("create," session ",dm_document")
if (doc_id == NULL) {
err_msg = dmAPIGet("getmessage," session)
print "Error creating a document object"
print err_msg
exit
}

the_file = "chap_1.wp5"
cmd_str = "setfile," session "," doc_id "," the_file ",wp5"
err_flag = dmAPIExec( cmd_str )
if (err_flag == 0) {
err_msg = dmAPIGet("getmessage," session)
print "Error setting file contents"
print err_msg
exit
}

author_list = "W. Smith,J. Jones,R. Abnous,S. Ruppenthal"
count = split(author_list, author, ",")
cmd_str = "append," session "," doc_id ",authors"
for (i=1; i<=count; i++) {
err_flag = dmAPIExec( cmd_str, author[i] )
if (err_flag == 0) {
err_msg = dmAPIGet("getmessage," session)
print "Error appending authors' name to the 'authors' attribute"
print err_msg
exit
}
}

cmd_str = "truncate," session "," doc_id ",authors"
err_flag = dmAPIExec( cmd_str )
if (err_flag == 0) {
err_msg = dmAPIGet("getmessage," session)
print "Error truncating the authors' names"
print err_msg
exit
}
```

# Type

## Purpose

Returns an identifier that is used in read-only calls to obtain the data dictionary information for a type or attribute.

## Syntax

```
dmAPIGet("type,session,object_type[,attribute]
[,bus_policy_id,state]")
```

## Arguments

**Table 2-161. Type method arguments**

| Argument | Description |
|---|---|
| *session* | Identifies an open repository session. |
| *object_type* | Identifies the type you are inquiring about or, if you are inquiring about an attribute, the type to which the attribute belongs. Use the type's name; for example, dm_document or dm_folder. |
| *attribute* | Identifies the attribute for which you want to object data dictionary information. Use the attribute's name. |
| *bus_policy_id* | Identifies the lifecycle associated with the type specified in *object_type*. Use the policy's object ID. *bus_policy_id* must identify a policy that can be attached to objects of the type you identify in *object_type*.<br><br>If you include this, you must also include the state argument.<br><br>Refer to Usage notes for an explanation of the purpose of these arguments. |
| *state* | Identifies a particular state in the lifecycle. |

## Return value

The Type method returns an identifier of the form tdm_*typename*[.*attribute_name*] that points to an object. If you inquired about an object type, the object is a dmi_dd_type_info

object. If you inquired about an attribute, the object is a dmi_dd_attr_info object. (Refer to Chapter 2, Object Reference , in the *EMC Documentum Object Reference Manual* for a description of the attributes in these objects.)

## Usage notes

Use the Type method, in conjunction with a Dump or Get method, to obtain current data dictionary information for a type or attribute. The type or attribute may have some information that is overridden when an object is in a specific business state. If you include the optional *bus_policy_id* and *state* arguments, the method returns the information defined for the business state instead of the default information.

Using the identifier returned by the Type method in a Dump method returns all of the attributes in the object represented by the identifier. For example, the following Dump method returns all the attributes in the dd type info object for the document type:

```
dmAPIGet("dump,s0,tdm_document")
```

**Note:** If the identifier represents a type, the returned attributes include the type's dm_type and dmi_type_info attributes.

The following Dump method returns all the attributes in the dd attr info object for the document type's subject attribute:

```
dmAPIGet("dump,s0,tdm_document.subject")
```

Using the identifier in a Get method returns the value in the object for one attribute. For example, the following Get method returns the value in the dd attr info object of the not_null attribute for the document type's subject attribute:

```
dmAPIGet("get,s0,tdm_document.subject,not_null")
```

If you request information about an attribute, you can specify any object type for which the attribute is defined or inherited. Only the attribute's information is returned. You do not receive any information about the containing type. However, attribute information can differ depending on the object type. (Because it is possible to override attribute values, an attribute's domain definition may differ from type to type.)

## Example

The following example uses the Type method to help determine the number of attributes in the dm_document type.

```
type_id = dmAPIGet("type,"session ",dm_document");
count = dmAPIGet("get," session "," type_id ",attr_count")
print "Documents have "count" attributes";
```

# Unaudit

## Purpose

Stops auditing of an event.

## Syntax

```
dmAPIExec("unaudit,session[,object_id],event_name
[,controlling_app][,policy_id ][,state_name]]")
```

## Arguments

**Table 2-162. Unaudit method arguments**

| Argument | Description |
|---|---|
| *session* | Identifies an open repository session. |
| *object_id* | Identifies the particular object or object type for which the event is being audited. Use the object's object ID for individual objects. For object types, use the object ID of the type's dm_type object.<br><br>If this is 0 or omitted, auditing is stopped for all occurrences of the specified event. |
| *event_name* | Defines the event to stop auditing. |
| *controlling_app* | An application code. If this argument is set, Content Server stops the auditing of the event for those objects whose a_controlling_app value is set to this application code.<br><br>Setting this argument to dm_all_applications is equivalent to not including the argument. Auditing for the event is stopped for all objects of the specified type regardless of their controlling application.<br><br>You cannot include this argument if:<br>• *object_id* is set to 0 or omitted<br>• *object_id* identifies a particular object<br>• *event_name* is dm_all, all, or dm_all_workflow |

| Argument | Description |
|---|---|
| *policy_id* | Identifies a lifecycle. Use the object ID of the lifecycle's dm_policy object. If the event is dm_bp_attach, you can set *policy_id* to the keyword default.<br><br>If *policy_id* is set and *object_id* identifies an object type, Content Server stops auditing the event for objects of the particular type attached to this lifecycle. If *object_id* identifies an individual object, Content Server stops auditing the event for the object when it is attached to this lifecycle.<br><br>You cannot include this argument if:<br>• *object_id* is set to 0 or omitted<br>• *event_name* is dm_all, all, or dm_all_workflow |
| *state_name* | Identifies a particular state in the lifecycle identified in *policy_id*. Use the state's name.<br><br>If *state_name* is included and *object_id* identifies an object type, Content Server stops auditing the event for objects of the type that are in this particular lifecycle state. If *object_id* identifies an individual object, the object is audited for this event only when the object is in this lifecycle state.<br><br>*state_name* is ignored unless you also include *policy_id*.<br><br>You cannot include this argument if:<br>• *object_id* is set to 0 or omitted<br>• *event_name* is dm_all, all, or dm_all_workflow |

## Return value

The Unaudit method returns TRUE if successful or FALSE if unsuccessful.

## Usage notes

The Unaudit method stops auditing of an event and destroys the registry object associated with the event's auditing registration. Executing Unaudit does not destroy any audit trail records created while the event was being audited. You must have Config_audit privileges to use this method.

Each Unaudit method removes the audit registration created by one Audit method and stored in the repository as a registry object. You cannot use one Unaudit method to stop

auditing all events against one object or object type. If there are audit registrations for multiple events against the same object or object type, you must remove each audit registration with a separate Unaudit method call. Auditing is initiated by issuing an Audit method that identifies a particular event and an object or object type to be audited when that event occurs. To stop the auditing, you must issue an Unaudit method that identifies that same event and object or object type.

If there is no registry object that matches the parameters in the Unaudit method, the method fails with an error.

Executing the Unaudit method also creates an audit trail entry with the event name dm_unaudit to record the execution of the Unaudit method.

## Related methods

Audit, page 115

## Example

```
status = dmAPIExec("unaudit," session ",0900007b80005640,
dm_save")
```

# Unfreeze

## Purpose

Unfreezes a frozen object and, optionally, if the object is a virtual document, the object's associated assembly.

## Syntax

```
dmAPIExec("unfreeze,session,object_id[,thaw_components]")
```

## Arguments

**Table 2-163. Unfreeze method arguments**

| Argument | Description |
|---|---|
| *session* | Identifies an open repository session. |
| *object_id* | Specifies the object that you want to unfreeze. Use the object's ID or an indirect reference (*@object_id*) that points to the object.<br><br>If you are unfreezing the assembly for a virtual document (setting thaw_components to T), object_id must identify the document with which the assembly is associated. Generally, this is the virtual document. However, a virtual document's assembly is sometimes associated with a simple document. In such cases, specify the simple document's object ID in this argument. |
| *thaw_components* | Indicates whether you want to unfreeze the assembly associated with the specified object. Set this to T to unfreeze the assembly. The default is F (FALSE). You can use this flag only if the specified object has an associated assembly. Refer to the Usage notes for more details. |

## Return value

The Unfreeze method returns TRUE if successful or FALSE if unsuccessful.

## Usage notes

The Unfreeze method unfreezes a frozen object. A frozen object is an object that has been marked as unchangeable by the Freeze method. With one exception, unfreezing a frozen object makes it changeable once again.

The exception occurs when you unfreeze an older version of an object—the object remains unchangeable because objects that have been versioned are always unchangeable. This situation occurs primarily in the context of virtual documents. Sometimes you might wish to freeze an assembly created for an older version of a virtual document. In such cases, if you later unfreeze the assembly, the document remains unchangeable.

Unfreezing an object sets the following attributes of the object to F (FALSE):

- r_immutable_flag

  This attribute indicates whether the object is unchangeable. Note that if the object has been versioned, unfreezing the object *does not* unset this attribute; it remains set to T.

- r_frozen_flag

  This attribute indicates whether the object is unchangeable because it was specifically frozen.

If you set the *thaw_components* flag in the Unfreeze method, then the following attribute of the specified document is set to F (FALSE):

- r_has_frzn_assembly
- This attribute indicates whether the document's assembly is frozen.

When you unfreeze an assembly, the method changes the following attributes for each component in the assembly:

- r_immutable_flag
- This attribute is set to F.
- r_frzn_assembly_cnt

  This attribute, which maintains a count of the number of frozen assemblies to which the object belongs, is decremented by one.

## Related methods

Freeze, page 221

## Example

In this example, doc_id is a variable that contains the object ID of the document that has the associated assembly.

```
cmd = sprintf("unfreeze,c," doc_id ",T")
status = dmAPIExec(cmd)
if (status != 1)
  {
```

```
    error_msg = dmAPIGet("getmessage,s0");
    printf "%s", error_msg;
  }
cmd = sprintf("save,c," doc_id)
status = dmAPIExec(cmd);
if (status != 1)
  {
    error_msg = dmAPIGet("getmessage,s0");
    printf "%s", error_msg;
  }
```

# Uninstall

## Purpose

Uninstalls an installed lifecycle, workflow, or activity definition.

## Syntax

```
DmAPIExec("uninstall,session,object_id[,notify_flag]")
```

## Arguments

**Table 2-164. Uninstall method arguments**

| Argument | Description |
|---|---|
| *session* | Identifies an open repository session. |
| *object_id* | Identifies the lifecycle, activity or process definition that you want to uninstall. Use the object ID. |
| *notify_flag* | A boolean flag that indicates whether to inform owners of SysObjects associated with this policy. The default is TRUE. |

## Return value

The Uninstall method returns TRUE if successful or FALSE if unsuccessful.

## Usage notes

The Uninstall method moves a lifecycle (also called a document life cycle), process definition, or activity definition object from the installed state to the validated state. Uninstalling a process or activity definition halts any running workflows that reference the uninstalled process.

The lifecycle, process, or activity object has to be already installed.

You must have Relate permission or Sysadmin or Superuser privileges to use the Uninstall method.

For business policies, if you specify notify_flag=TRUE, the server sends mail to each of the owners of the SysObjects associated with the policy.

## Related methods

Install, page 289
Validate, page 494

## Example

The following example uninstalls a lifecycle.

```
if (dmAPIExec("uninstall," session "," policy_id) == 0) {
print "Error uninstalling lifecycle policy '" policy_id "'"
print dmAPIGet ("getmessage," session)
}
```

# Unlink

## Purpose

Removes a link between an object and a folder.

## Syntax

```
dmAPIExec("unlink,session,object_id,location|alias")
```

## Arguments

**Table 2-165. Unlink method arguments**

| Argument | Description |
|---|---|
| *session* | Identifies an open repository session. |
| *object_id* | Specifies the object that you want to unlink. Use the object's ID or an indirect reference (*@object_id*) that points to the object. |
| *location* | Identifies the folder or cabinet from which you want to unlink the object. You can use either the object ID of the folder or cabinet or its folder path. A folder path has the format:<br>`/cabinet_name{/folder_name}`<br>If you include this argument, do not include *alias*. |
| *alias* | Identifies an alias in place of a location. The alias is resolved when the method is executed. This argument is intended for use in applications. The alias has the format:<br>`%[alias_set_name.]alias_name`<br>*alias_set_name* is the name of an alias set object.<br>*alias_name* is a value in the alias_name attribute of the alias set object.<br>For information about aliases and their use, refer to Appendix A, Aliases in *Content Server Fundamentals*. |

## Return value

The Unlink method returns TRUE if successful or FALSE if unsuccessful.

## Usage notes

Using the Unlink method has the following conditions:

- If you are unlinking the object from its primary storage location, you must have at least Write access to the object and have Change Location permission on the object.
- To unlink secondary links:
  - You must have at least Browse access to the object.
  - You must have Change Location permission on the object.
- If the repository is running under folder security, you must also have at least Write permission for the folder or cabinet from which you are unlinking the object.

Documents and folders must have at least one link to a folder or cabinet. Consequently, if you unlink an object from its only link, you must relink it to some folder or cabinet before saving or checking in the object.

The repeating attribute i_folder_id stores the object IDs of all folders and cabinets to which an object is linked. The object ID stored in the first position of this attribute (i_folder_id[0]) represents the object's primary storage location. If you unlink the object from the folder represented by i_folder_id[0] (or simply remove the value in this position), the system automatically shifts the remaining values up one position. The object ID previously in the second position (i_folder_id[1]) becomes the value in the first position (i_folder_id[0]) and, consequently, represents the new primary storage location of the object.

The server looks in the repository defined in the docbase_scope attribute of the session's session config object for the folder identified in the *location* argument. If the folder is not found in that repository, the method fails.

## Related methods

Link, page 297

## Example

This example:

- Fetches a specified folder object
- Unlinks the folder from its cabinet
- Links the folder to another cabinet

Saves the folder object

```
cmd_str = "fetch," session "," fldr
err_flag = dmAPIExec( cmd_str )
if (err_flag == 0) {
cmd_str = "getmessage," session
err_msg = dmAPIGet( cmd_str )
print "Error fetching folder '" fldr "'"
print err_msg
exit
}

cmd_str = "unlink," session "," fldr "," cab1
err_flag = dmAPIExec( cmd_str )
if (err_flag == 0) {
cmd_str = "getmessage," session
err_msg = dmAPIGet( cmd_str )
print "Error unlinking folder '" fldr "' from cabinet '" cab1 ","
print err_msg
exit
}

cmd_str = "link," session "," fldr "," cab2
err_flag = dmAPIExec( cmd_str )
if (err_flag == 0) {
cmd_str = "getmessage," session
err_msg = dmAPIGet( cmd_str )
print "Error linking folder '" fldr "' to cabinet '" cab2 ","
print err_msg
exit
}

cmd_str = "save," session "," fldr
err_flag = dmAPIExec( cmd_str )
if (err_flag == 0) {
cmd_str = "getmessage," session
err_msg = dmAPIGet( cmd_str )
print "Error saving folder '" fldr "'"
print err_msg
exit
}
```

# Unlock

## Purpose

Removes an intention lock without saving any changes that may have been made to the locked object.

## Syntax

```
dmAPIExec("unlock,session,object_id[,send_mail]
[,compound_arch_value][,special_app_value]")
```

## Arguments

**Table 2-166. Unlock method arguments**

| Argument | Description |
|---|---|
| *session* | Identifies an open repository session. |
| *object_id* | Specifies the object that you want to unlock. Use the object's ID or an indirect reference (*@object_id*) that points to the object. |
| *send_mail* | Directs the server to send electronic mail to the lock's owner, notifying that user that the lock is being removed. The default for this flag is T (TRUE). If you do not want the lock owner notified, set this flag to F (FALSE). |
| *compound_arch _value* | Sets the a_compound_architecture attribute of the object. Refer to Application-specific arguments, page 478 in the Usage notes for an extended discussion of this argument. |
| *special_app_value* | Sets the a_special_app attribute of the object. Refer to Application-specific arguments, page 478 in the Usage notes for an extended discussion of this argument. |

## Return value

The Unlock method returns TRUE if successful or FALSE if unsuccessful.

## Usage notes

To use the Unlock method, you must be:

- The lock owner,
- The owner of the object that you are unlocking, or
- A user with Sysadmin or Superuser user privileges.

If you are the lock owner, the *send_mail* flag is ignored, regardless of its setting.

You do not have to checkout an object to unlock it, nor do you have to save the object after you unlock it.

Executing the Unlock method clears the object's r_lock_machine attribute. (When an object is locked, this attribute is set to the name of the client machine on which the user who locked the object was working.)

This method does not require a Save method call to take effect.

### Application-specific arguments

The *compound_arch_value* and *special_app_value* arguments set the object's a_compound_architecture and a_special_app attributes, respectively. These attributes are intended for internal and user-defined application use.

The a_compound_architecture attribute is used internally by WorkSpace's Virtual Document Manager (VDM) to determine if the object can be structurally changed in the VDM. If an object's a_compound_architecture attribute is set to any value other than an empty string, VDM assumes the object is read only and does not allow any structural changes in the object. A_compound_architecture is a string-valued attribute with a length of 16.

The a_special_app attribute is used by the server and WorkSpace to determine if the object is currently reserved for use by another application such as Lotus Notes or Interleaf. If this is any value other than an empty string, the system assumes that another application currently has control of the object. A_special_app is a string-valued attribute with a length of 32.

User-defined applications can use these attributes by setting them on check out and check in or when an object is unlocked (as in a Cancel Checkout operation). For an unlock operation, these attributes are generally set to the values they had before the object was checked out (locked).

## Related methods

Checkin, page 138
Unfreeze, page 469

## Example

This example:

- Checks out a specified document
- Unlocks the document

```
cmd_str = "checkout," session "," doc_id
err_flag = dmAPIExec(cmd_str)
if (err_flag == 0) {
cmd_str = "getmessage," session
err_msg = dmAPIGet( cmd_str )
print "Error checking out the document"
print err_msg
exit
}

cmd_str = "unlock," session "," obj_id
err_flag = dmAPIExec(cmd_str)
if (err_flag == 0) {
cmd_str = "getmessage," session
err_msg = dmAPIGet( cmd_str )
print "Error unlocking the document"
print err_msg
exit
}
```

# Unmark

## Purpose

Removes symbolic labels from an object.

## Syntax

```
dmAPIExec("unmark,session,object_id,label{,label}")
```

## Arguments

**Table 2-167. Unmark method arguments**

| Argument | Description |
|---|---|
| *session* | Identifies an open repository session. |
| *object_id* | Specifies the object whose label you want to remove. Use the object's ID or an indirect reference (*@object_id*) that points to the object. |
| *label* | Specifies the symbolic label that you want to remove from the object. You can specify more than one label. You cannot specify the symbolic label CURRENT. |

## Return value

The Unmark method returns TRUE if successful or FALSE if unsuccessful.

## Usage notes

You must check out or fetch an object before you can use Unmark to remove any of its symbolic labels. After you complete the Unmark operation, you must check in or save the object to save the changes made by the Unmark method.

You cannot use Unmark to remove implicit version labels or to remove the CURRENT label from an object version.

## Related methods

Mark, page 310

## Example

The following example:

- Checks out a document
- Uses the Unmark method to remove one of the document's symbolic labels
- Saves the document

```
doc_id = dmAPIGet("checkout," session "," obj_id)
dmAPIExec("unmark,"session","obj_id",Release1")
dmAPIExec("save," session "," obj_id)
```

# Unprint

## Purpose

Removes a job from a print queue.

## Syntax

```
dmAPIExec("unprint,session[,printer_name],job_id")
```

## Arguments

**Table 2-168. Unprint method arguments**

| Argument | Description |
|---|---|
| *session* | Identifies an open repository session. |
| *printer_name* | Identifies the printer associated with the print queue. Use the object name of the output device object that represents the printer. If you do not specify the printer, the default is the system printer. |
| *job_id* | Specifies which print job you want to remove. (Refer to Usage notes for information about obtaining the print job number.) |

## Return value

The Unprint method returns TRUE if successful or FALSE if unsuccessful.

## Usage notes

The *job_id* is the value returned by the Print method if you used that method to place the job on the print queue. You can also obtain the *job_id* by executing the Lpq method—the job_id is included in Lpq's output.

## Related methods

Lpq, page 308
Print, page 321

## Example

```
job_id = dmAPIExec(sprintf("unprint,%s,engr_prntr,%s",session, job_id));
```

# Unregister

## Purpose

Removes the current user's registration for a specified event.

## Syntax

```
dmAPIExec("unregister,session,object_id,event[,user]")
```

## Arguments

**Table 2-169. Unregister method arguments**

| Argument | Description |
|---|---|
| *session* | Identifies an open repository session. |
| *object_id* | Identifies the object or object type that is the target of the event.<br><br>To identify a particular object, use the object's object ID or an indirect reference (*@object_id*).<br><br>To identify an object type, use the object ID of the type's dm_type object. The object type must be SysObject or a SysObject subtype.<br><br>**Note:** Registering for notification on an object type requires Sysadmin or Superuser privileges. Consequently, if you had the privileges to register for notification on an object type, you are allowed to unregister for notification on that type even if you no longer have Sysadmin or Superuser privileges.<br><br>Appendix B, System Events, lists the valid object types for system-defined events. |
| *event* | Identifies the event whose registration you want to cancel.<br><br>Appendix B, System Events , lists system-defined events and the target object types for each event. |
| *user* | User who initially registered the event. Used by a user with Sysadmin or Superuser user privileges to unregister an event previously registered by another user. |

## Return value

The Unregister method returns TRUE if successful or FALSE if unsuccessful.

## Usage notes

Event notification registrations are defined for a specific event (or all events) on a specific object or object type. For example, you can register to receive notification each time a specific document is checked out or a lifecycle is validated. Consequently, cancelling an event registration requires you to specify both the event and the object or object type that is the target of the event. If you want to remove the registration for the document's checkout, you must issue the Unregister method specifying the document as the *object_id* and dm_checkout as the *event*. ( Appendix B, System Events, lists the system-defined events and the object types for which each can be registered.)

If you have registered for several events against the same object or object type, you must remove each with a separate Unregister method call. For example, if you have registered to receive notification each time a particular lifecycle is validated, installed, or uninstalled, you must remove these registrations using separate Unregister method calls, one for each event.

If you are not registered for the specified event, the method fails with an error.

Only a user with Sysadmin or Superuser privileges can unregister events owned by another user.

## Related methods

Audit, page 115
Register, page 346
Unaudit, page 466

## Example

This example removes the current user's registration to receive notification whenever the specified document is checked out:

```
dmAPIExec(sprintf("unregister,%s,%s,dm_checkout", session,
doc_id));
```

# Updatepart

## Purpose

Modifies an existing component of a virtual document.

## Syntax

```
dmAPIExec("updatepart,session,virtdoc_id,containment_id
[,version_label][,order_no][,use_node_ver_label]
[,follow_assembly][,copy_child]
[,containment_type,containment_desc]")
```

## Arguments

**Table 2-170. Updatepart method arguments**

| Argument | Description |
|---|---|
| *session* | Identifies an open repository session. |
| *virtdoc_id* | Identifies the virtual document that contains the component you are modifying. Use the object's ID or an indirect reference (*@object_id*) that points to the object. |
| *containment_id* | Identifies the component you want to modify. Use the r_object_id value of the component's associated containment object. |
| *version_label* | Indicates which version of the component to include in the virtual document. You can specify either an implicit or symbolic version label. |
| *order_no* | Defines the component's position within the virtual document.<br><br>Use this argument only when the calling application bypasses the automatic ordering management provided by the Server and manages the component order within the virtual document itself.<br><br>Order_no requires an integer value. |

| Argument | Description |
|---|---|
| *use_node_ver_label* | Determines how the server chooses late-bound descendants of a component. This argument is useful only if the component is a virtual document.<br><br>T (TRUE) directs the server to use the component's early-bound symbolic label to resolve late-bound descendants. F (FALSE) directs the server to resolve the component's descendants using the late-bound version label specified in the IN DOCUMENT clause.<br><br>Refer to the Usage notes for an expanded explanation. |
| *follow_assembly* | Determines whether the server uses the containment objects or the component's assembly to find the component's descendants. This argument is only useful if the component is a virtual document.<br><br>T (TRUE) directs the server to use the component's assembly (if it has an assembly). F (FALSE) directs the server to use the containment objects. |
| *copy_child* | Determines whether the Documentum client copies or references the component when a user copies the containing virtual document. This is an integer attribute. Valid values are:<br>• 0, which lets users decide at the time the copy operation is requested<br>• 1, which directs the client to reference the component<br>• 2, which directs the client to copy the component<br><br>**Note:** The reference is an internal pointer. Referencing the component does not create a dm_reference object.<br><br>This argument is only useful if the component is itself a virtual document. |

| Argument | Description |
|---|---|
| *containment_type* | Used only if the containing virtual document (identified in *document_id*) is an XML document. This argument identifies what type of reference to the component is found in the containing document.<br><br>If you include this, you must also include *containment_desc*.<br><br>This argument is used primarily by the Documentum clients. Refer to XML support, page 490 for details. |
| *containment_desc* | Used only if the containing virtual document (identified in *document_id*) is an XML document. This argument identifies the text of reference to the component in the containing document.<br><br>If you include this, you must also include *containment_type*.<br><br>This argument is used primarily by the Documentum clients. Refer to XML support, page 490 for details. |

## Return value

The Updatepart method returns TRUE if successful or FALSE if unsuccessful.

## Usage notes

The Updatepart method modifies how a component behaves within the virtual document. For example, you can use Updatepart to change a component's binding, so that a different version is used in the virtual document. Or, you can use it to define which of the component's descendants to include in the virtual document. If you are managing the order numbers in the virtual document, you can also use Updatepart to modify a component's position within the document.You cannot use Updatepart to modify the content or attributes of a component.

To use Updatepart, you must have either Version or Write permission on the containing virtual document (the object specified in the *virtdoc_id* argument). Version permission is required if you want to version the object. Write permission is required if you want to save the object.

Changes you make are not visible until you check in or save the containing virtual document.

### Identifying the component

Identify the component you want to modify by using its containment ID in the *containment_id* argument. A component's containment ID is the object ID of the containment object that links the component to the virtual document. You can obtain this object ID with the following query:

```
SELECT "r_object_id", CONTAIN_ID FROM "dm_sysobject"
IN DOCUMENT ID('virtual_doc_id')
```

The query returns a list of the object IDs and containment IDs for all the direct components of the virtual document.

### Modifying assembly behavior

To modify assembly behavior, use the *use_node_ver_label* and *follow_assembly* arguments. These arguments let you define how the server chooses a component's descendants when the virtual document that contains the component is assembled. You can set either or both of these arguments.

#### Use_node_ver_label

The *use_node_ver_label* argument sets the use_node_ver_label attribute in the component's containment object.

If use_node_ver_label is TRUE for the component and the component is early bound to the virtual document, the server resolves any late-bound descendants of the component using the early-binding label specified for the component. It ignores any binding condition specified at the time of assembly. If an early-bound descendant is found that has use_node_ver_label set to TRUE, then that descendant's label is used to resolve descendants from that point.

If use_node_ver_label is FALSE or if the component is late bound to the virtual document, the server resolves the component's late-bound descendants using the binding condition specified at the time of assembly.

(use_node_ver_label, page 167 of *Content Server Fundamentals* contains an expanded explanation of how use_node_ver_label affects the assembly of virtual documents.)

#### Follow_assembly

The *follow_assembly* argument sets the follow_assembly attribute in the component's containment object. If the component has an associated assembly and follow_assembly is set to TRUE, the server will use the assembly, rather than the containment objects, to determine the component's descendants. In these cases, any binding conditions for the descendants are ignored. The descendants are taken from the assembly.

If follow_assembly is FALSE or if the component has no assembly, the server uses the containment objects and binding specifications to determine the component's descendants.

### Modifying copy behavior

Setting the *copy_child* argument sets the copy_child attribute in the component's containment object. This attribute controls how the Documentum client handles the component when users copy the containing virtual document. The client can either copy or reference the component.

Copy_child is an integer attribute with three valid values:

- 0, which means that users make the decision to copy or reference the component when they request the operation
- 1, which directs the client to reference the component in the new copy
- 2, which directs the client to copy the component for the new copy

The default value is 0.

**Note:** The reference is an internal pointer. It is not a dm_reference object.

### XML support

The *containment_type* and *containment_desc* arguments are used internally by the DFC to correctly set the references in the containing document when the component is updated. The arguments set the a_contain_type and a_contain_desc attributes in the containment object.

Although you can use these arguments to update the attributes to values you choose, if you do, the behavior of the XML document when manipulated using DesktopClient or the DFC is undefined.

For more information about the values set by these arguments and their use, refer to XML support, page 165 of *Content Server Fundamentals*.

## Related methods

Appendpart, page 84
Insertpart, page 280
Removepart, page 364

## Example

```
result = dmAPIExec("updatepart," session "," virtual_doc_id ",
"containment_id ",CURRENT,,T,F")
if (result == 0)
{
print "Error updating the component's containment object"
print dmAPIGet("getmessage," session)
exit
}
```

# Useacl

## Purpose

Assigns an ACL to an object.

## Syntax

```
dmAPIExec("useacl,session,object_id,acl_specification")
```

## Arguments

**Table 2-171. Useacl method arguments**

| Argument | Description |
|---|---|
| *session* | Identifies an open repository session. |
| *object_id* | Identifies the object to which you are attaching the ACL. Use the object's ID or an indirect reference (*@object_id*) that points to the object. |
| *acl_specification* | Specifies which ACL you want to use for the specified object. The ACL you specify must be in the same repository as the object identified in *object_id*.<br><br>Use one of the following keywords to identify the ACL:<br><br>FOLDER, to identify the ACL associated with the object's primary folder<br><br>TYPE, to identify the ACL associated with the object's type definition<br><br>USER, to identify the ACL associated with the dm_user object representing the creator of the object<br><br>NONE, meaning no default ACL<br><br>The keywords are not case sensitive. |

## Return value

The Useacl method returns TRUE if successful or FALSE if unsuccessful.

## Usage notes

When a repository is running in ACL mode, every SysObject in the repository has an associated ACL. When you create a document or other SysObject, there are a variety of ways to designate which ACL to use with the new object. The Useacl method is one of those ways. (For an extended discussion of creating and assigning ACLs, refer to Managing ACLs, page 387 of the *Content Server Administrator's Guide*.)

The Useacl method lets you select which default ACL you want to associate with the object. The default ACLs for a particular object are the ACLs associated with the object's primary folder, its object type definition, and the dm_user object representing its creator. The primary folder is the folder in which the object is stored when it is created. If you want the object to use the same ACL as the folder in which you are storing the object, specify the keyword FOLDER in the Useacl method:

```
API>create,s0,dm_document
...
<doc_id>
API>useacl,s0,doc_id,FOLDER
API>save,s0,doc_id
```

**Note:** If the object was placed directly in a cabinet, then its primary folder is the cabinet and the cabinet's ACL is used.

Each type definition has an associated ACL. This ACL does not govern access to the type but instead serves as a potential default ACL for objects of that type. If you want to use the type's default with your new object, specify TYPE in the Useacl method:

```
API>create,s0,dm_document
...
doc_id
API>useacl,s0,doc_id,TYPE
API>save,s0,doc_id
```

The final option is the default associated with you, the user who created the object. Each user is represented by an object of type dm_user. Each user object has an associated ACL. Like the ACL associated with types, the user's ACL does not govern access to the user but is instead a potential default ACL for all objects created by the user. If you want to use your ACL as the object's default ACL, specify USER in the Useacl method:

```
API>create,s0,dm_document
...
doc_id
API>useacl,s0,doc_id,USER
API>save,s0,doc_id
```

For a new object, the server always uses one of the default ACLs as the basis for the object's ACL unless you specifically assign another ACL to the object. If you create an object, then grant or revoke permissions, and then save the object, the server copies the default ACL, makes the changes to the default, and assigns the new copy to the object. If you want a new ACL for the object that contains only the permissions you assign and does not use a default ACL as its basis, issue the Useacl method with NONE option.

If you do not choose which default to use by explicitly issuing a Useacl method, the server automatically uses the default ACL defined in the server's configuration. (Refer to The default ACLs, page 400 in the *Content Server Administrator's Guide* for information about this.)

## Related methods

Grant, page 258
Revoke, page 400

## Example

```
err = dmAPIExec ("fetch," session "," obj_id)
if (err == 0)
{
print "Error fetching the object '" obj_id "'"
print dmAPIGet ("getmessage," session)
exit
}

err = dmAPIExec ("useacl," session "," obj_id ", FOLDER")
if (err == 0)
{
print "Error using the ACL from FOLDER"
print dmAPIGet ("getmessage," session)
exit
}

err = dmAPIExec ("save," session "," obj_id)
if (err == 0)
{
print "Error saving the object '" obj_id "'"
print dmAPIGet ("getmessage," session)
exit
}
```

**Note:** obj_id is a SysObject id.

# Validate

## Purpose

Validates a lifecycle, workflow, or activity definition.

## Syntax

```
dmAPIExec("validate,session,object_id[,check_activity]")
```

## Arguments

**Table 2-172. Validate method arguments**

| Argument | Description |
|---|---|
| *session* | Identifies an open repository session. |
| *object_id* | Identifies the lifecycle, activity, or process definition that you want to validate. Use the object ID. |
| *check_activity* | Indicates whether to validate the activities referenced by the process definition. Use this only if object_id refers to a process definition. The default value is F (FALSE). |

## Return value

The Validate method returns TRUE if the object is valid or FALSE if not.

## Usage notes

The Validate method validates a lifecycle (a dm_policy object), process definition, or activity definition object.

The Validate method requires at least Relate permission or Sysadmin or Superuser privileges on the target of the validation. For workflow definitions, Validate requires Relate permission on the activities referenced by the definition. If the method succeeds, it sets the r_definition_state of the object to Validated.

You cannot validate an object that is already validated or installed.

For activities, the Validate method verifies that all package definitions conform to the allowable forms.

To validate a workflow definition, the method:

- Verifies that of all the referenced activities (by activity_def_id) exist in the installed or validated state. Also, verifies that none of the referenced activities are foreign objects.
- Verifies that all the referenced (by link_src_activity and link_dest_activity) activities exist.
- Makes sure that the specified port names (in link_src_port and link_dest_port) exist in the associated activity object.
- Makes sure that the source port and the destination port are connectable. Two ports are connectable if the following criteria are met:
  - The source port specifies an output port, whereas the destination port specifies an input port.
  - The two ports are compatible.
- Validates that each input and output port takes only one link.
- Verifies activities as follows:
  - There is at least one start activity
  - There is only one end activity
  - There are no outgoing links from the end activity
- Validates the referenced activity definitions, if *check_activity* is set to TRUE.

Validating a lifecycle consists of checking a subset of the lifecycle's attributes for valid values:

- The r_definition_state of the policy must be draft.
- The lifecycle's definition includes at least one attachable state.
- The primary type is specified in the included_type attribute.
- The subtypes specified in the include_subtypes exist and are all subtypes of the primary type specified in the included_type attribute.
- The dm_aggr_domain, dm_expression, and dm_procedure objects referred to in the lifecycle exist.

## Related methods

Install, page 289
Invalidate, page 291
Save, page 405
Uninstall, page 472

## Example

The following example validates a lifecycle.

```
if (dmAPIExec("validate," session "," policy_id) == 0) {
```

```
print "Error validatinglifecycle policy '" policy_id "'"
print dmAPIGet ("getmessage," session)
}
```

# Values

## Purpose

Returns the number of values in an attribute.

## Syntax

```
dmAPIGet("values,session,object_id,attribute")
```

## Arguments

**Table 2-173. Values method arguments**

| Argument | Description |
|---|---|
| *session* | Identifies an open repository session. |
| *object_id* | Identifies the object that contains the attributes whose values you are counting. Use the object's ID or an indirect reference (*@object_id*) that points to the object. |
| *attribute* | Specifies the attribute whose values you want to count. You can specify any attribute of the object's type. If the object is a virtual document, you can also specify either of the following computed attributes:<br>• _componentID<br>• _containID |

## Return value

The Values method returns a number that indicates how many values are defined for an attribute.

Specifying the _componentID attribute returns a count of the number of components in the virtual document.

Specifying the _containID attribute returns a count of the number of virtual documents that contain the object.

## Usage notes

This method is most useful for repeating attributes. Single-valued attributes always return 1.

If you use this method to return the number of values in the special attributes, _accessor_name and _accessor_permit, you must first use the Get method to obtain the value of _accessor_name[0]. This is because the values for these attributes are not computed until _accessor_name[0] is fetched.

## Related methods

Count, page 164
Locate, page 304

## Example

This example:

- Fetches a specified object
- Determines the number of its attributes
- Loops though each attribute and gets its name, datatype, and value
- Prints the name and value of each attribute (If the attribute is a repeating attribute, all of its values are printed.)

```
cmd_str = "fetch," session "," obj_id
err_flag = dmAPIExec(cmd_str)
if (err_flag == 0) {
cmd_str = "getmessage," session
err_msg = dmAPIGet( cmd_str )
print "Error fetching the object"
print err_msg
exit
}

cmd_str = "count," session "," doc_id
attr_cnt = dmAPIGet(cmd_str)
if (attr_cnt == NULL) {
cmd_str = "getmessage," session
err_msg = dmAPIGet( cmd_str )
print "Error retrieving the number of attributes"
print err_msg
exit
}

for (i=0; i<attr_cnt; i++) {
cmd_str = "get," session "," obj_id ",_names[" i "]"
attr_name = dmAPIGet(cmd_str)
if (attr_name == NULL) {
cmd_str = "getmessage," session
err_msg = dmAPIGet( cmd_str )
```

```
print "Error retrieving the attribute name : " i
print err_msg
exit
}

cmd_str = "datatype," session "," obj_id "," attr_name
d_type = dmAPIGet(cmd_str)
if (d_type == NULL) {
cmd_str = "getmessage," session
err_msg = dmAPIGet( cmd_str )
print "Error retrieving datatype for attribute '" attr_name "'"
print err_msg
exit
}

if (d_type == 0) d_type_str = "boolean"
if (d_type == 1) d_type_str = "integer"
if (d_type == 2) d_type_str = "string"
if (d_type == 3) d_type_str = "ID"
if (d_type == 4) d_type_str = "time"
if (d_type == 5) d_type_str = "float"
#
cmd_str = "repeating," session "," obj_id "," attr_name
r_flag = dmAPIGet(cmd_str)
if (r_flag == NULL) {
cmd_str = "getmessage," session
err_msg = dmAPIGet( cmd_str )
if (err_msg != NULL) {
print
"Error retrieving repeating flag for attribute '
" attr_name "'"
print err_msg
exit
}
}

if (r_flag == 0) {
cmd_str = "get," session "," obj_id "," attr_name
value = dmAPIGet(cmd_str)
if (value == NULL) value = "NULL"
out_str=sprintf("%2s)%19s : %s", i, attr_name,
 d_type_str, value)
}
else {
cmd_str = "values," session "," obj_id
cnt = dmAPIGet(cmd_str)
if (cnt == NULL) {
cmd_str = "getmessage," session
err_msg = dmAPIGet( cmd_str )
print
"Error retrieving the number of values
 for attribute '" attr_name "'"
print err_msg
exit
}
out_str=sprintf("%2s)%19s:(%s value(s))",i,
attr_name, d_type_str, cnt)
}
print out_str
```

```
if (r_flag != 0 && cnt > 0) {
for (j=0; jcnt; j++) {
cmd_str = "get," session "," obj_id "," attr_name
"[" j "]"
value = dmAPIGet(cmd_str)
if (value == NULL) value = "NULL"
out_str = sprintf("%29s : %s", j, value)
print out_str
}
}
}
```

# Vdmpath

## Purpose

Returns the path or paths through a virtual document that lead to a particular component.

## Syntax

```
dmAPIGet("vdmpath,session,component_id[,root_id]
[,shortest_path]{,version_list}
```

## Arguments

**Table 2-174. Vdmpath method arguments**

| Argument | Description |
|---|---|
| *session* | Identifies an open repository session. |
| *component_id* | Identifies the component you want to locate by path. Use the object's ID or an indirect reference (*@object_id*) that points to the object. |
| *root_id* | Identifies a virtual document that contains the component. Use the document's object ID.<br><br>If you include this argument, the method returns only paths that start with the specified virtual document.<br><br>If unspecified, the method returns all paths to component. |
| *shortest_path* | A flag that directs the method to return only the shortest path to the component if set to TRUE. This argument is FALSE by default. |
| *version_list* | Defines one or more version labels to be used to select a version when a late-bound component is encountered in a path. You can specify implicit or symbolic version labels.<br><br>If you include multiple version labels, separate them with commas and enclose the list in single quotes.<br><br>Refer to the Usage notes for details. |

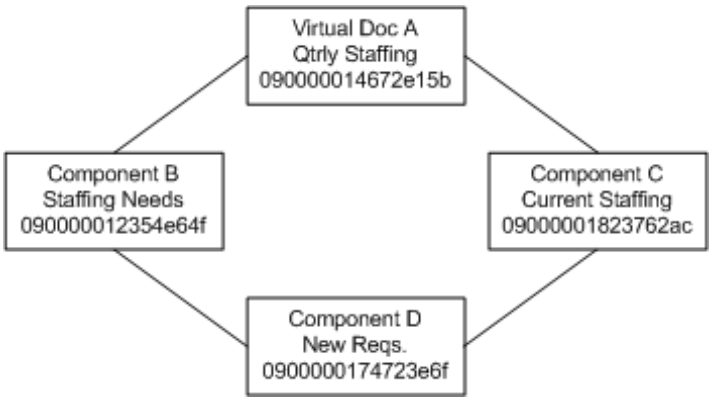
## Return value

The Vdmpath method returns a collection in which each result object represents one path to the component. The result objects have four repeating attributes: r_object_id, i_chronicle_id, object_name, and title. The values at corresponding index positions across the attributes represent one node in the path to the component.

For example, suppose you want to determine the paths to Component D in the virtual document shown in Figure 2–1, page 502. You issue the following Vdmpath method:

```
dmAPIGet("vdmpath,c,0900000174723e6f,090000014672e15b")
```

**Figure 2-1. Virtual document for Vdmpath return value example**




The method returns a collection that contains two result objects, each representing one path to Component D. Table 2–175, page 502 shows the attribute values for each of the result objects.

**Table 2-175. Example of attribute values in result objects returned by Vdmpath**

| Index position | Attributes of the result objects | | | |
|---|---|---|---|---|
| | r_object_id | i_chroni-cle_id | object_name | title |
| First Result Object Path ABD | | | | |
| [0] | 090000014672e15b | Object ID of the first version of Virtual Doc A | Virtual Doc A | Qtrly Hiring |

| Index position | Attributes of the result objects | | | |
|---|---|---|---|---|
| | r_object_id | i_chroni-cle_id | object_name | title |
| [1] | 090000012354e64f | Object ID of the first version of Compo-nent B | Component B | Staffing Needs |
| [2] | 0900000174723e6f | Object ID of the first version of Compo-nent D | Component D | New Reqs. |
| Second Result Object Path ACD | | | | |
| [0] | 090000014672e15b | Object ID of the first version of Virtual Doc A | Virtual Doc A | Qtrly Hiring |
| [1] | 09000001823762ac | Object ID of the first version of Compo-nent C | Component C | Current Staffing |
| [2] | 0900000174723e6f | Object ID of the first version of Compo-nent D | Component D | New Reqs. |

Each result object represents one path to the component. In the example, the first result object describes the path ABD. The second result object describes the path ACD. The values at each index position in the attributes (represented by a row in the table) describe an individual node in the path represented by the result object.

## Usage notes

Use this method instead of Vdmpathdql if performance is more important than flexibility in choosing a path.

The Vdmpath method begins with the component you identify and searches up the hierarchy of virtual documents containing the component by recursively querying dmr_containment objects. (A dmr_containment object contains information about parent nodes.) First, the method finds all documents that directly contain the component. Then, it finds all documents that contain those immediate parents of the original component. It continues the search in this manner until the top-level containing document or documents are found. The resulting paths are returned in a collection. (Refer to the previous section, Return value, page 502, for details about the return value.)

### Permissions and returned paths

The Vdmpath method searches only documents for which the World permission is at least Browse. If the search encounters a document for which the World permission is None, the search on that path stops. The returned path in such a case begins with the last document in the path for which World has at least Browse permission. For example, suppose the path to Component E is A>B>C>D>E and that World has no permission to Component B (that is, World's permission to B is None). When the method encounters B in the path to E, the search stops and the path is reported as C>D>E.

Paths that stop before the top-level containing document are not included in the results if the root _id argument is included in the method.

### Including shortest path

Setting the shortest_path flag to TRUE directs the method to return the shortest path found. If there are multiple paths that qualify as the shortest path, the method returns the first one it encounters.

In some instances, the shortest path may be a path that stops before reaching the top-level containing document. If you want the path to include a top-level document, include root_id in the method. The method will then return the shortest path through the document identified in root_id. Note, though, that this may not be the shortest possible path; it is only the shortest path that starts with the specified top-level document.

### Specifying versions

Some of the components found in a path may be late-bound to the containing virtual document. Late binding means that no particular version of the component is bound to the virtual document. Instead, the component's entire version tree is bound to the virtual document. Use the version_list argument to identify which version to choose if a late-bound component is encountered in a path.

You can use implicit or symbolic version labels to identify the version. If you include multiple version labels, separate them with commas and enclose them in single quotes. For example, the following method defines the labels approved and released for late-bound components:

```
dmAPIGet("vdmpath,s0,component_id,,,'approved,released'")
```

(Note the placeholder commas for the unused optional arguments.)

If two or more versions of a late-bound component match the specified version labels, Content Server chooses whichever version is returned first to it by the RDBMS.

If the late-bound component doesn't have any versions that match the specified version labels, the path search stops. If root_id is included in the method, the path is dropped from the results. If root_id is not included, the shortened path, up to the non-matching component, is returned

## Related methods

Vdmpathdql, page 506

## Examples

This example returns the paths to Component K (09000001236487ef). All late-bound components chosen for the paths will have either the "approved" or "released" version label.

```
dmAPIGet("vdmpath,c,09000001236487ef,,,'approved,released'")
```

The next example returns the paths to Component K found in the virtual document identified by 0900000153457b23. All late-bound components chosen for the paths will have either the "approved" or "released" version label.

```
dmAPIGet("vdmpath,c,09000001236487ef,0900000153457b23,,
'approved,released'")
```

The following example sets the shortest_path argument to TRUE and directs the server to select late-bound components that have the version label released:

```
dmAPIGet("vdmpath,c,09000001236487ef,,T,released'")
```

# Vdmpathdql

## Purpose

Returns the path or paths through a virtual document that lead to a particular component.

## Syntax

```
dmAPIGet("vdmpathdql,session,component_id[,root_id]
[,shortest_path][,type][,binding_condition]
[,nodesort_by]")
```

## Arguments

**Table 2-176. Vdmpathdql method arguments**

| Argument | Description |
|---|---|
| *session* | Identifies an open repository session. |
| *component_id* | Identifies the component you want to locate. Use the object's ID or an indirect reference (*@object_id*) that points to the object. |
| *root_id* | Identifies a virtual document that contains the component. Use the document's object ID.<br><br>If you include this argument, the method returns only paths that start with the specified virtual document.<br><br>If unspecified, the method returns all paths to the component. |
| *shortest_path* | A flag that directs the method to return only the shortest path to the component if set to TRUE. This parameter is FALSE by default. |
| *type* | Identifies the object type of the parent documents in the paths to the component. Use the object type's internal name. For example, dm_document. If not included, the default value is dm_sysobject.<br><br>You must include type if the binding condition references any attributes not defined for the dm_sysobject type. |

| Argument | Description |
|---|---|
| *binding_condition* | Defines a qualification used to select a version when a late-bound component is encountered in the path search. The qualification is that part of a DQL SELECT statement that appears after the WHERE clause. Refer to the Usage notes for syntax details particular to the Vdmpathdql method. |
| *nodesort_by* | Defines a selection criteria for returned versions if more than one version of a late-bound component meets the conditions imposed in binding_condition.<br><br>Include this only if binding_condition is included. Refer to the Usage notes for details. |

## Return value

The Vdmpathdql method returns a collection in which each result object represents one path to the component. The result objects have four repeating attributes: r_object_id, i_chronicle_id, object_name, and title. The values at corresponding index positions across the attributes represent one document in the path to the component. For an illustrated example, refer to the description of the return value for the Vdmpath method on Return value, page 502.

## Usage notes

Use Vdmpathdql when you want more flexibility in choosing late-bound components for the paths than offered by the Vdmpath method. Both methods, Vdmpathdql and Vdmpath, return the paths or paths to a given component by recursively searching dmr_containment objects. However, Vdmpathdql allows you to define a more complex condition for the selection of any late-bound components in the path.

### Permissions and return results

You must have at least Browse permission on the documents in returned paths. If the search encounters a document for which you don't have at least Browse permission, the search on that path stops. The returned path in such a case begins with the last document in the path for which you have Browse permission. For example, suppose the path to Component E is A>B>C>D>E and that you have no permission to Component B (that is, your permission to B is None). When the method encounters B in the path to E, the search stops and the returned path is reported as C>D>E.

### Including shortest path

Setting the shortest_path flag to TRUE directs the method to return the shortest path found. If there are multiple paths that qualify as the shortest path, the method returns the first one it encounters.

In some instances, the shortest path may be a path that stops before reaching the top-level containing document. If you want the path to include a top-level document, include root_id in the method. The method will then return the shortest path through the document identified in root_id. Note, though, that this may not be the shortest possible path; it is only the shortest path that starts with the specified top-level document.

### The type argument

Include the type argument if you include a binding condition that references attributes that are not defined for the SysObject object type. The type argument must identify the object type for which the attributes in the binding condition are defined.

### Including a binding condition

Some of the components found in a path may be late-bound to the containing virtual document. Late binding means that no particular version of the component is bound to the virtual document. Instead, the component's entire version tree is bound to the virtual document. Use the binding_condition argument to identify which version to choose if a late-bound component is encountered in a path.

The binding condition can be any valid WHERE clause qualification. (Refer to The WHERE clause, page 142 in the *Content Server DQL Reference Manual* for a description of a qualification.) Any attributes referenced in the binding condition must be fully qualified with the object type name for which the attributes are defined. If that object type is not dm_sysobject, you must identify the object type in the type argument.

Enclose the binding condition in single quotes.

**Note:** If the object type is not dm_sysobject, you must also include the type argument.

For example, suppose you want to find the paths to Component E and you want to include only released versions of late-bound components in the paths. Issue the following Vdmpathdql method:

```
dmAPIGet("vdmpathdql,s0,Component_E_id,,,,
'any dm_sysobject.r_version_label in
(''CURRENT'',''valid'')'")
```

If the search encounters a late-bound document, the server chooses either the version with the CURRENT or valid label for inclusion in the path.

If multiple versions satisfy the binding condition and the nodesort_by argument is not included in the method, the server chooses a version randomly. If nodesort_by is included, the server uses the instructions in that argument to select a version.

If the late-bound component has no versions that satisfy the binding condition, the path search stops. If root_id is included in the method, the path is dropped from the results.

If root_id is not included, the shortened path, up to the non-matching component, is returned.

### Including nodesort_by

Include nodesort_by if you believe that two or more versions of a late-bound node may qualify for inclusion in a path. The nodesort_by argument identifies which of the qualifying versions to select. The format of the argument is:

```
object_type.attr_name {,object_type.attr_name} [ASC|DESC]
```

The object type must be the object type identified in the type argument or dm_sysobject if the type argument is not included.

The qualifying versions are sorted by the values in the specified attributes. The default sort order is ascending (ASC).

## Related methods

Vdmpath, page 501

## Example

The following example chooses path components that either:

- Carry the CURRENT or VALID symbolic label and have not expired, or
- Are flagged as always good

```
dmAPIGet("vdmpathdql,c,<doc_id>,,,webpage_type,
'(webpage_type.expiration_date>DATE(TODAY) AND
ANY webpage_type.r_version_label IN
(''CURRENT'', ''VALID'')) OR
webpage_type.forever_good_flag=TRUE',
webpage_type.r_creation_date DESC")
```

# Verifyaudit

## Purpose

Verifies that an audit trail object has not been modified.

## Syntax

```
dmAPIExec("verifyaudit,session,object_id")
```

## Arguments

**Table 2-177. Verifyaudit method arguments**

| Argument | Description |
|---|---|
| *session* | Identifies an open repository session. |
| *object_id* | Object ID of a dm_audittrail, dm_audittrail_acl, or dm_audittrail_group object. |

## Return value

Verifyaudit returns TRUE if the audit_signature attribute of an audit trail object has a valid value. If the signature is invalid or missing, the method returns FALSE. Anyone can use this method.

## Usage notes

Anyone can use this method. No special privileges are required.

The method decrypts the value in the audit_signature attribute and then computes the value of the _sign_data computed attribute for the audit trail entry. It then hashes the sign_data value, based on the audit_version value, and compares the hashed value to the audit_signature value of the audit trail entry. If the values match, the method returns T, indicating that the record has not been changed.

In a single repository that has multiple servers, the method fails if the servers are using different AEKs and a server attempts to verify a signature created by another server with a different AEK. To ensure that this does not occur, either all servers for a particular repository must use the same AEK or the application must check that the server issuing

the Verifyaudit method is the same server that generated the signature. The server that generates the signature is recorded in the host_name attribute of the audit trail entry.

## Related methods

Audit, page 115
Createaudit, page 169
Unaudit, page 466

## Example

The following example verifies that the signature on the audit trail object represented by the object ID 5f0000025c23ae1c is valid, indicating that the object has not been changed:

```
dmAPIExec("verifyaudit,s0,5f0000025c23ae1c")
```

# Verifyesignature

## Purpose

Verifies that an electronic signature added by Addesignature is valid.

## Syntax

```
dmAPIExec("verifyesignature,session,object_id")
```

## Arguments

**Table 2-178. Verifyesignature method arguments**

| Argument | Description |
|---|---|
| *session* | Identifies an open repository session. |
| *object_id* | Object ID of the object whose signature is to be verified. This must be a SysObject or SysObject subtype. |

## Return value

Verifyesignature returns TRUE if successful or FALSE if unsuccessful.

## Usage notes

You must have installed the server with a Trusted Content Services license to use this method.

Use Verifyesignature to verify that an electronic signature added to an object with the Addesignature method is valid. You cannot use Verifyesignature to verify an electronic signature created by an Adddigsignature method or a Signoff method.

You must have at least Read permission on the object identified in the *object_id* argument to execute this method.

The Verifyesignature method examines the audit trail entries to find all entries for the particular object with the event name dm_addesignature. The method computes the hash values of the source content and the signed content. It compares the computed hash values to the corresponding hash values stored in the most recent audit trail entry.

Verifyesignature also checks that the signatures on the object are numbered consecutively.

## Related methods

Addesignature, page 48
Verifyaudit, page 510

## Example

```
status=dmAPIExec("verifyesignature,S0,09000002ae36c123")
```

# Chapter 3

# Functions for Creating Plug-in Libraries

The API interface between the plug-in and client or server, which retrieves content through the plug-in, consists of C functions that Content Server or the DMCL expects in the plug-in library for external storage content retrieval.

The functions are:



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 per 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

## Purpose

Called when a session terminates.

## Syntax

```
void dm_close_all (long session)
```

## Arguments

**Table 3-1. dm_close_all arguments**

| Argument | Description |
|---|---|
| *session* | Identifies the terminated session. |

## Return value

None.

## Usage notes

When a session terminates, this function is called to let the plug-in library to cleanup any internal data structure(s) based on the specified session.

# dm_close_content

## Purpose

Enables the plug-in to perform internal cleanup.

## Syntax

```
BOOL dm_close_content (long handle)
```

## Arguments

**Table 3-2. dm_close_content arguments**

| Argument | Description |
|---|---|
| *handle* | Identifies the read request. |

## Return value

TRUE or FALSE

## Usage notes

To enable the plug-in to perform internal cleanup, this function is called after the read operation for the supplied handle is complete or is interrupted in the middle.

# dm_deinit_content

## Purpose

Performs global internal cleanup operations.

## Syntax

```
void dm_deinit_content(void)
```

## Arguments

None

## Return value

None

## Usage notes

Just before the server or client unloads the plug-in library, this function is called to let the plug-in perform any global internal cleanup operations.

# dm_init_content

## Purpose

Use once when the plug-in library is loaded to let the plug-in to initialize any internal data structures.

## Syntax

```
BOOL dm_init_content (long maxsession, int mode)
```

## Arguments

**Table 3-3. dm_init_content arguments**

| Argument | Description |
|---|---|
| *maxsession* | Contains the maximum number of concurrent sessions. |
| *mode* | Indicates whether the plug-in is being invoked by the client (dmcl) or the server. A value of 1 indicates client and 0 indicates server. |

## Return value

The dm_init_content function returns a positive value for successful initialization. A negative return value forces the server or client to unload the plug-in library. This function should return a positive value when called multiple times within the same address space.

## Usage notes

This is the first function called by the server or client (depending on the value of the a_exec_mode attribute) once the plug-in library is loaded.

# dm_open_content

## Purpose

This function is called when the server or client needs to retrieve the content for the token.

## Syntax

```
BOOL dm_open_content ( long session, char *other_args,
char *token, char *store_object_id, void *callback,
long *handle, long errcode)
```

## Arguments

**Table 3-4. dm_open_content arguments**

| Argument | Description |
|---|---|
| *session* | Indicates the session that needs to retrieve the content. |
| *other_args* | Indicates the *other_args* supplied when executing a Mount method. NULL for the dm_extern_url, dm_extern_free storage types and when the 'token' specified in a Setpath operation is an absolute path. |
| *token* | Is the path-translated token for which to retrieve the content. |
| *store_object_id* | Indicates the external storage object ID. |
| *callback* | 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. |
| *handle* | Identifies the read request. Filled in on initiaization and passed for subsequent read operations. |
| *errcode* | Contains error code in case of failure. |

## Return value

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.

## Usage notes

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.

### Call back function

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

## Purpose

Enables backward compatibility for enhancement in future releases.

## Syntax

```
void dm_plugin_version(unsigned int *major, unsigned int *minor)
```

## Arguments

**Table 3-5. dm_plugin_verson arguments**

| Argument | Description |
|---|---|
| *major* | Set to 1. |
| *minor* | Set to 0. |

## Return value

None

## Usage notes

This 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 (by the dmcl or by the server). For the 4.0 server release, the plug-in protocol version is 1.0. Therefore, the plug-in must set 'major' to 1 and 'minor' to 0.

# dm_read_content

## Purpose

Requires the plug-in to return the content data into the location pointed to by buffer supplied.

## Syntax

```
long dm_read_content ( long handle, char *buffer,
long buffer_size, long *more_data, long *errcode)
```

## Arguments

**Table 3-6. dm_read_content arguments**

| Argument | Description |
|---|---|
| *handle* | Identifies the read request. |
| *buffer* | 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. |
| *buffer_size* | Contains the buffer size (16k). |
| *more_data* | When positive, indicates more reading is required. |
| *errcode* | Contains error code in case of failure. |

## Return value

Returns the number of bytes read and filled into buffer.

## Usage notes

The plug-in must maintain its own internal book-keeping to start reading from the next byte after the previous read.

# Appendix A

# API Method Quick Reference

This appendix contains only the formal syntax of the API server methods. For a full description of each method and its use, refer to Chapter 2, API Server Methods .

**Table A-1. API method quick reference**

| Method | *Syntax* |
|---|---|
| Abort, page 40 | dmAPIExec("abort,*session*[,*workflow_id*]") |
| Acquire, page 42 | dmAPIExec("acquire,*session*,*workitem_id*") |
| Addactivity, page 44 | dmAPIExec("addactivity,*session*,*process_id*, *activity_identifier*,*activity_definition_id* [,*activity_type*[,*activity_priority*]]") |
| Adddigsignature, page 46 | dmAPIGet("adddigsignature,*session*,*object_id*[,*user_name*][,*reason*]") |
| Addesignature, page 48 | dmAPIGet("addesignature,*session*,*object_id*[,*user_auth_name*],*password*[,*justification*][,*format_to_sign*][,*hash_algorithm*][,*presignature_hash*], *sig_method_name*[,*app_properties*][,*passthrough_arg1*][,*passthrough_arg2*]") |
| Addlink, page 52 | dmAPIExec ("addlink,*session*,*process_id*,*link_identifier*,*source_activity*,*source_port*,*destination_activity*,*destination_port*") |
| Addnote, page 54 | dmAPIExec("addnote,*session*,*annotation_id*,*object_id*[,*keep_permanent*]") |

| Method | *Syntax* |
|---|---|
| Addpackage, page 56 | dmAPIGet("addpackage,*session*,*object_id* ,*start_activity_name*,*input_port_name*,*package_name* ,*package_type*[,*note_id*[,*keep_permanent*]]{,*component_ID*}{,*component_name*}")<br>dmAPIGet("addpackage,*session*,*object_id*,*package_name* ,*package_type*{,*component_ID*}{,*component_name*}") |
| Addpackageinfo, page 60 | dmAPIExec("addpackageinfo,*session*,*activity_id*,*port_name* ,*package_name*,*package_type*[,*package_id*[,*package_label*]] [,*package_operation*][,*r_package_flag*]") |
| Addport, page 64 | dmAPIExec("addport,*session*,*activity_id*,*port_name*,*port_type*") |
| Addrendition, page 66 | dmAPIExec("addrendition,*session*,*object_id*,*file_name*,*format* [,*page_number*][,*storage_name*][,*atomic*][,*keep_flag*] [,*page_modifier*][,*batch_flag*][,*other_file*]") |
| Addroutecase, page 72 | dmAPIExec("addroutecase,*session*,*activity_id*, *route_case_identifier*,*routing_condition*{,*output_port*}")<br>dmAPIExec("addroutecase,*session*,*activity_id*, *route_case_identifier*[,,{*output_port*}]")<br>dmAPIExec("addroutecase,*session*,*activity_id*,EXCEPTIONAL{,*output_port*}") |
| Anyevents, page 75 | dmAPIExec("anyevents,*session*") |
| Append, page 77 | dmAPISet("append,*session*,*object_id*,*attribute*[,*pattern*]", *"value"*) |
| Appendcontent, page 80 | dmAPISet("appendcontent,*session*,*object_id*[,*length*][,*set_resource*]", *"content"*) |
| Appendfile, page 82 | dmAPIExec("appendfile,*session*,*object_id*,*file_name*[,*other_file*]") |
| Appendpart, page 84 | dmAPIGet("appendpart,*session*,*document_id*,*component_id* [,*version_label*][,*use_node_ver_label*][,*follow_assembly*] [,*copy_child*][,*containment_type*,*containment_desc*]") |
| Appendstate, page 90 | dmAPIGet("appendstate,*session*,*object_id"*) |

| Method | *Syntax* |
|---|---|
| Apply, page 92 | dmAPIGet("apply,*session*,*object_id*,*function_name* {,*argument*,datatype,*value*}") |
| Archive, page 98 | dmAPIGet("archive,*session*,*predicate*[,*operator*][,*priority*] [,*send_mail*][,*due_date*]") |
| Assemble, page 100 | dmAPIExec("assemble,*session*,*document_id*[,*virtual_doc_id*] [,*interrupt_freq*][,*qualification*][,*nodesort_list*]") |
| Assume, page 106 | dmAPIExec("assume,*session*,*user_auth_name*, *password*\|*login_ticket* [,*domain*][,*user_arg*]") |
| Attach, page 111 | dmAPIExec("attach,*session*,*object_id* [,default\|*policy_id*[,*position*\|*state_name*]] [,*alias_set_position*\|*alias_set_name*]") |
| Audit, page 115 | dmAPIExec("audit,*session*[,*object_id*],*event_name* [,*audit_subtypes*][,*controlling_app*] [,*policy_id*[,*state_name*]][,*sign_audit*][,*authentication*] [,*event_description*][,*attribute_list*][,*esignature_required*]") |
| Authenticate, page 123 | dmAPIExec("authenticate,*session* [,*repository_name*],*user_auth_name*, *password*\|*ticket* [,*domain*][,*user_arg*]") |
| Begintran, page 127 | dmAPIExec("begintran,*session*") |
| Bindfile, page 129 | dmAPIExec("bindfile,*session*,*object_id*,*page_number*,*src_id* ,*src_page_number*") |
| Branch, page 131 | dmAPIGet("branch,*session*,*object_id*[,*version_label*]") |
| Cachequery, page 133 | **Note:** This method, while still supported, is replaced by Query_cmd.<br>dmAPIGet("cachequery,*session*,*dql_query*") |
| Changepassword, page 135 | dmAPIExec("changepassword,*session*,*old_password*, *new_password*") |
| Checkin, page 138 | dmAPIGet("checkin,*session*,*object_id*[,*save_lock*] {,*version_label*}") |

| Method | *Syntax* |
|---|---|
| Checkinapp, page 141 | dmAPIGet("checkinapp,*session*,*object_id*[,*save_lock*] [,*version_label*][,*old_compound_arch_value*][,*old_special_app_value*][,*new_compound_arch_value*][,*new_special_app_value*]") |
| Checkout, page 146 | dmAPIGet("checkout,*session*,*object_id*[,*version_label*][,*compound_arch_value*][,*special_app_value*]") |
| Close, page 149 | dmAPIExec("close,*session*,*collection_identifier*") |
| Commit, page 151 | dmAPIExec("commit,*session*") |
| Complete, page 153 | dmAPIExec("complete,*session*,*workitem_id*[,[*return_value*],[*os_error*],[*result_id*][,*user_time*][,*user_cost*]") |
| Connect, page 155 | dmAPIGet("connect,*repository*,*user_auth_name*,*password*\|*ticket* [,*domain*][,*user_arg*][,*secure_mode*][,*application_token*][,*application_id*][,*force_authenticate*]") |
| Count, page 164 | dmAPIGet("count,*session*,*object_id*") |
| Create, page 167 | dmAPIGet("create,*session*,*object_type*") |
| Createaudit, page 169 | dmAPIGet("createaudit,*session*,*audited_obj_id*,*event_name*[,*string_1*][,*string_2*][,*string_3*][,*string_4*][,*string_5*][,*id_1*][,*id_2*][,*id_3*][,*id_4*][,*id_5*]") |
| Datatype, page 171 | dmAPIGet("datatype,*session*,*object_id*,*attribute*") |
| Delegate, page 174 | dmAPIExec("delegate,*session*,*workitem_id*,*user_name*") |
| Demote, page 176 | For unscheduled demotions:<br>dmAPIExec("demote,*session*,*sysobject_id* [,*to_state*\|*to_position*,*return_to_base*]")<br>For scheduled demotions:<br>dmAPIExec("demote,*session*,*sysobject_id*,*scheduled_date*,[*pattern*],*to_state*\|*to_position*\|cancel[,*return_to_base*]") |
| Dequeue, page 179 | dmAPIExec("dequeue,*session*,*stamp*") |
| Dereference, page 181 | dmAPIGet("dereference,*session*,*object_id*") |

| Method | *Syntax* |
|---|---|
| Describe, page 184 | `dmAPIGet("describe,session,type,object_type")`<br>`dmAPIGet("describe,session,table,tablename")` |
| Destroy, page 186 | `dmAPIExec("destroy,session,object_id[,force_flag]")` |
| Disassemble, page 190 | `dmAPIExec("disassemble,session,document_id")` |
| Disconnect, page 192 | `dmAPIExec("disconnect,session")` |
| Dump, page 194 | `dmAPIGet("dump,session,object_id\|type_identifier")` |
| Dumpconnection, page 196 | `dmAIGet("dumpconnection[,session]")` |
| Dumploginticket, page 198 | `dmAPIGet("dumploginticket,session,login_ticket\|aac_token")` |
| Encryptpass, page 201 | `dmAPIGet("encryptpass,session,password")` |
| Execquery, page 203 | `dmAPIExec("execquery,session,[readquery_flag],dql_query")` |
| Execsql, page 205 | `dmAPIExec("execsql,session,sql_statement")` |
| Execute, page 207 | `dmAPIExec("execute,session,workflow_id")` |
| Fetch, page 209 | `dmAPIExec("fetch,session,object_id[,type] [,persistent_cache][,consistency_check_value]")` |
| Flush, page 213 | For persistent client caches, session caches, and data dictionary caches<br>`dmAPIExec("flush,session,cache_keyword,[object_type][,reset_change_bit][,include_subtypes]")` |
| Flushcache, page 217 | For in-memory server and client object caches<br>`dmAPIExec("flushcache,session[,discard_changed]")` |
| Flushconnectpool, page 219 | `dmAPIExec("flushconnectpool")` |
| Freeze, page 221 | `dmAPIExec("freeze,session,object_id[,freeze_components]")` |
| Get, page 224 | `dmAPIGet("get,session,object_id\|type_identifier,attribute[[index]] [,pattern]")` |

| Method | *Syntax* |
|---|---|
| Getconnection, page 229 | dmAPIGet("getconnection,*session*,*scoping_value* [,*connect_flag*]") |
| Getcontent, page 231 | dmAPIGet("getcontent,*session*,*document_id*[,*format*] [,*page_num*][,*page_modifier*][,*get_resource*]") |
| Getdocbasemap, page 235 | dmAPIGet("getdocbasemap,*session*[,*protocol*,*host_name*,*port_number*]") |
| Getdocbrokermap, page 237 | dmAPIGet("getdocbrokermap,*session*") |
| Getevents, page 239 | dmAPIGet("getevents,*session*") |
| Getfile, page 241 | dmAPIGet("getfile,*session*,*object_id*[,*file_name*][,*format*] [,*page_number*][,*get_resource*][,*page_modifier*]") |
| Getlastcoll, page 245 | dmAPIGet("getlastcoll,*session*") |
| Getlogin, page 247 | dmAPIGet("getlogin,*session*[,*user*]") |
| Getmessage, page 251 | dmAPIGet("getmessage[,*session*][,*severity_level*]") |
| Getpath, page 253 | dmAPIGet("getpath,*session*,*object_id*[,*page*][,*format*] [,*page_modifier*][,*get_resource*]") |
| Getservermap, page 256 | dmAPIGet("getservermap,*session*\|apisession],*repository* [,*protocol*][,*host_name*][,*port_number*]") |
| Grant, page 258 | dmAPIExec("grant,*session*,*object_id*[,*accessor_name*\|*alias*] [,*permit_type*][,unused][,*basic_permission*\|*extended_permit*][,*application_permit*]") |
| Halt, page 265 | dmAPIExec("halt,*session*,*workflow_id* [,*activity_sequence_number*[,*suspend_interval*]]") |
| Id, page 267 | dmAPIGet("id,*session*,*qualification*") |
| Initcrypto, page 269 | dmAPIExec("initcrypto,apisession[,*location*][,*passphrase*]") |
| Insert, page 271 | dmAPISet("insert,*sesion*,*object_id*,*attribute*[[*index*]] [,*pattern*]","*value*") |

| Method | *Syntax* |
|---|---|
| Insertcontent, page 274 | dmAPISet("insertcontent,*session*,*object_id*[,*page_number*][,*length*][,*set_resource*]","*content*") |
| Insertfile, page 277 | dmAPIExec("insertfile,*session*,*object_id*,*file_name*[,*page_number*][,*other_file*]") |
| Insertpart, page 280 | dmAPIGet("insertpart,*session*,*document_id*,*component_id*[,*version_label*],*containment_id*\|*order_no*[,*orderno_flag*][,*use_node_ver_label*][,*follow_assembly*][,*copy_child*][,*containment_type*,*containment_desc*]") |
| Insertstate, page 287 | dmAPIGet("insertstate,*session*,*policy_id*[,*position*\|*state_name*]") |
| Install, page 289 | dmAPIExec("install,*session*,*object_id*[,*notify_flag*]")<br>dmAPIExec("install,*session*,*object_id*[,*install_activity*][,*resume*]") |
| Invalidate, page 291 | dmAPIExec("invalidate,*session*,*object_id*") |
| Iscached, page 293 | dmAPIGet("iscached,*session*,typecache,*cache_key*") |
| Kill, page 295 | dmAPIExec("kill,*session*,*session_to_kill*[,*immediacy_level*] [,*message*]") |
| Link, page 297 | dmAPIExec("link,*session*,*obj_id*, *folder_specification*\|*alias*") |
| Listconnection, page 300 | dmAPIGet("listconnection,*session*") |
| Listmessage, page 302 | dmAPIGet("listmessage,*session*[,*severity_level*]") |
| Locate, page 304 | dmAPIGet("locate,*session*,*object_id*,*attribute*,*value*") |
| Lock, page 306 | dmAPIExec("lock,*session*,*object_id*[,*validate_stamp*]") |
| Lpq, page 308 | dmAPIGet("lpq,*session*[,*printer*]") |
| Mark, page 310 | dmAPIExec("mark,*session*,*object_id*,*label*{,*label*}") |
| Mount, page 312 | dmAPIExec("mount,*session*,*object_id*,*path*") |
| Movestate, page 314 | dmAPIExec("movestate,*session*,*object_id*,*old_position*,*new_position*") |

| Method | *Syntax* |
|---|---|
| Next, page 316 | dmAPIExec("next,*session*,*collection_identifier*") |
| Offset, page 318 | dmAPIGet("offset,*session*,*object_id*,*attribute*") |
| Pause, page 320 | dmAPIExec("pause,*session*,*workitem_id*") |
| Print, page 321 | dmAPIGet("print,*session*,*object_id*[,*printer*][,*print_cover*] [,*save_output*][,*num_copies*][,*starting_content_page*] [,*ending_content_page*]") |
| Promote, page 323 | For unscheduled promotions:<br>dmAPIExec("promote,*session*,*object_id*[,*to_state*\|*to_position* ,*test_only_flag*,*override_flag*]")<br>For scheduled promotions:<br>dmAPIExec("promote,*session*,*object_id*,*scheduled_date*[,*pattern*],*to_state*\|*to_position*\|cancel[,*override_flag*]") |
| Prune, page 326 | dmAPIExec("prune,*session*,*object_id*[,*keepSLabel*]") |
| Publish_dd, page 329 | dmAPIExec("publish_dd,*session*[,*locale*][,*type*][,*attribute*][,*force_publish*]") |
| Purgelocal, page 332 | dmAPIExec("purgelocal,*session*") |
| Query, page 337 | dmAPIGet("query,*session*,*dql_query*") |
| Queue, page 339 | dmAPIGet("queue,*session*,*object_id*,*queue_owner*,*event*,*priority* [,*send_mail*],[*due_date*],*message*")<br>dmAPIGet("queue,*session*,*workflow_id*,[*supervisor*],*event_type*,[*priority*,[*send_mail*[,*due_date*[,*message*]]]]") |
| Readquery, page 342 | dmAPIGet("readquery,*session*,*dql_query*") |
| Refresh, page 344 | dmAPIExec("refresh,*session*,*object_id*") |
| Register, page 346 | dmAPISet("register,*session*,*object_id*,*event*[,*priority*][,*send_mail*]","*message*") |
| Reinit, page 349 | dmAPIExec("reinit,*session*[,*server_config_name*][,*send_to_docbroker*]") |

| Method | *Syntax* |
|---|---|
| Remove, page 351 | dmAPIExec("remove,*session*,*object_id*,*attribute*[[*index*]]") |
| Removeactivity, page 353 | dmAPIExec("removeactivity,*session*,*process_id*,*activity_identifier*") |
| Removecontent, page 355 | dmAPIExec("removecontent,*session*,*object_id*[,*page_number*]") |
| Removelink, page 357 | dmAPIExec("removelink,*session*,*process_id*,*link_identifier*") |
| Removenote, page 358 | dmAPIExec("removenote,*session*,*annotation_id*,*object_id*") |
| Removepackage, page 360 | dmAPIExec("removepackage,*session*,*object_id*,*start_activity_name*,*port_name*,*package_name*")<br>dmAPIExec("removepackage,*session*,*object_id*,*package_name*") |
| Removepackageinfo, page 362 | dmAPIExec("removepackageinfo,*session*,*activity_id*,*port_name*,*package_name*") |
| Removepart, page 364 | dmAPIExec("removepart,*session*,*document_id*, *containment_id*\|*order_no*[,*orderno_flag*]") |
| Removeport, page 367 | dmAPIExec("removeport,*session*,*object_id*,*port_name*") |
| Removerendition, page 368 | dmAPIExec("removerendition,*session*,*object_id*,*format*[,*page_number*][,*atomic*][,*page_modifier*]") |
| Removeroutecase, page 371 | dmAPIExec("removeroutecase,*session*,*activity_id*,*route_case_identifier*") |
| Removestate, page 373 | dmAPIExec("removestate,*session*,*object_id*[,*position*\|*state_name*]") |
| Repeat, page 375 | dmAPIGet("repeat,*session*,*workitem_id* {,*user_name*\|*group_name*}") |
| Repeating, page 377 | dmAPIGet("repeating,*session*,*object_id*,*attribute*") |
| Reset, page 380 | dmAPIExec("reset,*session*,*object_id*") |

| Method | *Syntax* |
|---|---|
| Resolvealias, page 382 | dmAPIGet("resolvealias,*session*,[*object_id*], %[*alias_set_name*.]*alias_name*") |
| Restart, page 386 | dmAPIExec("restart,*session*[,*server_config_name*] [,*reset_client*]")<br>dmAPIExec("restart,*session*,*workflow_id*[,*activity_seq_no*]") |
| Restore, page 389 | dmAPIGet("restore,*session*[,*predicate*][,*dump_file*] [,*operator*][,*priority*][,*send_mail*][,*due_date*]") |
| Resume, page 392 | dmAPIExec("resume,*session*,*object_id*[,*activity_seq_no*]")<br>dmAPIExec("resume,*session*,*sysobject_id* [,*from_state*\|*from_position*,*return_to_base_flag* ,*test_only_flag*,*override_flag*]")<br>dmAPIExec("resume,*session*,*sysobject_id*,*scheduled_date* ,[*pattern*],[*from_state*\|*from_position*\|cancel ,*return_to_base_flag*,*override_flag*]") |
| Retrieve, page 396 | dmAPIGet("retrieve,*session*,*qualification*") |
| Revert, page 398 | dmAPIExec("revert,*session*,*object_id*[,*acl_only*]") |
| Revoke, page 400 | dmAPIExec("revoke,*session*,*object_id*,*accessor_name*\|*alias* [,*permit_type*][,*unused*] [,*basic_permit*\|*extended_permit*\|*application_permit*]") |
| Save, page 405 | dmAPIExec("save,*session*,*object_id*[,*save_lock*] {,*version_label*}") |
| Saveasnew, page 409 | dmAPIGet("saveasnew,*session*,*object_id*,*share_content*[,*keep_storage_areas*]") |
| Seek, page 413 | dmAPIExec("seek,*session*,*collection*,*position*[,*direction*]") |
| Set, page 416 | dmAPISet("set,*session*,*object_id*,*attribute*[*[index]*] [,*pattern*]","*value*\|*alias*") |
| Setbatchhint, page 422 | dmAPIExec("setbatchhint,*session*,*size*") |
| Setcontent, page 424 | dmAPISet("setcontent,*session*,*object_id*[,*format*] [,*page_number*][,*length*][,*set_resource*]","*content*") |

| Method | *Syntax* |
|---|---|
| Setcontentattrs, page 427 | dmAPIExec("setcontentattrs,*session*,*object_id*,*format*[,*page_number*][,*page_modifier*],*parameters*") |
| Setdoc, page 432 | dmAPIExec("setdoc,*session*,*object_id*,*is_virtual_doc*") |
| Setfile, page 434 | dmAPIExec("setfile,*session*,*object_id*,*file_name* [,*page_number*\|*format*][,*other_file*]") |
| Setoutput, page 437 | dmAPIExec("setoutput,*session*,*workitem_id* {,*output_port_name*}") |
| Setpath, page 439 | dmAPIExec("setpath,*session*,*object_id*,*file_name* [,*page_number*\|*format*[,*other_file*]]") |
| Setperformers, page 441 | dmAPIExec("setperformers,*session*,*object_id*,*activity_name* [,*performer_list*]") |
| Setpriority, page 443 | dmAPIExec("setpriority,*session*,*workitem_id*,*priority*") |
| Setsupervisor, page 445 | dmAPIExec("setsupervisor,*session*,*workflow_id*, *new_supervisor*") |
| Shutdown, page 447 | dmAPIExec("shutdown,*session*[,*immediate*][,*delete_entry*]") |
| Signoff, page 449 | dmAPIExec("signoff,*session*,*object_id*,[*user_auth_name*], *password*[,*reason*]") |
| Suspend, page 453 | dmAPIExec("suspend,*session*,*sysobject_id* [,*from_state*\|*from_position*,*test_only_flag*,*override_flag*]")<br>dmAPIExec("suspend,*session*,*sysobject_id*,*scheduled_date*, [*pattern*],*from_state*\|*from_position*\|cancel [,*override_flag*]") |
| Trace, page 456 | dmAPIExec("trace,*session*,*severity_level*[,*logfile*], *option*\|*facility*\|all") |
| Truncate, page 462 | dmAPIExec("truncate,*session*,*object_id*,*attribute*") |
| Type, page 464 | dmAPIGet("type,*session*,*object_type*[,*attribute*] [,*bus_policy_id*,*state*]") |

| Method | *Syntax* |
|---|---|
| Unaudit, page 466 | dmAPIExec("unaudit,*session* [,*object_id*],*event_name* [,*controlling_app*][,*policy_id* ][,*state_name*]]") |
| Unfreeze, page 469 | dmAPIExec("unfreeze,*session*,*object_id*[,*thaw_components*]") |
| Uninstall, page 472 | DmAPIExec("uninstall,*session*,*object_id*[,*notify_flag*]") |
| Unlink, page 474 | dmAPIExec("unlink,*session*,*object_id*,*location*\|*alias*") |
| Unlock, page 477 | dmAPIExec("unlock,*session*,*object_id*[,*send_mail*] [,*compound_arch_value*][,*special_app_value*]") |
| Unmark, page 480 | dmAPIExec("unmark,*session*,*object_id*,*label*{,*label*}") |
| Unprint, page 482 | dmAPIExec("unprint,*session*[,*printer_name*],*job_id*") |
| Unregister, page 484 | dmAPIExec("unregister,*session*,*object_id*,*event*[,user]") |
| Updatepart, page 486 | dmAPIExec("updatepart,*session*,*virtdoc_id*,*containment_id* [,*version_label*][,*order_no*][,*use_node_ver_label*] [,*follow_assembly*][,*copy_child*] [,*containment_type*,*containtment_desc*]") |
| Useacl, page 491 | dmAPIExec("useacl,*session*,*object_id*, folder\|user\|type\|none") |
| Validate, page 494 | dmAPIExec("validate,*session*,*object_id*[,*check_activity*]") |
| Values, page 497 | dmAPIGet("values,*session*,*object_id*,*attribute*") |
| Vdmpath, page 501 | dmAPIGet("vdmpath,*session*,*component_id*[,*root_id*] [,*shortest_path*]{,*version_list*} |
| Vdmpathdql, page 506 | dmAPIGet("vdmpathdql,*session*,*component_id*[,*root_id*] [,*shortest_path*][,*type*][,*binding_condition*] [,*nodesort_by*]") |

| Method | *Syntax* |
| --- | --- |
| Verifyaudit, page 510 | `dmAPIExec("verifyaudit,`*`session`*`,`*`object_id`*`")` |
| Verifyesignature, page 512 | `dmAPIExec("verifyesignature,`*`session`*`,`*`object_id`*`")` |

# Appendix B

# System Events

This appendix lists and describes the system events recognized by Content Server. You can use an Audit method to register any of these events for auditing. You can also use a Register method to register to receive an Inbox notification for any of these events.

## API events

Table B–1, page 539 lists the events arising from API methods. (Events arising from methods that are specific to workflows are listed separately in Table B–2, page 544.)

**Table B-1. Events arising from API 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_adddigsigna-ture | dm_sysobject | Add Digital Signature | Execution of the Adddigsignature method.<br><br>**Note:** Adddigsignature methods are always audited. It is not possible to unregister this event. |

| Event | Target object type | Event description | Trigger |
|---|---|---|---|
| dm_addesignature | dm_sysobject | Add Electronic Signature | Execution of the Addesignature method.<br><br>Addesignature methods are always audited and the entries are always signed by Content Server.It is not possible to modify this behavior. |
| dm_addnote | dm_sysobject<br>dm_process | Add Note | Addition of a note to a SysObject.<br><br>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 Audit method.<br><br>**Note:** Audit methods are always audited. It isn't possible to unregister this event. |
| dm_authenticate | dm_user | Authenticate User | Execution of Authenticate method |
| dm_branch | dm_sysobject<br>dm_retainer | Branch Version | Execution of Branch method |
| dm_checkin | dm_sysobject<br>dm_retainer | Checkin Object | Execution of Checkin method |
| dm_checkout | dm_sysobject<br>dm_retainer | Checkout Object | Execution of Checkout method |
| dm_connect | dm_user | Logon | Execution of Connect method |

| Event | Target object type | Event description | Trigger |
|---|---|---|---|
| dm_destroy | dm_sysobject<br>dm_acl<br>dm_user<br>dm_group<br>dm_retainer | Destroy Object | Execution of Destroy method |
| dm_disassemble | dm_sysobject | Disassemble Virtual Document | Execution of Disassemble method |
| dm_disconnect | dm_user | Logoff | Execution of Disconnect method |
| dm_fetch | dm_sysobject<br>dm_retainer | Fetch Object | Execution of Fetch method |
| dm_freeze | dm_sysobject | Freeze Object | Execution of Freeze method |
| dm_getfile | dm_sysobject | View/Export | Occurs when a document is viewed or exported or when the following methods are executed: Getcontent and Getpath. |
| dm_getlogin | dm_user | Get Login | Execution of Getlogin method |
| dm_insertpart | dm_sysobject | Insert into Virtual Document | Execution of an Insertpart method |
| dm_install | dm_policy<br>dm_process<br>dm_activity | Install | Execution of an Install method |
| dm_invalidate | dm_process<br>dm_activity | Invalidate | Execution of an Invalidate method |
| dm_kill | not applicable | Kill Session | Execution of a Kill method<br><br>**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 |
| dm_logon_failure | dm_user | Logon Failure | User attempts to Connect 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<br>dm_retainer | Prune Versions | Execution of a Prune method |

| Event | Target object type | Event description | Trigger |
|---|---|---|---|
| dm_purgeaudit | dm_audittrail<br>dm_audittrail_acl<br>dm_audittrail_<br>group | Purge Audit | Execution of a Destroy method or PURGE_AUDIT administration method to remove audit trail entries<br><br>Purging an audittrail entry is always auditing. It is not possible to unaudit this event. |
| dm_remove_<br>dynamic_group | dm_group | Remove From Dynamic Group | Execution of an IDfSession.<br>removeDynamicGroups call |
| dm_removecon-<br>tent | dm_sysobject | Remove Content | Execution of a Removecontent method |
| dm_removenote | dm_sysobject<br>dm_process | Remove Note | Execution of a Removenote method |
| dm_removepart | dm_sysobject | Remove from Virtual Document | Execution of a Removepart method |
| dm_removerendi-<br>tion | dm_sysobject | Remove Rendition | Execution of a Removerendition method |
| dm_removereten-<br>tion | dm_sysobject | Remove Retention | Object removed from control of a retention policy |
| dm_restore | dm_sysobject | Restore Object | Execution of a Restore method |
| dm_save | dm_sysobject<br>dm_acl<br>dm_group<br>dm_user<br>dm_retainer | Save Object | Execution of a Save method.<br><br>**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<br>dm_acl<br>dm_retainer | Copy Object | Execution of a Saveasnew method |
| dm_setfile | dm_sysobject | Set Content | Execution of the following methods:<br><br>Appendcontent Appendfile Bindfile Insertfile Insertcontent Setcontent Setpath |
| dm_setoutput | dm_process | Setoutput | Execution of a Setoutput method. |

| Event | Target object type | Event description | Trigger |
|---|---|---|---|
| dm_setretention-status | dm_retainer | Set Retention Status | Change to status of a retainer object. |
| dm_signoff | dm_sysobject | Signoff | Execution of Signoff method<br><br>**Note:** Signoff methods are always audited. It isn't possible to unregister this event. |
| dm_unaudit | all object types | Unaudit Event | Execution of an Unaudit method<br><br>**Note:** Unaudit methods are always audited. It isn't possible to unregister this event. |
| dm_unfreeze | dm_sysobject | Unfreeze Object | Execution of an Unfreeze method |
| dm_uninstall | dm_policy<br>dm_process<br>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 an Unlock method |
| dm_unmark | dm_sysobject | Remove Version Label | Execution of an Unmark method |
| dm_updatepart | dm_sysobject | Update in Virtual Document | Execution of an Updatepart method |
| dm_validate | dm_policy<br>dm_process<br>dm_activity | Validate | Execution of a Validate method |

# Workflow events

Table B–2, page 544 lists the events specific to workflows. Note that several API events listed in the previous table are also applicable to workflows.

**Table B-2. 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<br><br>**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 |
| dm_changedactivity instancestate | 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_changestateactiv-ity | 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_changestatepro-cess | dm_process | Change Workflow Template State | A process definition changes state |

| Event | Target object type | Event description | Trigger |
|---|---|---|---|
| dm_changestatework-flow | 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. |
| dm_portselect | dm_process | Select Workflow Port | Selection of output ports by a user or Content Server upon completion of an activity.<br><br>**Note:** This event is not triggered if the activity has a transition type of prescribed. |
| dm_pseudocomplet-edworkitem | dm_process | Pseudo_Complete Workitem | A work item is marked as pseudo-completed by Content Server |
| dm_removeattach-ment | 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. |

| Event | Target object type | Event description | Trigger |
|---|---|---|---|
| dm_selectedworkitem | dm_process | Select Workitem | A work item is acquired. |
| dm_startworkflow | dm_process | Start Workflow | A workflow is started. |
| dm_startedworkitem | dm_process | Start Workitem | A work item is generated. |
| dm_wf_autodelegate_failure | dm_process | Auto Delegation Failed | Automatic delegation of an activity failed. |

# Lifecycle events

Table B–3, page 546 lists the events specific to lifecyles.

**Table B-3. LIfecycle events**

| Event | Target object type | Event description | Trigger |
|---|---|---|---|
| dm_bp_attach | dm_sysobject<br>dm_retainer | Attach Lifecycle | Attaching a lifecycle to an object |
| dm_bp_demote | dm_sysobject<br>dm_retainer | Demote from Lifecycle State | Demoting an object from a lifecycle state |
| dm_bp_promote | dm_sysobject<br>dm_retainer | Promote to Lifecycle State | Promoting an object to a lifecycle state |
| dm_bp_resume | dm_syobject<br>dm_retainer | Resume Lifecycle | Resuming an object's suspended lifecycle |
| dm_bp_suspend | dm_sysobject<br>dm_retainer | Suspend Lifecycle | Suspending an object's lifecycle |

# Index

## A

## B



## C

# D

## E



## F



## G

## H



## I



## J



## K



## L



## M

## N



## O



## P

## Q



## R

## S



## T

## U



## V



## W