Content Server DQL Reference Manual

documentum

Version 5.2.5
December 31, 2003
All Platforms

Table of Contents

List of Figures

List of Tables

Preface

This manual is the reference manual for Documentum's Document Query Language, supported by Content Server. It is a companion to *Content Server API Reference Manual*, *Content Server 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 that uses DQL. 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

Revision History

The following changes have been made to this document.

Revision History

Revision Date	Description
December 2003	Initial publication
December 31, 2003	Republished Content Server documentation set to fix cross references that did not have associated page numbers.

Chapter 1

DQL Language Elements

This chapter describes the building blocks of a DQL statement, including:

- Literals, page 13, which describes the literal formats for the Documentum datatypes
- Special Keywords, page 18, which describes the special keywords that you can use in DQL queries
- Functions, page 19, which describes the functions that you can use in DQL queries
- Predicates, page 27 , which describes the predicates that you can use in expressions in queries
- Logical Operators, page 35 , which describes the logical operators supported by DQL
- DQL Reserved Words, page 36 , which cross-references you to a list of the words reserved in DQL

Literals

Literals are values that are interpreted by the server exactly as they are entered. Content Server recognizes five types of literals:

- Integer Literals, page 13
- Floating Point Literals, page 14
- Character String Literals, page 14
- ID Literals, page 15
- Date Literals, page 15

Integer Literals

An integer literal specifies any whole number and is expressed in the following format:

```
[+ | -] n
```

where *n* is any number between -2147483647 and +2147483647.

DQL does not support the negative integer value -2147483648 because this number is not supported in a number of relational databases. If you enter this number, your results are unpredictable.

Floating Point Literals

A floating point literal specifies any number that contains a decimal point and is expressed in the following format:

```
5.347
21.
0.45
.66
-4.12
```

A floating point literal can also be expressed in scientific notation. For example:

```
10.4e-6
```

or

```
-3.6E7
```

or

```
12e-3
```

DQL accepts either uppercase or lowercase in the notation.

If you assign an integer literal to an attribute that has a floating point datatype, the system automatically converts the integer to a floating point number.

The underlying RDBMS determines the maximum and minimum values that you can assign as a floating point literal. Table 1–1, page 14 lists the ranges for supported databases.

Note: Do not reset the decimal symbol at the operating system level to a comma. Doing so results in incorrect execution of some Documentum Administration jobs.

Table 1–1. Valid Ranges for Floating Point Literals

RDBMS	Range
Oracle	1.0×10^{-129} to 9.99×10^{-129}
DB2	1.0×10^{-307} to $1.798 \times 10^{+308}$
MS SQL Server and Sybase	$1.7e^{-308}$ to $1.7e^{+308}$

Character String Literals

Character string literals are strings of printable characters and are enclosed in single quotes. You cannot place non-printable characters, such as line feeds or carriage returns, in a character string literal. To include a single quote as part of the literal, include it twice. For example:

```
'The company''s third quarter results were very good.'
```

If you are running against an Oracle RDBMS, a character string literal can contain a maximum of 2,000 characters.

If you are running against an MS SQL Server or Sybase RDBMS, a character string literal can contain a maximum of 255 characters.

If you are running against a DB2 RDBMS, a character string literal contain a maximum of 32,672 characters.

ID Literals

An ID literal specifies an object ID as a 16-character string enclosed in single quotes. ID literals are generally used in DQL queries. For example:

```
SELECT "object_name" FROM "dm_document"
WHERE "r_object_id" = '099af3ce800001ff'
```

Note that you cannot use ID literals to assign object IDs to objects. Object IDs are assigned automatically by the server when the object is created.

Date Literals

A date literal specifies a date using the following syntax:

```
DATE(date_value [,'pattern'])
```

date_value can be defined using any of the valid character string formats representing a date, or it can be one of the keywords that represent dates.

There are some date formats that the server can interpret as more than one date. For example, 03/04/96 can be interpreted as March 4, 1996 or as April 3, 1996. To resolve this, some formats require you to specify what pattern the server uses to interpret the date.

Valid dates range from 01/01/1753 (January 1, 1753) to 12/31/9999 (December 31, 9999).

The following paragraphs describe the character string formats and keywords you can use to specify a date. When you use a character string format, you must enclose date_value in single quotes.

Default Formats

Table 1–2, page 15 lists the character string formats for *date_value* that are accepted by default:

Table 1–2. Default Date Character String Formats

Format	Examples
mm/dd/[*yy*]*yy*	DATE('03/24/1989') DATE('4/7/1992')
dd-mon-[*yy*]*yy*	DATE('4-Apr-1975')
month dd[,] [*yy*]*yy*	DATE('January 1, 1993')
mon dd [*yy*]*yy*	DATE('March 23 1990')

Format	Examples
the client's localized short date format	DATE('30-11-1990') (Assumes short date format *dd-mm-yyyy* is defined on the client machine)
ANSI format (*dow mon dd hh:mm:ss yyyy*)	No example

Although it is not required, you can specify a pattern for any of these formats except the short date format and the ANSI format. You cannot specify a pattern when you enter a date using either of those two formats.

When using the formats listed in Table 1–2, page 15, the following rules apply:

- It is not necessary to include leading zeros for months or days that are represented as a single digit.
- You can abbreviate a month's name to three letters.
- If you enter only the year and not the century, the server uses the century defined by the current date in the server machine. For example, 03/23/95 is interpreted as March 23, 1995 before the year 2000 and is interpreted as March 23, 2095 after the year 2000.

Short Date Formats

The server accepts a client's localized short date format as a valid character string format as long as the format contains only numeric characters. For example, *dd-mmm-yy* is not an accepted short date format.

Windows, Solaris, and AIX client platforms provide a default short date format. The Windows platform also lets you define your own default short date format. (Refer to Defining Short Date Formats, page 159 of the *Content Server Administrator's Guide* for information about defining short date formats.) For Sun and HP clients, which do not have a default short date format, the server assumes the default format is *mm/dd/yy hh:mi:ss*.

If the locally defined short date format conflicts with one of the default input formats, the locally defined format takes precedence. For example, assume that the locally defined format is *dd/mm/yyyy*. If a user wants to enter March 14, 1994 and enters 03/14/1994 (*mm/dd/yyyy*), the server interprets this as the 3rd day of the 14th month of the year 1994 and returns an error because there is no 14th month.

If you use the pattern argument to specify a format for a date, that pattern takes precedence over the short date format.

ANSI Format

You can also specify the date using the ANSI format (*dow mon dd hh:mm:ss yyyy*). However, DQL ignores the time fields. (To enter a date and time using the ANSI format, use the Set method.)

Other Character String Formats

The following character string formats for date_value are also acceptable but require a pattern argument:

[*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*]

If you do not enter the time (hh:mi:ss), the server assumes the time is 00:00:00. If you do not enter the day or month, the server assumes the first day of the month or the first month of the year, respectively.

For example, suppose the pattern argument is [*dd*/]*mm*/[*yy*]*yy* [*hh:mi:ss*]. The following date literal is interpreted as April 1, 1996 00:00:00 (assuming the current century is 1900).

```
DATE('04/96','mm/yy')
```

When you specify *date_value*, you can use any character except a space or an alphanumeric character as the delimiter. You must use the same delimiter to separate all elements of the date or time. For example, if you use a forward slash to separate *dd* and *mm*, you must use a forward slash to separate *mm* and *yy*. Similarly, if you use a period to separate *hh* and *mi*, you must use a period to separate *mi* and *ss*.

The delimiter that you use in the date can be different from the delimiter that you use in the time portion. For example, the following is valid:

```
'23-04-96 12.32.04'
```

It is not necessary to use the same delimiter for the *date_value* and pattern arguments. For example, the following function call is valid:

```
DATE('23-04-96 12.32.04','dd/mm/yy hh:mi:ss')
```

Date Literal Keywords

Four keywords represent dates. They are:

- TODAY, which returns the current date. The time defaults to 00:00:00. For example,

```
SELECT "supervisor_name", "object_name" FROM "dm_workflow"
WHERE "r_start_date" <= DATE(TODAY)
AND "r_runtime_state"=1
ORDER BY 1
```

- NOW, which returns the current date and time. For example,

```
SELECT "supervisor_name", "object_name" FROM "dm_workflow"
WHERE "r_start_date" <= DATE(NOW) AND "r_runtime_state"=1
ORDER BY 1
```

- YESTERDAY, which returns the current date minus one day. The time defaults to 00:00:00. For example,

```
SELECT * FROM dm_document
WHERE "r_creation_date" >= DATE(YESTERDAY)
AND
"r_creation_date" <= DATE(TODAY)
AND
ANY "authors" IN (john,henry,jane)
```

- TOMORROW, which returns the current date plus one day. The time defaults to 00:00:00. For example,

  ```
  SELECT "r_act_name", "object_name", "process_id"
  FROM "dm_workflow"
  WHERE ANY "r_pre_timer" >= DATE(TOMORROW)
  ORDER BY 2
  ```

Date Output Formats

By default, the server uses the client's localized short date format to output dates to the client. If the client's format represents years using two digits, dates in the current century are displayed using two digits. However, to avoid ambiguity, years in other centuries are displayed using all four digits (1834 or 1792).

The default short date formats, by platform, are:

- Windows NT 4.0 and Windows 2000: The default short date format is the format specified by selecting Control Panel>Regional Settings>Date/Time.
- UNIX/SunOS: The format is assumed to be *mm/dd/yy hh:mi:ss*.
- UNIX/Solaris and UNIX/AIX: The default short date format is the format defined by the machine's locale. (Locale is set by the UNIX setlocale command. Refer to Defining Short Date Formats, page 159 in the *Content Server Administrator's Guide* for information about setting this value.)
- UNIX/HP: The format is assumed to be *mm/dd/yy hh:mi:ss*.

  Note: The session config attribute r_date_format contains the date format that the server returns to a client for a given session.

If the date is entered as NULLDATE, then the output is the string NULLDATE.

You can override the default format using an argument to the Get method. The Get method returns the value of a specified attribute. If that attribute has a Date datatype, you can include a pattern argument for the Get method that defines the format in which the date is returned. For more information, refer to Get, page 195 of the *Content Server API Reference Manual*.

Dates that are included in error messages or a dump file are displayed in ANSI date format.

Special Keywords

DQL includes some special keywords for use in queries. These keywords have special meanings for Content Server when used in a DQL query. (They cannot be used in API methods.) The keywords are:

- USER, which identifies the current user.

  You can use USER in comparison operations in queries. For example,

  ```
  SELECT "process_id", "object_name" FROM dm_workflow
  WHERE supervisor_name=USER
  ```

- TRUE and FALSE, which represent the Boolean true and false.

You can use TRUE and FALSE in comparison operations involving any attribute having a BOOLEAN datatype. For example,

```
SELECT * FROM "dm_user"
WHERE "r_is_group" = TRUE
```

- DM_SESSION_DD_LOCALE, which represents the data dictionary locale most appropriate for the client's session locale.

  This keyword is particularly useful if the client session locale has no exact match recognized by the server. If you use this keyword, the server will return the information from the best matching locale. For example, suppose the server recognizes three locales, English (en), French (fr), and Spanish (es) and the client session locale is French Canadian (fr_cn). Suppose the client issues the following query:

  ```
  SELECT type_name, label_text from dmi_dd_type_info where
  nls_key='fr_cn'
  ```

  The query returns nothing because the locale fr_cn is not recognized by the server. However, the following query returns the information from the French locale:

  ```
  SELECT type_name, label_text from dmi_dd_type_info where
  nls_key=DM_SESSION_DD_LOCALE
  ```

  The server checks whether the locale identified in the client's session locale, fr_cn, exists in the Docbase. Since fr_cn isn't found, the server attempts to find a good match among the existing locales. It determines that fr (French) is a good match and uses that locale to execute the query.

  If a good match isn't found, the server uses the default locale to execute the query.

Functions

Functions are operations on values. DQL recognizes three groups of functions and two unique functions:

- Scalar Functions, page 20

  Scalar functions operate on one value and return one value.
- Aggregate Functions, page 21

  Aggregate functions operate on a set of values and return one value.
- Date Functions, page 23

  Date functions operate on date values.
- The ID Function, page 26

  The ID function, a unique function recognized by DQL, is used in the FOLDER and CABINET predicates and in the IN DOCUMENT and IN ASSEMBLY clauses.
- The MFILE_URL Function, page 26

  The MFILE_URL function returns URLs to content files and renditions in particular format.

Scalar Functions

The three scalar functions are:

- UPPER
- LOWER
- SUBSTR

UPPER

The UPPER function takes one argument and returns the uppercase of that value. The value supplied as the argument must be a character string or an attribute that has a character string datatype.

For example, the following statement returns the object names of all documents that have the word government in their title. Because LIKE returns only exact matches, the UPPER function is used to ensure that all instances are found, regardless of the case (upper, lower, or mixed) in which the word appears in the title.

```
SELECT "object_name" FROM "dm_document"
WHERE UPPER("title") LIKE '%GOVERMENT%'
```

LOWER

The LOWER function takes one argument and returns the lowercase of that value. The value supplied as the argument must be a character string or an attribute that has a character string datatype.

For example, the following statement returns the subjects in lowercase of all documents owned by regina:

```
SELECT LOWER("subject") FROM "dm_document"
WHERE "owner_name" = 'regina'
```

SUBSTR

The SUBSTR function returns some or all of a particular string. Its syntax is:

```
substr(string_value,start[,length])
```

The value of string_value can be a literal string or a column or attribute name. If you specify a column or attribute, the server uses the value in that column or attribute as the string value. The attribute you specify can be either a single-valued attribute or a repeating attribute.

The start argument identifies the starting position, in the specified string, of the substring you want returned. Positions are numbered from the left, beginning at 1. (If you specify 0 as the start, the server automatically begins with 1.) The argument must be an integer.

The length argument defines how many characters should be returned in the substring.

Length is an optional argument. If you don’t specify a length, the default behavior of the function differs depending on the RDMBS you are using. For Oracle and DB2, the default is the entire string value or the full width of the column if an attribute or column is specified. For all other databases, the function returns 1 character.

For example, suppose you have a document subtype called purchase_order, which has an attribute called order_no. The values in this attribute are 10-digit numbers, with the last three digits representing a specific sales region. The following statement returns all documents for which the last three digits in the order_no attribute are 003:

```
SELECT "r_object_id","order_no" FROM "purchase_order"
WHERE SUBSTR("order_no",8,3) = ’003’
```

You can also use the SUBSTR function with the SELECT DQL statement (refer to Select, page 109). For example:

```
SELECT SUBSTR("emp_name",1,4) AS short_name FROM "employee"
```

You must use the AS clause option to rename the query result attribute that holds the return value of the SUBSTR function. If you do not, the server cannot return the result of the function.

If you specify a repeating attribute, the function returns one value for each value in the attribute.

You cannot use the SUBSTR function in assignment statements.

Additionally, you cannot specify SUBSTR in a LIKE predicate. Refer to Pattern Matching with LIKE, page 31 for more information about using LIKE.

Aggregate Functions

The five aggregate functions are:

- COUNT
- MIN
- MAX
- AVG
- SUM

COUNT

The COUNT function counts values. The syntax is:

```
COUNT ([DISTINCT] name | *)
```

The name argument must identify an attribute or column. You can count all values for the attribute or column or you can include the DISTINCT keyword to count the number of unique values.

Using an asterisk directs the server to count all items that match specified criteria. For example, the following statement counts all documents that belong to the user named grace:

```
SELECT COUNT(*) FROM "dm_document"
WHERE "owner_name" = 'grace'
```

MIN

The MIN function returns the minimum value in a given set of values. The syntax is:

```
MIN(DISTINCT name | [ALL] value_expresssion)
```

The DISTINCT *name* argument directs the server to first select all distinct values from the set (no duplicates) and then return the minimum value. *name* can be an attribute or column name. Note that for this function, the DISTINCT name option has little meaning.

[ALL] *value_expression* directs the server to return the minimum value found in the set of values specified by the *value_expression* argument. *value_expression* can be any valid numeric expression or an attribute or column name. The keyword ALL is optional; whether it is specified or omitted, all of the values in the set are evaluated.

For example, assuming that rental_charges is a user-defined object type, the following statement returns the minimum rent charged for a two-bedroom apartment:

```
SELECT MIN("charge") FROM "rental_charges"
WHERE "style" = 'apt' AND "bedroom" = 2
```

MAX

The MAX function returns the maximum value in a given set of values. The syntax is:

```
MAX(DISTINCT name | [ALL] value_expresssion)
```

The DISTINCT *name* argument directs the server to first select all distinct values from the set (no duplicates) and then return the maximum value. *name* can be an attribute or column name. Note that for the MAX function, this option has little meaning.

[ALL] *value_expression* directs the server to return the maximum value found in the set of values specified by the *value_expression* argument. *value_expression* can be any valid numeric expression or an attribute or column name. The keyword ALL is optional; whether it is specified or omitted, all of the values in the set are evaluated.

For example, assuming that rental_charges is a user-defined object type, the following statement returns the maximum rent charged for a two-bedroom apartment:

```
SELECT MAX("charge") FROM "rental_charges"
WHERE "style" = 'apt' AND "bedroom" = 2
```

AVG

The AVG function returns an average. The syntax is:

```
AVG(DISTINCT name | [ALL] value_expression)
```

The DISTINCT *name* option directs the server to select all distinct values from the set (no duplicates) and return the average of those distinct values. *name* can be an attribute or column name.

[ALL] *value_expression* directs the server to return the average value found in the set of values specified by the *value_expression* argument. The value of *value_expression* can be any valid numeric expression or an attribute or column name. Use the optional keyword ALL to include all of the values in the set in the operation.

For example, assuming that rental_charges is a user-defined object type, the following statement returns the average rent charged on a two-bedroom apartment:

```
SELECT AVG("charge") FROM "rental_charges"
WHERE "style" = 'apt' AND "bedroom" = 2
```

SUM

The SUM function returns a total. The syntax is:

```
SUM(DISTINCT name | [ALL] value_expression)
```

The DISTINCT *name* option directs the server to select all distinct values from the set (no duplicates) and return the sum of those distinct values. *name* can be an attribute or column name.

[ALL] *value_expression* directs the server to return the sum of the values in the set of values specified by the *value_expression* argument. The value of *value_expression* can be any valid numeric expression or an attribute or column name. Use the optional keyword ALL to include all of the values in the set in the operation.

For example, assuming that rent_records is a user-defined object type that contains payment records of tenants, the following statement provides a total of the rents paid in May:

```
SELECT SUM("rent_amt") FROM "rent_records"
WHERE "mon" = 'may' AND UPPER("paid") = 'Y'
```

Date Functions

The four date functions are:

- DATEDIFF (refer to DATEDIFF , page 23)
- DATEADD (refer to DATEADD, page 24)
- DATEFLOOR (refer to DATEFLOOR, page 25)
- DATETOSTRING (refer to DATETOSTRING, page 25)

DATEDIFF

The DATEDIFF function subtracts two dates and returns a number that represents the difference between the two dates. The syntax is:

```
DATEDIFF(date_part, date1, date2)
```

The *date_part* argument defines the units in which the return value is expressed. Valid values are year, month, week, and day.

The *date1* and *date2* arguments specify the dates to be subtracted. *date1* is subtracted from *date2*. You can specify the dates as date literals or the names of single-valued attributes with a date datatype.

For example, the following statement uses the DATEDIFF function to return all tasks that were started a month or more late:

```
SELECT "task_number", "supervisor_name"
FROM dm_tasks_queued
WHERE
DATEDIFF(month,"plan_start_date", "actual_start_date")>=1
```

This next example returns all tasks that were started more than one week ago:

```
SELECT "task_number", "r_task_user"
FROM dm_tasks_queued
WHERE DATEDIFF(week, "actual_start_date", DATE(TODAY))>=1
```

If the Docbase™ is using Oracle or DB2, the return value is a floating point number.

If the Docbase is using DB2, the server assumes 365 days per year and 30.42 days per month (the mean number of days in a month). These assumptions can cause return values that differ from the expected value. To illustrate, the following example, which asks for the number of days between March 1, 1996 and Feb 1, 1996, returns 30.42 instead of 28:

```
DATEDIFF(day, date('03/01/1996 0:0:0'),
 date('02/01/1996 0:0:0'))
```

If the Docbase is using MS SQL Server or Sybase, the return value is an integer for all units except day. If you specify day in the function, the return value is a floating point.

The MS SQL Server and Sybase implementations round up if the difference is one half or greater of the specified unit. The implementations round down if the difference is less than one half of the specified unit. To illustrate, the following example, which asks for the difference between March 1, 1996 and July 1, 1996 expressed as years, returns 0 because the difference is only 4 months.

```
DATEDIFF(year,date('03/01/1996'),date('07/01/1996'))
```

DATEADD

The DATEADD function adds a number of years, months, weeks, or days to a date and returns the new date. The syntax is:

DATEADD(*date_part*, *number*, *date*)

The*date_part* argument defines the units that are being added to the specified date. Valid date parts are year, month, week, and day.

The *number* argument defines how date_part values are being added to the date. For example, the following statement uses the DATEADD function to obtain all tasks that started more than a week ago but are not yet finished:

```
SELECT "task_number", "supervisor_name" FROM dm_tasks_queued
WHERE DATEADD(week, 1, "actual_start_date") < DATE(TODAY)
AND "actual_completion_date" IS NULLDATE
AND NOT("actual_start_date" IS NULLDATE)
```

DATEFLOOR

The DATEFLOOR function rounds a given date down to the beginning of the year, month, or day. The syntax is:

`DATEFLOOR(`*`date_part`*`,`*`date`*`)`

The *date* argument is the name of a date attribute. The function rounds the value of the attribute down to the beginning of the unit specified as the value of date_part. Valid date_part values are year, month, and day.

For example, suppose that a document's r_creation_date attribute has a value of March 23, 1996 at 9:30:15 am. Using the DATEFLOOR function on this attribute returns these values:

Specifying	Returns
`DATEFLOOR(year,"r_creation_ date")`	January 1, 1996 at 00:00:00
`DATEFLOOR(month,"r_creation_ date")`	March 1, 1996 at 00:00:00
`DATEFLOOR(day,"r_creation_date")`	March 23, 1996 at 00:00:00

DATETOSTRING

The DATETOSTRING function returns a date as a character string in a particular format. The syntax is:

`DATETOSTRING(`*`date`*`,'`*`format`*`')`

The date argument is the name of a date attribute. The format argument defines how you want the character string formatted. The format can be any of the valid character string input formats for date literals, subject to the following RDBMS restrictions and qualifications:

- MS SQL Server and Sybase do not support month. If you specify a format that contains month, such as *month dd yyyy*, the date is returned in the default short date format.
- Oracle and DB2 always return the month name as a fixed-length 9-character string. This means that if the name of the month is shorter than 9 characters, the string value is padded.

Suppose that a document's r_creation_date attribute has a value of May 14, 1995. Here are some examples of the values returned by the DATETOSTRING function using this date and a variety of formats.

Specifying	Returns
`DATETOSTRING("r_creation_date",'dd-mon-yy')`	`14-May-95`
`DATETOSTRING("r_creation_date",'mm/dd/yy')`	`05/14/95`
`DATETOSTRING("r_creation_date",'month yy')`	`May 95`

The ID Function

The ID function is used to identify a folder or cabinet whose contents you want to search. You can use the ID function in FOLDER and CABINET predicates and IN DOCUMENT and IN ASSEMBLY clauses. The syntax is:

```
ID('object_id')
```

The *object_id* argument identifies the folder, cabinet, or virtual document you want to search. The object must reside in the current Docbase.

Use the literal 16-character representation of the object's ID. For example, the following statement returns all documents in the specified folder with titles containing the characters Pond:

```
SELECT * FROM "dm_document"
WHERE FOLDER (ID('099af3ce800001ff'))
AND "title" LIKE '%Pond%'
```

The MFILE_URL Function

The MFILE_URL function returns URLs to content files or renditions associated with a document. Only those objects on which the user has at least Read permission are returned if MFILE_URL is included in a selected values list.

The syntax of MFILE_URL is:

```
MFILE_URL('format',page_no,'page_modifier')
```

The arguments are ANDed together to determine which URLs are returned. Only the format argument is required. page_no and page_modifier are optional.

The format argument restricts the returned URLs to those pointing to files in the specified format. Enclose the format value in single quotes. If you specify format as an empty string, Content Server takes the format value from the returned object's a_content_type attribute.

The page_no argument restricts the returned URLs to those pointing to content files associated with given page number. If you don't include page_no, the value defaults to 0, which represents the first content file associated with an object. To indicate that all content pages should be considered, define page_no as -1. If you define page_no as -1, Content Server returns all URLs that satisfy the other arguments regardless of the page with which the content files are associated.

The page_modifier argument restricts the returned URLs to those pointing to content files that have the specified page_modifier. The page_modifier argument must be enclosed in single quotes. If you specify a value for page_modifier (other than an empty string), all the returned URLs point to renditions, as content files for primary pages have no page modifier. To return URLS for content files that have no page modifier, define page_modifier as an empty string. (Content Server sets the page_modifier attribute to an empty string by default when a user adds a content file to an object or saves a rendition without a page modifier.) If you don't include page_modifier, Content Server returns the URLs that satisfy the other arguments regardless of whether their content files have a page modifier or not.

Examples

The following statement returns the URLs to the jpeg_th renditions of the first content pagesof documents owned by ronaldh that have a page modifier of image1.

```
SELECT MFILE_URL('jpeg_th',0,'image1')
FROM "dm_document" WHERE "object_owner"='ronaldh'
```

The following example returns the owner name and object ID of all documents that have the subject prairie chickens. For each returned document, it also returns URLs to the primary content files associated with the document.

```
SELECT owner_name,r_object_id,MFILE_URL('',-1,'')
FROM "dm_document" WHERE "subject"='prairie_chickens'
```

Predicates

Predicates are used in WHERE clauses to restrict the objects returned by a query. The WHERE clause defines criteria that each object must meet. Predicates are the verbs within the expression that define the criteria.

For example, the following statement returns only documents that contain the value approved in their keywords attribute:

```
SELECT "r_object_id", "object_name", "object_owner"
FROM "dm_document"
WHERE ANY "keywords" = 'approved'
```

In this example, the equal sign (=) is a predicate. It specifies that any object returned must have a keywords attribute value that equals (matches) the specified word (in this case, the word approved).

DQL recognizes these predicates:

- Arithmetic operators (refer to Arithmetic Operators, page 28)
- Comparison operators (refer to Comparison Operators, page 28)
- Column and attribute predicates (refer to Column and Attribute Predicates, page 28)
- SysObject predicates (refer to SysObject Predicates, page 33)

Arithmetic Operators

Arithmetic operators perform arithmetic operations on numerical expressions. Table 1–3, page 28 lists the arithmetic operators that you can use as predicates.

Table 1–3. Arithmetic Operators

Operator	Operation
+	Addition
–	Subtraction
/	Division
*	Multiplication
**	Exponentiation

Comparison Operators

Comparison operators compare one expression to another. Table 1–4, page 28 lists the comparison operators that can be used as predicates.

Table 1–4. Valid Comparison Operators for DQL Statements

Operator	Relationship
=	Equal
>=	Greater than or equal to
<=	Less than or equal to
>	Greater than
<	Less than
<>	Not equal
!=	Not equal

Column and Attribute Predicates

Column and attribute predicates let you compare values in a registered table's columns or an object's attributes to defined criteria. These predicates are divided into two groups. One group is used only when the search criterion specifies a column or single-valued attribute. The other group is used when the search criteria specifies a repeating attribute.

Predicates for Columns and Single-Valued Attributes

The predicates in this group allow you to compare a value in a table column or single-valued attribute to defined criteria. Table 1–5, page 29 lists the predicates in this group.

Table 1–5. Predicates for Columns and Single-Valued Attributes

Predicate	Description
IS [NOT] NULL	Determines whether the attribute is assigned a value. This predicate is useful only for registered table columns.
IS [NOT] NULLDATE	Determines whether the attribute is assigned a null date.
IS [NOT] NULLSTRING	Determines whether the attribute is assigned a null string.
IS [NOT] NULLINT	Determines whether the attribute is assigned a null integer value.
[NOT] LIKE *pattern* [ESCAPE *character*]	Determines whether the attribute is like a particular pattern. (Refer to The ESCAPE Character, page 32 for information about the ESCAPE clause option.)
[NOT] IN *value_list*	Determines whether the attribute is in one of a particular list of values.
[NOT] EXISTS (*subquery*)	Determines whether the attribute matches a value found by the subquery.
comparison_op SOME (*subquery*) *comparison_op* ANY (*subquery*)	Determines whether the comparison between the attribute value and the results of the subquery evaluate to TRUE in any case.
comparison_op ALL (*subquery*)	Determines whether the comparison between the attribute value and the results of the subquery evaluate to TRUE in all cases.

For example, the following statement selects all documents that have a title that starts with the word Breads:

```
SELECT * FROM "dm_document"
WHERE "title" LIKE 'Breads%'
```

This next example selects all workflows that are supervised by one of the users named in the predicate:

```
SELECT "r_object_id", "supervisor_name" FROM "dm_workflow"
WHERE "supervisor_name" IN ('carrie','davidk','holly')
ORDER BY 2
```

Predicates for Repeating Attributes

The predicates for repeating attributes let you compare the values in repeating attributes to some defined criteria. The basic syntax for repeating attribute predicates is:

```
[NOT] ANY predicate
```

Table 1–6, page 30 lists the predicates in this group.

Table 1–6. Predicates for Repeating Attributes

Predicate	Description
attr_name [NOT] LIKE *pattern* [ESCAPE *character*]	Evaluates to TRUE if any value of the repeating attribute is [not] like a particular pattern. (Refer to The ESCAPE Character, page 32 for information about the optional ESCAPE clause.)
attr_name IN (*value_list*)	Evaluates to TRUE if any value of the attribute matches a value in the *value_list*. *Value_list* is a comma-separated list of values.
[IN|EXISTS] *attr_name* IN (*subquery*)	Evaluates to TRUE if any value of the attribute matches a value returned by the subquery. IN and EXISTS are database hints that may enhance performance by changing the structure of the generated SQL query. Refer to Appendix A, Using DQL Hints, for more information about these optional hints.
attr_name IS [NOT] NULL	Evaluates to TRUE if any value of the attribute is [not] null.
attr_name IS [NOT] NULLDATE	Evaluates to TRUE if any value of the attribute is [not] a nulldate.
attr_name IS [NOT] NULLSTRING	Evaluates to TRUE if any value of the attribute is [not] a null string.
attr_name IS [NOT] NULLINT	Evaluates to TRUE if any value of the specified repeating attribute is [not] a null integer value.
attr_name comparison_op value_expression	Evaluates to TRUE if the comparison operation is TRUE for any value of the attribute.

For example, the following statement returns the object names of all documents with an author whose name begins with a letter in the first half of the alphabet:

```
SELECT "object_name" FROM "dm_document"
WHERE ANY "authors" <= 'M'
```

You can use logical operators to build more complex predicate expressions, such as:

```
[NOT] ANY predicate AND|OR [NOT] ANY predicate {AND|OR
[NOT] ANY predicate}
```

For example, the following statement selects all documents that have Ingrid as an author and a version label of 1.0:

```
SELECT "r_object_id", "object_name" FROM "dm_document"
WHERE ANY "authors" = 'Ingrid' AND
ANY "r_version_label" = '1.0'
```

The predicate statement returns all objects that meet the specified criteria regardless of where the qualifying value is found in the attribute's value list. For example, look at the following statement:

```
SELECT "object_name" FROM "dm_document"
WHERE ANY "authors" IN ('jeanine','harvey') AND
ANY "keywords" = 'working'
```

This statement returns the name of all documents with either jeanine or harvey as a value anywhere in the authors attribute values list and with the keyword working as a value in any position in the list of values in the keywords attribute.

In some circumstances, you may want only objects for which the specified values are found in the same respective positions. For example, assume that the search finds an object for which jeanine is the value at position 3 in the authors attribute. You want the name of this object only if the keyword working is in position 3 of the keywords attribute.

You can impose this restriction by enclosing the predicate expression in parentheses:

```
[NOT] ANY (predicate AND|OR [NOT]predicate {AND|OR [NOT]predicate})
```

For example, the statement below returns the names of documents for which either jeanine and working or harvey and working (or both) are values of the authors and keywords attributes, respectively, and occupy corresponding positions in the attributes.

```
SELECT "object_name" FROM "dm_document"
WHERE ANY ("authors" IN ('jeanine','harvey')
AND "keywords" = 'working')
```

Notes:

- To return values at a matching index position level, the keyword ANY is outside the parentheses.
- You cannot specify an index position at which to look for values. You can specify only that the values be found in the same respective positions.
- Using the logical operator OR returns the same results as using the basic syntax, because the server returns the object if either predicate is true.

For more information and examples of querying repeating attributes, refer to Repeating Attributes in Queries, page 269 of *Content Server Fundamentals*.

Pattern Matching with LIKE

The [NOT] LIKE predicate for both single-valued and repeating attributes lets you identify a pattern to match the attribute value. This pattern is specified as a character string literal. For example, the following predicate returns all objects whose subject contains the word Cake:

```
subject LIKE '%Cake%'
```

When you use LIKE, the value in the attribute must match the string exactly, including the case of each character. If you use NOT LIKE, then the comparison is TRUE for any value that does not match exactly.

Sometimes, however, you may not know the text of the character string. For example, you might want to retrieve all documents that have the word Cake in their title but not know all of the titles. For these instances, Documentum provides two pattern-matching characters. These characters serve as wildcards. The two characters are:

- The percent sign (%)
- The underbar (_)

You can include the pattern-matching characters anywhere in a character string.

The Percent Sign

The percent sign replaces 0 or more characters. For example, suppose the predicate contains:

```
"subject" LIKE 'Breads%'
```

This returns any object whose subject begins with the word Breads, regardless of the how many characters or words follow Breads.

The Underbar

The underbar replaces one character. For example, suppose the predicate contains:

```
"subject" LIKE 'Bread_'
```

This returns any object whose subject is the single word *Bread*, followed by a space, an *s* (Breads), or any other single, printable character.

Matching Cases

You can use the UPPER and LOWER functions to retrieve an object that matches a value regardless of the case of the letters in the value. For example, to retrieve any object with the word cake in the title, regardless of its position in the title or the case:

```
UPPER("title") LIKE '%CAKE%'
```

Using the UPPER function changes all titles to uppercase for the comparison so that the case of the characters is not a factor. You can use the LOWER function the same way:

```
LOWER("title") LIKE '%cake%'
```

The ESCAPE Character

There may be occasions when the pattern you want to match includes a percent sign (%) or an underbar (_). To match either of those characters literally, you must specify an escape character and insert that character before the percent sign or underscore in the pattern. Use the optional ESCAPE clause to specify the escape character.

For example, suppose you want to find all documents whose object names contain an underscore. Because the underscore is interpreted as a wild card by default, you must define and use an escape character in the query:

```
SELECT "r_object_id" FROM "dm_document"
WHERE "object_name" LIKE '%\_%' ESCAPE '\'
```

In the above example, the backslash is defined as the escape character. Placed in the pattern directly ahead of the underscore, it tells the server to treat the underscore as a literal character rather than a wild card.

In addition to the wild-card characters, an escape character can also escape itself. For example, suppose you want to match the string %_\ and that you have defined the backslash as the escape character.Your LIKE predicate would look like this:

```
LIKE '\%\_\\' ESCAPE '\'
```

In the above example, the backslash character escapes not only the percent sign and underscore but also itself.

Escape characters can escape only the two wild-card characters and themselves. If you insert an escape character before any other character in a pattern, it generates an error.

You can use any printable character as an escape character.

SysObject Predicates

Three predicates restrict the search specified in a FROM clause.

When you specify an object type in a FROM clause, the server examines that type and its subtypes for any objects that fulfill the conditions specified in the rest of the query. However, sometimes you may want to limit the search to specific subtypes, folders, or cabinets. These three predicates allow you to do so. They are:

- TYPE
- FOLDER
- CABINET

The TYPE Predicate

The TYPE predicate restricts the search to objects of a single type or one of its subtypes. The syntax is:

```
TYPE(type_name)
```

The type_name argument must identify a subtype of a type specified in the FROM clause.

For example, the following statement retrieves all documents of the type legal_doc or accounting_doc or a subtype:

```
SELECT * FROM "dm_document"
WHERE TYPE("legal_doc") OR TYPE("accounting_doc")
```

Using the TYPE predicate provides one way to select from more than one type. For example, to retrieve all documents or workflow process definitions that have been created after a particular date, you could use the following statement:

```
SELECT "r_object_id", "object_name", "owner_name", "r_creation_date"
FROM "dm_sysobject" WHERE TYPE("dm_document") OR TYPE("dm_process")
```

The FOLDER Predicate

The FOLDER predicate identifies what folders to search. The syntax is:

```
[NOT] FOLDER(folder_expression {,folder_expression} [,DESCEND])
```

The folder_expression argument identifies a folder in the current Docbase. You can't search a remote folder (a folder that doesn't reside in the current Docbase). Valid values are:

- An ID function
- The ID function (described in The ID Function, page 26) identifies a particular folder.
- A folder path

  A folder path has the format:

  ```
  /cabinet_name{/folder_name}
  ```

  Enclose the path in single quotes. Because cabinets are a subtype of folder, you can specify a cabinet as the folder.
- The keyword DEFAULT

  The keyword DEFAULT directs the server to search the user's default folder. Note that a user's default folder is the same as the user's default cabinet (because cabinets are a subtype of folders).

The DESCEND keyword directs the server to search the specified folder or folders and any local folders directly or indirectly contained within that folder or folders. The predicate does not search any contained, remote folders.

When the search finds a remote folder, it returns the object ID of the associated mirror object in the current Docbase. It does not return the object's ID in the remote Docbase.

DESCEND applies to all folders specified in the predicate. If you want to search one folder without descending but descend through another folder, include two folder predicates in your statement and OR them together. For example:

```
FOLDER ('/cakes/yeasted', DESCEND) OR FOLDER (DEFAULT)
```

The keyword NOT directs the server not to search a particular folder.

The CABINET Predicate

The CABINET predicate restricts the search to a particular cabinet. Its syntax is:

```
[NOT] CABINET(cabinet_expression [,DESCEND])
```

The cabinet_expression argument must identify a cabinet that resides in the current Docbase. Valid values are:

- An ID function

  The ID function (described in The ID Function, page 26) must specify a cabinet ID.
- A folder path

  The folder path must identify a cabinet. Its format is:

  ```
  /cabinet_name
  ```

  Enclose the path in single quotes.

The keyword DESCEND directs the server to search the specified cabinet and any folders directly or indirectly contained in the cabinet that reside in the current Docbase. The predicate does not search any contained, remote folders.

When the search finds a remote folder, it returns the object ID of the associated mirror object in the current Docbase. It does not return the object's ID in the remote Docbase.

The keyword NOT directs the server not to search a particular cabinet.

Logical Operators

Logical operators are used in WHERE clauses. DQL recognizes three logical operators:

- AND
- OR
- NOT

AND and OR

The AND and OR operators join two or more comparison expressions to form a complex expression that returns a single Boolean TRUE or FALSE. The syntax for these operators is:

```
expression AND | OR expression
```

Two expressions joined using AND return TRUE only if both expressions are true. For example, the following complex expression returns TRUE when both of its component expressions are true, and FALSE when only one of them is true:

```
"title" = 'Yeast Breads of England' AND "subject" = 'Breads'
```

Similarly, the following expression also returns TRUE only if both conditions are true:

```
"subject" = 'Breads' AND ANY "authors" IN ('clarice','michelle','james')
```

Two expressions joined using OR return TRUE if either expression is true. For example, the following expression returns true when either comparison is true:

```
"subject" = 'Cheeses' OR "owner_name" = 'donaldm'
```

NOT

The NOT operator reverses the logic of an expression. The following expression is a simple example:

```
"subject" = 'Cheeses'
```

When the server encounters this expression, it is looking for objects that have a subject attribute with the value Cheeses. The server returns TRUE if the subject value is Cheeses and FALSE if it is not.

Now, look at the same expression with the NOT operator:

```
NOT("subject" = 'Cheeses')
```

In this instance, the server looks for the objects with subject attributes that do not contain the value Cheeses. The server returns TRUE if the subject is not Cheeses and FALSE if the subject is Cheeses.

Order of Precedence

You can join any number of expressions together with logical operators. Content Server™ imposes no limit. (Note that your underlying RDBMS may impose a limit.) The resulting complex expression is evaluated from left to right in the order of precedence of the operators. This order, from highest to lowest, is:

NOT
AND
OR

To illustrate, look at the following example:

```
NOT expression1 AND expression2 OR expression3 AND expression4
```

The server resolves this expression as follows:

1. Evaluates NOT expression1
2. ANDs the result of Step 1 with the result of expression2
3. ANDs the results of expression3 and expression4
4. ORs the results of Steps 2 and 3

You can change the order of evaluation by using parentheses. For example:

```
NOT expression1 AND (expression2 OR expression3) AND expression4
```

The server resolves this expression as follows:

1. Evaluates NOT expression1
2. ORs the results of expression2 and expression3
3. ANDs the results of Step 1 and Step 2
4. ANDs the results of Step 3 with the result of expression4

Similarly, you can use parentheses to change the precedence of the NOT operator:

```
NOT(expression1 AND expression2 AND expression3)
```

The complex expression inside the parentheses is evaluated first and the NOT operator applied to its result.

DQL Reserved Words

Table B–1, page B–329 lists the DQL reserved words. If you use any of these as object or attribute names, you must enclose the name in double quotes whenever it is used in a DQL statement.

Chapter 2

DQL Statements

This chapter contains descriptions of the DQL statements. The description of each statement includes:

- Syntax
- Argument descriptions (if any)
- Return value (if appropriate)
- Detailed information about required permissions and usage
- Related statements
- Example

Quoting Object Type Names and Attribute Names: Documentum recommends that applications put double quotes around all object type names and attribute names. Doing that will ensure that the names won't conflict with any DQL reserved words or any words reserved by your underlying RDBMS. Documentum object type names and attribute names will not generate conflicts, but using the quotes in applications will make sure that no conflicts are generated by user-defined type or attribute names. To encourage this best practice, the DQL examples in our manuals use the double quotes.

Abort

Purpose

Cancels an explicit transaction.

Syntax

```
ABORT [TRAN[SACTION]]
```

General Notes

The ABORT statement terminates a transaction that was opened with the BEGIN TRAN statement. Any changes made while the transaction was open are rolled back. They are not saved to the Docbase.

The ALTER TYPE statement does not participate in transactions. If you alter a type as part of an explicit transaction and then issue an ABORT statement, the changes you made to the type are not reversed.

You can include either keyword, TRAN or TRANSACTION.

Related Statements

Begin Tran, page 58
Commit, page 62

Alter Group

Purpose

Modifies a user group.

Syntax

```
ALTER GROUP group_name ADD members
ALTER GROUP group_name DROP members
ALTER GROUP group_name SET ADDRESS email_address
ALTER GROUP group_name SET PRIVATE TRUE|FALSE
```

Syntax Description

Table 2–1. ALTER GROUP Syntax Description

Variable	Description
group_name	Identifies the group you want to modify. Use the group's name. The name can be a string, which must be enclosed in single quotes, or an identifier, which need not be in quotes.
members	Identifies users, groups, or both to add or drop from the group. You can specify user names representing individual users or names representing groups. Use a comma-separated list to specify multiple names. Alternatively, you can use a SELECT statement to identify the members (illustrated in Examples, page 42). When you are identifying an individual user, the name must be the value of the user's user_name attribute.
email_address	Defines an electronic mail address for a group. Specify this as a character string literal. You can use any email address that is valid for your environment. To remove a group's email address, specify email_address as an empty string.

Permissions

To alter a group, you must be either the group's owner or a user with Superuser user privileges.

General Notes

Each ALTER GROUP statement can perform only one type of operation. That is, you cannot add and drop members in the same statement. Nor can you set an email address in the same statement that adds or drops members.

The SET PRIVATE Clause

SET PRIVATE sets the is_private attribute for the group. Setting the attribute to TRUE makes the group a private group; setting it to FALSE makes the group a public group.

Related Statements

Create Group, page 63
Drop Group, page 88

Examples

The following example adds two users to the group called superusers:

```
ALTER GROUP superusers ADD steve,chip
```

The next example uses a SELECT statement to identify which users to add to the engineering group:

```
ALTER GROUP engineering ADD (SELECT "user_name"
FROM "dm_user" WHERE "user_os_name" LIKE '%eng%')
```

The final example defines an email address for the engineering group:

```
ALTER GROUP engineering
SET ADDRESS 'engineering@lion.stellar.com'
```

Alter Type

Purpose

Modifies an object type.

Syntax

```
ALTER TYPE type_name
[FOR POLICY policy_id STATE state_name]
type_modifier_list [PUBLISH]

ALTER TYPE type_name
[FOR POLICY policy_id STATE state_name]
MODIFY (attribute_modifier_clause)[PUBLISH]

ALTER TYPE type_name
ADD attribute_def {,attribute_def}[PUBLISH]

ALTER TYPE type_name
DROP attribute_name {,attribute_name}[PUBLISH]

ALTER TYPE type_name ADD_FTINDEX [ON] attribute_name
{,attribute_name}

ALTER TYPE type_name DROP_FTINDEX [ON] attribute_name
 {,attribute_name}
```

Syntax Description

Table 2-2. ALTER TYPE Syntax Description

Variable	Description
type_name	Identifies the type to alter. Use the name defined for the type when it was created.
policy_id	Identifies the default lifecycle for the type. Use the policy's object ID. Including the FOR POLICY clause requires at least Version permission on the lifecycle identified by *policy_id*.
state_name	Identifies one state in the default lifecycle. Use the state's name as defined in the dm_policy object.

Variable	Description
type_modifier_list	Lists one or more specifications that set or alter type data dictionary information for the type. Valid type modifiers are: *update_modifier* *mapping_table_specification* *constraint_specification* *component_specification* *type_drop_clause* Separate multiple type modifiers with commas. Refer to Type Modifiers, page 76 for information about the syntax and use of each specification.
attribute_modifier _clause	Defines the change you want to make to the attribute. The syntax for this clause is: *attribute_name domain* or *attribute_name (attribute_modifier_list)* *domain* is any valid DQL datatype. *attribute_modifier_list* lists one or more modifiers that set or alter data dictionary information for the attribute. Valid attribute modifiers are: *update_modifier value_assistance_specification* *mapping_table_specification default_specification* *constraint_specification attribute_drop_clause* Separate multiple modifiers with commas. Refer to Attribute Modifiers, page 71 for information about the syntax and use of each attribute modifier.

Variable	Description
attribute_def	Defines the attributes you want to add to the type. *attribute_def* has the following syntax: `attribute_name domain [REPEATING] (attribute_modifier_list)` *attribute_name* is the name of the attribute and *domain* is any valid DQL datatype. The keyword REPEATING defines the attribute as a repeating attribute. *attribute_modifier_list* lists one or more modifiers that define data dictionary information for the attribute. Refer to the preceding description of *attribute_modifier_clause* for a list of valid attribute modifiers. Separate multiple modifiers with commas.
attribute_name	For the DROP option, *attribute_name* identifies the attribute that you want to remove from the specified type. For the ADD_FTINDEX and DROP_FTINDEX options, *attribute_name* identifies an attribute that you want to add or drop from the full-text index. The attribute must be defined for the type specified. It cannot be an inherited attribute.

Permissions

To issue an ALTER TYPE statement that changes locale-specific data dictionary information, your session locale must match exactly one of the locales identified in the dd_locales attribute of the docbase config.

Table 2–3, page 45 lists the ALTER TYPE operations and describes who can perform them and on which types.

Table 2–3. Alter Type Operations

Alteration	Who Can Do It	To Which Types?
Set default ACL	Type owner or Superuser	All types
Add or drop an indexed attribute	Type owner or Superuser	All types
Set default storage area	Type owner or Superuser	All types
Set or drop data dictionary information	Type owner or Superuser	All types, with two exceptions. See * below table.
Add read/write attributes	Type owner or Superuser	User-defined only
Add read-only attributes	Superuser	User-defined only

Alteration	Who Can Do It	To Which Types?
Drop read/write attributes	Type owner or Superuser	User-defined only
Lengthen character string attributes	Type owner or Superuser	User-defined only

*It is not possible to add value assistance or constraints to a system-defined object type.

(For the definition of a read and write or read-only attribute, refer to Chapter 1, Object Basics, in the *Content Server Object Reference Manual*. For details about using indexed attributes, refer to Chapter 9, Full-Text Indexes, in the *Content Server Administrator's Guide*.)

General Notes

You cannot change the definition of a system-defined type. You can only set a variety of defaults and data dictionary information. You can change the definition of a user-defined type. (Refer to Table 2–3, page 45 for a summary of valid operations, the types on which they can be performed, and who can perform them.)

With one exception, you cannot issue an ALTER TYPE statement in an explicit transaction. The exception is adding or removing indexed attributes from an object type. An ALTER TYPE statement that only adds or removes an indexed attribute can be issued in an explicit transaction Any other ALTER TYPE operation is forbidden in an explicit transaction. If you do add or remove an indexed attribute in an explicit transaction, the changes are not reversed if you abort the transaction.

After you execute an ALTER TYPE statement, the changes are not reflected to users until the global type cache is updated. The updating is automatic but may take several minutes.

Localization

If you change data dictionary information that is locale-specific, such as label_text, it is changed only in the current locale (defined in the session's session_locale attribute).

If you change data dictionary information that isn't locale-specific, such as a constraint definition, it is changed in all locales.

PUBLISH

Including PUBLISH causes the server to publish the data dictionary information for the object type or attribute immediately. Including PUBLISH publishes the changes made in the ALTER TYPE statement and any other changes to the type or attribute that are not yet published.

If you don't include PUBLISH, the information is published the next time the Data Dictionary Publish job runs.

You can include PUBLISH with any ALTER TYPE statement except those that add or drop attributes from the list of indexed attributes.

It isn't necessary to include PUBLISH if you are dropping an attribute from the type definition. The data dictionary information for the attribute is automatically removed in such cases.

Type Modifiers

Type modifiers set or change some attributes of the type info object and data dictionary information for the type. For example, you can set the default storage area, ACL, or default lifecycle or add constraints to the type. You can also drop data dictionary information.

You cannot include a type modifier if the ALTER TYPE statement is executing inside an explicit transaction.

update_modifiers

Update modifiers set or remove attribute values in the dm_nls_dd_info, dm_dd_info, and dmi_type_info objects for the type. The dm_nls_dd_info and dm_dd_info objects hold data dictionary information for the type. The dmi_type_info object holds non-structural information about the type. You can set attributes in the dmi_type_info object that define the type's default ACL, default permissions, default storage area, and default group.

An update modifier is also used to define a default lifecycle for the type.

Setting or Removing Data Dictionary Values

You can set any attribute in the dm_nls_dd_info and dm_dd_info objects that are applicable to types. Use one of the following statement clauses:

```
SET attribute_name[[index]]=value
APPEND attribute_name=value
INSERT attribute_name[[index]]=value
REMOVE attribute_name[[index]]
TRUNCATE attribute_name[[index]]
```

The *attribute_name* is the name of an attribute defined for the dm_nls_dd_info or dm_dd_info object type. The nls_dd_info or dm_dd_info attribute must be applicable to object types and settable by users.

Include *index* if the attribute is a repeating attribute. Its interpretation varies:

- For a SET operation, the index defines which value to set.

  If the operation is adding a new value to the attribute, the index must identify the next available position in the list. If the SET operation is replacing an existing value, specify the index of the existing value.
- For an INSERT operation, the index defines where to insert the new value.

  Existing values are renumbered after the insertion.
- For a REMOVE operation, the index defines which value to remove.
- For a TRUNCATE operation, the index defines the starting position for the truncation.

  All values in the list, beginning at that position, are truncated.

The APPEND statement clause doesn't require an index value for repeating attributes because it automatically puts the new value at the end of the repeating attribute's value list.

You must enclose the index in square brackets.

value is a literal value. If you use a SET statement clause and the attribute is single-valued, *value* can be NULL.

You cannot use the SET, INSERT, or APPEND statement clauses to alter any of the following attributes:

- From dm_dd_info type

parent_id, default_value, cond_value_assist, cond_computed_expr, val_constraint, unique_keys, foreign_keys, primary_key

- From the dm_nls_dd_info type

  parent_id

Defining the Default ACL

A type's default ACL does not define access to the type. Users can assign a type's default ACL to any object of the type they create. Use the following syntax to set a type's default ACL:

```
ALTER TYPE type_name
SET DEFAULT ACL acl_name [IN acl_domain]
```

The value of *acl_name* is the ACL's object name. The *acl_domain* value is the name of its owner. The owner's name is either the user who created the ACL or the alias dm_dbo, representing the Docbase owner. The combination of the ACL name and domain uniquely identifies the ACL within the Docbase. (For more information about ACLs and their names and implementation, refer to Using ACLs, page 331 in the *Content Server Administrator's Guide* or Assigning an ACL, page 111 in *Content Server Fundamentals*.)

If either the name or domain includes a character (such as a space) that requires you to enclose the string in single quotes, then you must enclose both strings in single quotes.

If the default ACL is NULL, the server uses the default ACL defined for the type's supertype as the default. To set the default ACL to NULL, use the following syntax:

```
ALTER TYPE type_name SET DEFAULT ACL NULL
```

If you set the default ACL to NULL, the server automatically sets the default_owner_permit, default_group_permit, and default_world_permit attributes for the type to NULL also.

Defining the Default Storage Area

The default storage area for a type is where Content Server stores all content files associated with objects of that type. (Users can change the storage area for an individual object.)

To set the default storage area for a type, use the following syntax:

```
ALTER TYPE type_name SET DEFAULT STORAGE[=]storage_area_name
```

storage_area_name is the name of the storage object representing the storage area.

Defining the Default Group

To set the default group for a type, use the following syntax:

```
ALTER TYPE type_name SET DEFAULT GROUP[=]group_name
```

group_name can be a character string literal or an identifier. If it is a character string literal, you must enclose it in single quotes.

To set the default group to NULL, use:

```
ALTER TYPE type_name SET DEFAULT GROUP[=]NULL
```

Defining a Default Lifecycle

A lifecycle describes the life cycle of an object. Typically, the SysObject type and its subtypes have default lifecycles. If an object type has a default lifecycle defined for it, users or applications can attach the default simply by specifying the keyword Default in the Attach method. (An object is not attached to a default lifecycle automatically. Users or the application must explicitly issue an Attach method. Defining a default lets users or applications attach an object to the default without requiring them to know the name or object ID of the default. Also, it allows you to change the default without requiring you to rewrite and recompile any applications that reference the default.)

To set the default lifecycle for a type, use the following syntax:

```
ALTER TYPE type_name
SET DEFAULT BUSINESS POLICY[=]chronicle_id
[VERSION version_label]
```

chronicle_id is the object ID of the root version of the lifecycle. The VERSION clause identifies which version of the lifecycle you want to apply to the type. If you do not specify a version, the default is the CURRENT version.

mapping_table_specification

A mapping table specification is typically used to map a user-friendly character string value to an underlying value that may not be as readable or easy to understand. A mapping table can also be used to map localized or type-specific values to an underlying value.

For example, Desktop Client uses a mapping table defined at the type level to display a list of the display config objects appropriate for the object type. The actual names of the display config objects are mapped to more user-friendly strings. (Display config objects represent subsets of type attributes that have a common display definition.)

If you define a mapping table at the type level, the mappings apply to that object type and its subtypes.

constraint_specification

You can define the following constraints in a type modifier list:

- Primary key
- Unique key
- Foreign key
- Check

You must have Sysadmin or Superuser user privileges to create a foreign key constraint. You cannot add a constraint to a system-defined object type.

The syntax for adding a constraint to a type is:

```
ALTER TYPE type_name
[FOR POLICY policy_id STATE state_name]
ADD constraint_specification
```

You cannot include the FOR POLICY clause for primary, unique, or foreign constraints. The FOR POLICY clause can only be included when you are defining a check constraint. The clause defines the check constraint as applicable only when instances of the type are in the specified lifecycle state. You must have at least Version permission on the lifecycle identified by *policy_id*.

If you include the FOR POLICY clause, *type_name* must be the primary type for the lifecycle identified in *policy_id*. The *policy_id* is the object ID of the dm_policy object for the lifecycle. *state_name* is the name of the state for which you are defining the constraint. State names are defined in the lifecycle's dm_policy object.

To define a check constraint that applies to all states for an object, do not include the FOR POLICY clause. Any string literals in the check constraint must be ASCII characters.

For an explanation of each constraint specification, refer to constraint_specification, page 74.

component_specification

Component specifications identify which components can operate on objects of the type. Use the following syntax to identify a component for a type:

```
ALTER TYPE type_name
[FOR POLICY policy_id STATE state_name]
component_specification
```

Include the optional FOR POLICY clause to define the component as applicable only when objects of the type are in the specified state. You must have at least Version permission on the lifecycle identified by *policy_id* to include the FOR POLICY clause.

If you include the clause, *type_name* must be the primary type for the lifecycle identified in *policy_id*. *policy_id* is the object ID of the dm_policy object for the lifecycle. *state_name* is the name of the state for which you are defining the component. State names are defined in the lifecycle's dm_policy object.

The format of a valid component specification is described in component_specification, page 79.

type_drop_clause

A type drop clause removes constraint and component definitions defined for a type. You cannot remove inherited constraints and component definitions.

- To drop a primary key constraint, use:

  ```
  ALTER TYPE type_name
  DROP PRIMARY KEY
  ```

- To drop a unique key constraint, use:

  ```
  ALTER TYPE type_name
  DROP UNIQUE KEY [(attribute_list)]
  ```

  attribute_list identifies the type attributes that constitute the key. If there is more than one attribute, separate them with a comma.

- To drop a foreign key constraint, you must have Sysadmin or Superuser user privileges. Use the following syntax:

  ```
  ALTER TYPE type_name
  DROP FOREIGN KEY [(attribute_list)]
  REFERENCES type_name [(attr_list)]
  ```

  attribute_list identifies the type attributes that constitute the key. If there is more than one attribute, separate them with a comma. In the REFERENCES clause, *type_name* identifies the object type that contains the referenced attributes and *attr_list* names the referenced attributes.

- To drop check constraints, use:

  ```
  ALTER TYPE type_name
  [FOR POLICY policy_id STATE state_name]
  DROP CHECK
  ```

  DROP CHECK drops all check constraints defined for the type. If you include the FOR POLICY clause, the statement removes the check constraints defined for the specified state. You must have at least Version permission on the lifecycle identified by *policy_id* to include the FOR POLICY clause.

  Note: To drop a single check constraint, use an update modifier to remove or truncate the val_constraint[*x*], val_constraint_enf[*x*], and val_constraint_msg[*x*] attributes, where x is the index position of the check constraint you want to remove.

- To drop a component, use:

  ```
  ALTER TYPE type_name
  [FOR POLICY policy_id STATE state_name]
  DROP COMPONENTS
  ```

  DROP COMPONENTS drops all components defined for the type. If you include the FOR POLICY clause, the statement removes the components defined for the specified state. You must have at least Version permission on the lifecycle identified by *policy_id* to include the FOR POLICY clause.

Attribute Modifiers

Attribute modifiers set or drop data dictionary information for an attribute. Data dictionary information includes constraints, value assistance, default value definitions, and mapping information. Attribute modifiers set attributes of dm_nls_dd_info and dm_dd_info objects.

You cannot include an attribute modifier if the ALTER TYPE statement is executing in an explicit transaction.

update_modifiers

An update modifier lets you directly set an attribute in a dm_dd_info or nls_dd_info object. Use the following syntax to include an update modifier:

```
ALTER TYPE type_name
[FOR POLICY policy_id STATE state_name]
MODIFY (attribute_name (update_modifier))
```

Including the FOR POLICY clause makes the change to the attribute effective only when instances of the type are in the specified lifecycle state. You must have at least Version permission on the lifecycle identified by *policy_id*.

If you include the clause, *type_name* must be the primary type for the lifecycle identified in *policy_id*. The *policy_id* is the object ID of the dm_policy object for the lifecycle. The *state_name* is the name of the state for which you are defining the assistance. State names are defined in the lifecycle's dm_policy object.

For a description of valid update modifiers for an attribute, refer to update_modifier, page 71.

value_assistance_specification

A value assistance specification identifies one or more values for an attribute. Typically, value assistance is used to populate a pick list of values for the attribute when it appears as a field on a dialog box.

You cannot add value assistance to a system-defined object type.

Use the following syntax to add or change the value assistance specification for an attribute:

```
ALTER TYPE type_name
[FOR POLICY policy_id STATE state_name]
MODIFY (attribute_name (value_assistance_specification))
```

Including the FOR POLICY clause provides value assistance only when objects of the type are in the specified lifecycle state. You must have at least Version permission on the lifecycle identified by *policy_id*.

If you include the clause, *type_name* must be the primary type for the lifecycle identified in *policy_id*. The *policy_id* is the object ID of the dm_policy object for the lifecycle. The *state_name* is the name of the state for which you are defining the assistance. State names are defined in the lifecycle's dm_policy object.

For a description of valid value assistance specifications, refer to value_assistance_modifier, page 72.

mapping_table_specification

A mapping table specification typically maps a list of character string values to a list of integer values. This is useful when you want to provide users with an easily understood list of values for an attribute that is an integer data type. Use the following syntax to add or change a mapping table specification:

```
ALTER TYPE type_name
[FOR POLICY policy_id STATE state_name]
MODIFY (attribute_name (mapping_table_specification))
```

If you include the FOR POLICY clause, the mapped values are only available when instances of the type are in the specified lifecycle state. You must have at least Version permission on the lifecycle identified by *policy_id*.

When you include the clause, *type_name* must be the primary type for the lifecycle identified in *policy_id*. The *policy_id* is the object ID of the dm_policy object for the lifecycle. The *state_name* is the name of the state for which you are defining the mapping. State names are defined in the lifecycle's dm_policy object.

For a description of valid mapping table specifications, refer to mapping_table_specification, page 73.

default_specification

A default specification defines the default value for an attribute. When a new object of the type is created, the server automatically sets the attribute to the default value if no other value is specified by the user.

You cannot add an attribute and specify a default value for it in the same ALTER TYPE statement. You must use two ALTER TYPE statements: one to add the attribute and one to set its default value.

Use the following syntax to set or change a default specification:

```
ALTER TYPE type_name
MODIFY (attribute_name ([SET] default_specification))
```

The default value can be specified as:

- A literal value
- One of the following keywords: USER, NOW, TODAY, TOMORROW, YESTERDAY
- NULL (This option is not valid for repeating attributes.)

For a single-valued attribute, the default specification syntax is:

```
DEFAULT[=]default_value
```

For a repeating attribute, the default specification syntax is:

```
DEFAULT[=](default_value {,default_value})
```

For example, the following ALTER TYPE statement sets the default value for one single-valued attribute and one repeating attribute in the object type mytype:

```
ALTER TYPE "mytype"
MODIFY ("single_attr" (SET default=NOW)),
MODIFY ("rep_attr" (SET default=(USER)))
```

The default value must be appropriate for the datatype of the attribute. For example, you cannot specify the keyword USER for a date attribute. You cannot specify NULL for a repeating attribute.

constraint_specification

You can specify the following constraints in an attribute modifier list:

- Primary key
- Unique key
- Foreign key
- Not null
- Check

You must have Sysadmin or Superuser user privileges to create a foreign key constraint. You cannot add a constraint to a system-defined object type.

The syntax for adding a constraint to an attribute is:

```
ALTER TYPE type_name
[FOR POLICY policy_id STATE state_name]
MODIFY (attribute_name (ADD constraint_specification))
```

You cannot include the FOR POLICY clause for primary key, unique key, or foreign key constraints. The FOR POLICY clause can only be included when you are defining a NOT NULL or check constraint. The clause defines the constraint as applicable only when instances of the type are in the specified lifecycle state. To define a NOT NULL or check constraint that applies to all states for the object, do not include the FOR POLICY clause.

If you include the clause, you must have at least Version permission on the lifecycle identified by *policy_id*. The *type_name* must be the primary type for the lifecycle identified in *policy_id*. The *policy_id* is the object ID of the dm_policy object for the lifecycle. The *state_name* is the name of the state for which you are defining the constraint. State names are defined in the lifecycle's dm_policy object.

For an explanation of each constraint specification, refer to constraint_specification, page 74.

attribute_drop_clause

An attribute drop clause removes constraints, value assistance, or mapping information defined for the attribute. You cannot drop inherited constraints, value assistance, or mapping information.

- To drop a primary key constraint, use:

  ```
  ALTER TYPE type_name
  MODIFY (attribute_name (DROP PRIMARY KEY))
  ```

- To drop a unique key constraint, use:

  ```
  ALTER TYPE type_name
  MODIFY (attribute_name (DROP UNIQUE KEY
  [(attribute_name)]))
  ```

  attribute_name identifies the attribute you are removing from the unique key.

- To drop a foreign key constraint, you must have Sysadmin or Superuser user privileges. Use the following syntax:

  ```
  ALTER TYPE type_name
  MODIFY (attribute_name (DROP FOREIGN KEY
  [(attribute_name)] REFERENCES type_name [(attr_name)]))
  ```

  attribute_name identifies the attribute you are removing from the foreign key.

In the REFERENCES clause, *type_name* identifies the object type that contains the referenced attributes and *attr_name* identifies the referenced attribute. If *attr_name* is not included, r_object_id is the default.

- To drop check constraints, use:

```
ALTER TYPE type_name
[FOR POLICY policy_id STATE state_name]
DROP CHECK
```

DROP CHECK drops all check constraints defined for the type. If you include the FOR POLICY clause, the statement removes the check constraints defined for the specified state. You must have at least Version permission on the lifecycle identified by *policy_id*.

Note: To drop a single check constraint, use an update modifier to remove or truncate the val_constraint[*x*], val_constraint_enf[*x*], and val_constraint_msg[*x*] attributes, where x is the index position of the check constraint you want to remove.

- To drop value assistance, use the following syntax:

```
ALTER TYPE type_name
[FOR POLICY policy_id STATE state_name]
MODIFY (attribute_name (DROP VALUE ASSISTANCE))
```

- To drop mapping information, use the following syntax:

```
ALTER TYPE type_name
[FOR POLICY policy_id STATE state_name]
MODIFY (attribute_name (DROP MAPPING TABLE))
```

Modifying the Type Definition

You can modify the definition of a user-defined type by:

- Adding a new attribute
- Lengthening a character string attribute
- Dropping an attribute defined for the type

Adding a New Attribute

Use the following syntax to add a new attribute to a user-defined type:

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

attribute_def defines the new attribute and has the following syntax:

```
attribute_name domain [REPEATING]
(attribute_modifier_list)
```

The attribute name must following the naming rules outlined inObject Type and Attribute Names, page 27 in the *Content Server Object Reference Manual*.

domain is any valid DQL datatype. The keyword REPEATING defines the attribute as a repeating attribute. Valid attribute modifiers are described in Attribute Modifiers, page 71.

If the attribute is a string attribute and there are no instances of the type in the index, the attribute is marked for indexing.

If there are instances of the type in the index, the new attribute is not marked for indexing and a warning is issued. In such cases, you can use the ADD_FTINDEX clause to mark the attribute for indexing. Refer to Adding Indexed Attributes, page 55 for instructions.

Do not add more than 30 attributes in one ALTER TYPE statement. If the Docbase is running against DB2, the sum total of the lengths of all attributes defined for the type is constrained by the page size defined for the tablespace. Table 2–8, page 70 lists the tablespace page sizes and the corresponding maximum row length allowed.

Lengthening a Character String Attribute

The character string attribute must be defined for the object type. It cannot be an inherited attribute. The only valid change that you can make is to lengthen the attribute. The change is applied to all the type's subtypes and to all existing objects of the type and its subtypes.

For example, if a string attribute called employee_name is currently 24 characters, the following statement would change the length to 32 characters:

```
ALTER TYPE "test_result" MODIFY ("patient_name" char(32))
```

If you are running against Sybase, you can lengthen only one attribute in each execution of the statement.

Dropping Attributes from the Type

Only attributes that are defined for the type can be dropped. You cannot drop an inherited attribute. Dropping an attribute also removes the attribute from all subtypes of the type and removes all data dictionary information for the attribute.

The drop operation fails if the attribute is an indexed attribute for the type and there are existing indexed objects of the type or its subtypes. (An indexed attribute is an attribute whose value is stored in the full-text index.)

Note: Some relational database management systems (for example, DB2) don't support dropping columns from a table (an attribute is a column in a table in the underlying RDBMS). For those databases, if you drop an attribute, the corresponding column in the table representing the type is not actually removed. If you later try to add an attribute to that type that has the same name as the dropped attribute, you will receive an error message.

Adding Indexed Attributes

Use the ADD FTINDEX option to add attributes to the list of indexed attributes in these circumstances:

- To add an attribute that isn't a string attribute to the list of indexed attributes
- If an ALTER TYPE statement adding a string attribute resulted in a warning that the attribute wasn't marked for indexing.

When you add a string attribute to an object type, Content Server marks the attribute for indexing unless there are existing, instances of the type in the index. If there are instances of the type in the index, Content Server doesn't mark the attribute for indexing and issues a warning (recorded in the log file). In such circumstances, you can use ADD_FTINDEX to mark the attribute for indexing. However, you must first reset the index and must update the index after you mark the attribute.

Adding an indexed attribute succeeds only under the following conditions:

- The attribute whose value you want to include in the index must be defined for the type.
- The type must be dm_sysobject or one of its subtypes.
- There are no existing indexed objects of the type or its subtypes.

  If there are objects of the type or its subtypes already indexed, you must reset the index, issue the statement, and then update the index to add indexed attributes for the object type.

- The attribute should not contain values that have non-printing characters such as line feeds or carriage returns.

  The server will add attributes that contain non-printing characters to the indexed attributes but the indexing operations don't function properly if non-printing characters are in the attribute values.

For example, suppose a Docbase has a document subtype called report_form, which has an integer attribute called report_type. After using these forms for some time, the sales people ask you to add the report_type attribute to the index. Assuming that the existing reports are already indexed, to meet the request, you must:

- First, reset the index associated with the storage area in which reports are stored.
- Second, execute the following ALTER TYPE statment:

  ```
  ALTER TYPE "report_form" ADD_FTINDEX ON "report_type"
  ```

- Third, update the index.

When you add an indexed attribute to a type, the attribute's value is indexed for all objects of the type and any subtypes of the type.

Dropping Indexed Attributes

Dropping an attribute from the list of indexed attributes is one step operation if the index does not contain any indexed objects of the specified type or its subtypes. To drop the attribute, just execute the ALTER TYPE statement with the DROP_FTINDEX clause. For example:

```
ALTER TYPE "report_template" DROP_FTINDEX ON "writer_name"
```

If there are existing indexed objects of the type or its subtypes, you must first reset the index, then issue the ALTER TYPE statement that drops the indexed attribute, and then update the index.

For example, suppose you want to drop an indexed string attribute called report_type, which is defined for a subtype called report_form. To do this:

- First, reset the index associated with the storage area in which report objects are stored
- Second, issue the following statement:

  ```
  ALTER TYPE "report_from" DROP_FTINDEX ON "report_type"
  ```

- Third, update the index.

When you drop an indexed attribute from the index, the attribute's value is removed from the index for all objects of the type and any subtypes of the type.

Related Statements

Create Type, page 68
Drop Type, page 89

Examples

This example sets the owner's object-level permission and the default group for the user-defined type called legal_document:

```
ALTER TYPE "legal_document"
SET OWNER PERMIT browse,
DEFAULT GROUP lawyers
```

This example adds an attribute to the user-defined type called report_doc:

```
ALTER TYPE "report_doc"
ADD "monthly_total" integer
```

The next example changes the length of the client_comments attribute in the user-defined type called case_report:

```
ALTER TYPE "case_report"
MODIFY ("client_comments" string(255))
```

This next example sets the default ACL for the document object type:

```
ALTER TYPE "dm_document"
SET DEFAULT ACL generic_doc IN dm_dbo
```

This final example demonstrates the use of single quotes in setting the ACL:

```
ALTER TYPE "dm_document" SET DEFAULT ACL 'mktg default'
IN 'marketing'
```

Begin Tran

Purpose

Opens an explicit transaction.

Syntax

```
BEGIN TRAN[SACTION]
```

Permissions

Anyone can execute the BEGIN TRAN statement.

General Notes

The BEGIN TRAN or BEGIN TRANSACTION statement opens an explicit transaction. When you are working in an explicit transaction, none of the changes you make to files or attribute values in objects are saved until you issue a COMMIT statement to commit the changes.

Related Statements

Abort, page 40
Commit, page 62

Change...Object

Purpose

Changes the object type of one or more objects.

Syntax

```
CHANGE current_type OBJECT[S]
TO new_type[update_list]
[IN ASSEMBLY document_id [VERSION version_label] [DESCEND]]
[SEARCH fulltext search condition]
[WHERE qualification]
```

Syntax Description

Table 2–4. CHANGE...OBJECT Syntax Description

Variable	Description
current_type	Identifies the object type of the objects to change. Use the type's internal name (for example, dm_document). The type must be a SysObject or SysObject subtype (system- or user-defined).
new_type	Specifies the new object type for the objects. Use the type's internal name. Valid new types depend on the object's current type. Refer to Rules and Constraints on Changing Types, page 60 for a complete description.

update_list	Specifies one or more change operations to perform on the objects you are changing.Valid formats are:
	`set attribute_name = value set`
	`attribute_name[[index]] = value`
	`append [n] attribute_name = value`
	`insert attribute_name[[index]] = value`
	`remove attribute_name[[index]] truncate`
	`attribute_name[[index]]`
	If you specify more than one operation, use commas to separate the clauses. Refer to The Update Operations, page 144 for details of these options.

IN ASSEMBLY clause	Limits the statement to those objects that belong to a particular assembly. For details about the syntax and use, refer to The IN ASSEMBLY Clause, page 127.

Variable	Description
SEARCH clause	Restricts the candidates for change to those that satisfy the full-text search condition. Refer to The SEARCH Clause, page 128 for a description of its syntax and use.
WHERE clause	Defines a qualification used to restrict what objects are changed. Refer to The WHERE Clause, page 130 for a description of a valid qualification.

Return Value

The CHANGE...OBJECT statement returns a collection identifier. The collection has one query result object, with one attribute called objects_changed. The attribute contains the number of objects changed.

Permissions

To use this statement, you must have Delete permission for the objects you want to change.

To use the update_list option to change an object's owner (owner_name attribute), you must be have Superuser user privileges or Change Ownership privilege.

General Notes

The CHANGE...OBJECT[S] statement lets you change one or more objects from one type to another.

Note that when you change an object from a subtype to a supertype, you lose the values of the attributes that are inherited from the supertype.

Objects that meet the criteria in the statement are deleted from the Docbase and recreated as objects of the specified new type. The recreated objects retain their old object IDs and all their previous relationships. For example, if a document is annotated, changing that document to another type of document does not delete its annotations.

You cannot execute CHANGE...OBJECT on an object that is immutable.

Rules and Constraints on Changing Types

There are some constraints when you change an object's type:

- The object's current type and the new type must have the same type identifier, and they must be SysObjects or SysObject subtypes.
- The type identifier is the first two characters in an object ID. For example, all documents have 09 as their type identifier.
- The old and new types cannot be at the same level in the type hierarchy.

  You can move objects up or down in the object hierarchy but not laterally. For example, the dm_document and dm_process object types are peers—they are at the same level because both are direct subtypes of dm_sysobject. Now, suppose you have two dm_document subtypes, proposal_doc and report_doc. Using

CHANGE...OBJECT, you can change a proposal_doc or report_doc object to a dm_document object or a dm_document object to a proposal_doc or report_doc object, but you cannot change any of the document types or subtypes to dm_process objects.

When you change an object to a type that is higher in the hierarchy, the server drops any attributes from the old type that are not in the new type. Similarly, when you move an object to a type that is lower in the hierarchy, the server adds any attributes in the new type that were not in the old type. Unless you use the update_list option to set these new attributes, the server assigns them default values appropriate for their datatypes.

The optional clauses are applied in the following order:

- The SEARCH clause
- The IN ASSEMBLY clause
- The WHERE clause

Related Statements

Create...Object, page 65
Delete...Object, page 84
Update...Object, page 141

Example

This example changes all documents for which the subject is new book proposal to the document subtype book_proposal:

```
CHANGE "dm_document" OBJECTS TO "book_proposal"
SET "department" = 'marketing'
WHERE "subject" = 'new book proposal'
```

Commit

Purpose

Commits the changes made during an explicit transaction and closes the transaction.

Syntax

```
COMMIT [TRAN[SACTION]]
```

Permissions

Anyone can execute the COMMIT statement.

General Notes

The COMMIT statement closes a transaction that was opened with the BEGIN TRAN statement and commits to the Docbase any changes made to objects or files during the transaction.

You can include either TRAN or TRANSACTION.

Related Statements

Abort, page 40
Begin Tran, page 58

Create Group

Purpose

Creates a user group.

Syntax

```
CREATE [PUBLIC|PRIVATE] GROUP group_name
[WITH][ADDRESS email_address] [MEMBERS members]
```

Syntax Description

Table 2–5. CREATE GROUP Syntax Description

Variable	Description
group_name	Defines the name of the group that you are creating. This name must be unique among all group names and user names in the Docbase. You can specify the name as an identifier or as a character string literal. **Note:** Content Server stores all group names in lowercase.
email_address	Specifies the electronic mail address of the group. Use a character string literal. You can specify any email address that is valid for your environment.
members	Specifies users to be included in the group. You can specify user names representing individual users, names representing other groups, or both. The names must appear as a comma-separated list. Alternatively, you can use a SELECT statement to populate the group. When you are specifying an individual user, the name must be the value of the user's user_name attribute.

Return Value

The CREATE GROUP statement returns a collection whose result object has one attribute, new_object_id, which contains the object ID of the new group.

Permissions

You must have Create Group, Sysadmin, or Superuser user privileges to create a group.

General Notes

When you create a group, you are its owner and can alter or delete the group.

Public and Private Groups

The PUBLIC keyword creates the group as a public group. The PRIVATE keyword creates the group as a private group. When a user with Sysadmin or Superuser user privileges creates a group, the group is public by default. When a user with Create Group user privileges creates a group, the group is private by default.

The public or private setting is stored in the is_private attribute for the group. The setting is not used by Content Server. All groups, public or private, are visible in the dm_group type. This attribute is provided for use by user-written applications.

Related Statements

Alter Group, page 41
Drop Group, page 88

Examples

The following example creates a group called supers whose members are all the users with the Superuser user privilege:

```
CREATE GROUP supers MEMBERS
(SELECT "user_name" FROM "dm_user"
 WHERE "user_privileges" >= 16)
```

The next example creates a group that includes all of Ron's co-workers but not Ron:

```
CREATE GROUP rons_baby_gift WITH MEMBERS
(SELECT "user_name" FROM "dm_user"
 WHERE "user_name" != 'Ron')
```

This final example creates a group and defines an email address for that group:

```
CREATE GROUP client_group
WITH ADDRESS 'john@jaguar.docu.com'
MEMBERS john,regina,kendall,maria
```

Create...Object

Purpose

Creates an object.

Syntax

```
CREATE type_name OBJECT update_list
[,SETFILE 'filepath' WITH CONTENT_FORMAT='format_name']
{,SETFILE 'filepath' WITH PAGE_NO=page_number}
```

Syntax Description

Table 2–6. CREATE...OBJECT Syntax Description

Variable	Description
type_name	Identifies the type of object to create. Specify the name of the object type. You can use any valid type in the Docbase.
update_list	Specifies one or more operations you want to perform on the new object. Valid formats are: `set attribute_name = value set attribute_name[[index]] = value append [n]attribute_name = value insert attribute_name[[index]] = value remove attribute_name[[index]] truncate attribute_name[[index]] [un]link 'folder path' move [to] 'folder path'` If you include multiple clauses in the update list, use commas to separate the clauses.
SETFILE clause WITH CONTENT_ FORMAT option	Adds the first content file to the new object. Refer to WITH CONTENT_FORMAT Option, page 66 for details.
SETFILE clause WITH PAGE_NO option	Adds additional content to the object. Refer to WITH PAGE_NO Option, page 66 for details.

Return Value

The CREATE...OBJECT statement returns a collection whose result object has one attribute, object_created, which contains the object ID of the new object.

Permissions

Anyone can issue the CREATE...OBJECT statement. However, you must have Superuser privileges to include the SETFILE clause.

General Notes

The CREATE...OBJECT statement creates and saves a new object. As part of the process, you can set some of the object's attributes, specify a storage location for the object, and associate one or more content files with the object.

If you do not use the link update option to define a storage location, the new object is stored in your default folder or cabinet.

The SETFILE Clauses

A SETFILE clause adds content to the new object. You must have Superuser privileges to include the clause in the statement.

You cannot store the content file you add in a turbo store storage area.

Any object capable of having content may have multiple associated content files. All files must have the same content format and be ordered within the object.

The content format is defined when you add the first content file to the object. All subsequent additions must have the same format. Consequently, specifying the format for content additions after the first file is not necessary. Instead, you must specify the content's position in the ordered list of content files for the object.

To add the first content, use the SETFILE clause with the WITH CONTENT_FORMAT option.

To add additional content, use the SETFILE clause with the PAGE_NO option.

You can't include both options in a single SETFILE clause.

WITH CONTENT_FORMAT Option

Use this SETFILE option to add the first content file to a new object. The syntax is:

```
SETFILE 'filepath' WITH CONTENT_FORMAT='format_name'
```

The filepath must identify a location that is visible to Content Server.

The format name is the name found in the name attribute of the format's dm_format object.

WITH PAGE_NO Option

Use this SETFILE option to add additional content to a new object. The syntax is:

```
SETFILE 'filepath' WITH PAGE_NO=page_number
```

The filepath must identify a location that is visible to Content Server.

The page number identifies the file's position of the content file within the ordered contents of the new object. You must add content files in sequence. For example, you cannot add two files and specify their page numbers as 1 and 3, skipping 2. Because the first content file has a page number of 0, page numbers for subsequent additions begin with 1 and increment by 1 with each addition.

You cannot use the SETFILE clause with the PAGE_NO option in a CREATE...OBJECT statement unless the statement contains a prior SETFILE clause with the CONTENT_FORMAT option.

Related Statements

Change...Object, page 59

Delete...Object, page 84
Update...Object, page 141

Examples

The following example creates a new document and sets its title and subject:

```
CREATE "dm_document" OBJECT
SET "title" = 'Grant Proposal',
SET "subject" = 'Research Funding'
```

This example creates a new document, sets its title, and adds two content files. The example is shown for both Windows and UNIX platforms.

On Windows:

```
CREATE "dm_document" OBJECT
SET "title" = 'Grant Proposal',
SETFILE 'c:\proposals\grantreq.doc'
WITH CONTENT_FORMAT='msww',
SETFILE 'c:\proposals\budget.doc' WITH PAGE_NO=1
```

On UNIX:

```
CREATE "dm_document" OBJECT
SET "title" = 'Grant Proposal',
SETFILE 'u12/proposals/grantreq.doc
WITH CONTENT_FORMAT='msww',
SETFILE 'u12/proposals/budget.doc' WITH PAGE_NO=1
```

Create Type

Purpose

Creates an object type.

Syntax

```
CREATE TYPE type_name
[(attribute_def {,attribute_def})]
[WITH] SUPERTYPE parent_type
[type_modifier_list] [PUBLISH]
```

Syntax Description

Table 2–7. CREATE TYPE Syntax Description

Variable	Description
type_name	Names the new type. Use any valid name that is unique among the other user-defined type names in the Docbase. Types with names beginning with dm can only be created by a user with Superuser user privileges. The type name must consist of ASCII characters.

attribute_def	Defines an attribute for the new type. You can define up to 30 attributes. Each attribute definition has the following format:
	
	`attribute_name domain [REPEATING]`
	`[(attribute_modifier_list)]`
	
	attribute_name names the attribute and *domain* defines its datatype. The attribute name must consist of ASCII characters. The domain can be any valid DQL datatype. If the datatype is a character string datatype, the *domain* specification must also include the length.
	
	REPEATING defines the attribute as a repeating attribute.

Variable	Description
attribute_def continued	*attribute_modifier_list* defines data dictionary information for the attribute. Valid attribute modifiers are: *update_modifier* *value_assistance_specification* *mapping_table_specification* *default_specification* *constraint_specification* Separate multiple modifiers with commas. You cannot include an attribute modifier if the statement is inside an explicit transaction. Refer to Attribute Modifiers, page 71 for descriptions of the attribute modifiers.
parent_type	Identifies the supertype of the new type. Valid supertypes are: dm_sysobject and its subtypes dm_user and its subtypes dm_relation dm_state_extension dm_state_type dm_email_message user-defined types To create a type with no supertype, specify *parent_type* as NULL. This requires the Superuser user privilege.
type_modifier_list	Defines data dictionary information for the type. Valid type modifiers are: *update_modifier* *mapping table specification* *constraint_specification* *component_specification* Separate multiple modifiers with commas. You cannot include a type modifier if the statement is inside an explicit transaction. Refer to Type Modifiers, page 76 for descriptions of the type modifiers.

Return Value

The CREATE TYPE statement returns a collection whose result object has one attribute, new_object_ID, which contains the object ID of the new object type.

Permissions

You must have Create Type, Sysadmin, or Superuser privileges to create a new object type.

To define read-only attributes for a new type (attributes whose names begin with r_) or to create a type that has no supertype, you must have Superuser privileges.

If the statement sets locale-specific information for the new type or attributes, your session locale must match exactly one of the locales defined in the dd_locales attribute of the docbase config object.

General Notes

The user who creates a type becomes the type's owner.

You cannot include a CREATE TYPE statement in an explicit transaction.

All string attributes defined for the type are automatically marked for indexing. If you don't want a string attribute marked for indexing, use the ALTER TYPE statement after you create the type to remove the attribute from the list of indexed attributes. Refer to Dropping Indexed Attributes, page 56 for instructions.

Do not define more than 30 attributes in a single CREATE TYPE statement. If the type has more than 30 attributes, use ALTER TYPE to add the additional attributes. The total number of attributes in the new type cannot exceed the supported number of columns in a table in the underlying RDBMS. On DB2, the sum total of the lengths of the attributes defined for the type cannot exceed the maximum row length set by the page size of the tablespace. Table 2–8, page 70 lists the tablespace page sizes and the corresponding maximum row length allowed.

Table 2–8. DB2 Tablespace Page Sizes and Associated Maximum Row Lengths

Tablespace Page Size	Maximum Row Length, in bytes
4K	4005
8K	8101
16K	16,293
32K	32,677

Additionally, be sure to follow the naming rules for attributes described in Object Type and Attribute Names, page 27 in the *Content Server Object Reference Manual*.

Localization

When you create a new type, any data dictionary information that you define for the type or its attributes in the statement is published in all locales identified in the dd_locales attribute of the docbase config object.

PUBLISH

Including PUBLISH causes the server to publish the data dictionary information for the object type immediately. Publishing creates the dd type info , dd attr info, and dd common info objects that store the published data dictionary information for the object type.

If you don’t include PUBLISH, the information is published the next time the Data Dictionary Publish job runs.

Attribute Modifiers

Attribute modifiers define data dictionary information for an attribute. Defining attribute modifiers sets attributes in data dictionary-related objects for the type. It does not set any dm_type or dm_type_info attributes for the type.

update_modifier

An *update_modifier* specification can be:

```
SET attribute_name[[index]]=value
APPEND attribute_name=value
INSERT attribute_name[[index]]=value
REMOVE attribute_name[[index]]
TRUNCATE attribute_name[[index]]
```

attribute_name is the name of an attribute defined for the dm_nls_dd_info or dm_dd_info object type. The dm_nls_dd_info or dm_dd_info attribute must be applicable to object type attributes and settable by users. If the attribute is a dm_nls_dd_info attribute, only the dm_nls_dd_info object specific to the current locale is updated.

Include *index* if the attribute is a repeating attribute. Its meaning varies:

- For a SET operation, the index defines which value to set.

  If the operation is adding a new value to the attribute, the index must identify the next available position in the list. If the SET operation is replacing an existing value, specify the index of the existing value.
- For an INSERT operation, the index defines where to insert the new value.

  Existing values are renumbered after the insertion.
- For a REMOVE operation, the index defines which value to remove.
- For a TRUNCATE operation, the index defines the starting position for the truncation.

  All values in the list, beginning at that position, are truncated.

The APPEND statement clause doesn’t require an index value for repeating attributes because it automatically puts the new value at the end of the repeating attribute’s value list.

You must enclose the index in square brackets.

value is a literal value. If you use a SET statement clause and the attribute is single-valued, *value* can be NULL.

You cannot use the SET, INSERT, or APPEND formats against the following attributes:

- From dm_dd_info type

  parent_id, default_value, cond_value_assist, cond_computed_expr, val_constraint, unique_keys, foreign_keys, primary_key
- From the dm_nls_dd_info type

  parent_id

value_assistance_modifier

A value assistance modifier identifies one or more values for the attribute. Typically, value assistance is used to populate a pick list of values for the attribute when it appears as a field on a dialog box. A value assistance modifier has the following format:

```
VALUE ASSISTANCE IS
[IF (expression_string)
va_clause
(ELSEIF (expression_string)
va_clause)
ELSE]
va_clause
[DEPENDENCY LIST ([attribute_list])
```

You can include multiple ELSEIF clauses.

expression_string

expression_string defines a Boolean condition. If it returns TRUE, the server executes the associated *va_clause* to return values for the attribute. *expression_string* can be an expression or a complete user-defined routine. It must be written in Docbasic and must return a Boolean value. Any string literals in the expression or routing must consist of ASCII characters. You must have Sysadmin or Superuser user privileges to provide a user-defined routine for *expression_string*.

For expressions, the syntax is:

```
expression [LANGUAGE docbasic]
```

For user-defined routines, the syntax is:

```
routine_name
([routine_parameter {,routine_parameter}])
FROM OBJECT(object_id)
[LANGUAGE docbasic]
```

routine_parameter is an attribute name.

The FROM clause must identify an object whose content file contains the source code for the routine.

va_clause

The *va_clause* provides the values for the attribute. It has three possible formats. One format provides a list of literal values. The second defines a query to return the values. The third defines a user-written routine to return the values. You must have Sysadmin or Superuser user privileges to specify a user-written routine for value assistance.

Each format allows you to provide an estimate of the expected number of attribute values and indicate whether the values represent the complete list of allowed values for the attribute.

- To provide a list of literal values, use the syntax:

  ```
  LIST (literal_list)
  [VALUE ESTIMATE=number]
  [IS [NOT] COMPLETE]
  ```

- To provide values from a query, use the syntax:

  ```
  QRY 'query_string'
  [QRY ATTR = attribute_name]
  [ALLOW CACHING]
  [VALUE ESTIMATE=number]
  [IS [NOT] COMPLETE]
  ```

  query_string is a DQL query. You can use the special token, $$, in the query string. When the query is executed, a single dollar sign will appear in the query in place of the token.

  The QRY ATTR clause defines which of the attributes in the selected values list to display. Typically, *query_string* has only one attribute in its selected values list—the attribute for which you are providing the value assistance. However, some situations may require you to put more than one attribute in the selected values list. In such cases, the server assumes that the first attribute selected is the attribute for which you are providing value assistance. If this is not the case, you must include the QRY ATTR clause to define which attribute is the attribute for which you are providing assistance.

  The ALLOW CACHING clause permits clients to cache the query results.

- To provide values from a user-written routine, you must have at least SYSADMIN privileges. Use the syntax:

  ```
  FUNCTION(routine_name
  ([routine_parameter {,routine_parameter}])
  FROM OBJECT (object_id)
  [LANGUAGE docbasic])
  [SEPARATOR = character]
  [VALUE ESTIMATE=number]
  [IS [NOT] COMPLETE]
  ```

  routine_parameter is an attribute name.

  The FROM clause must identify an object whose content file contains the source code for the routine.

  The SEPARATOR clause defines a character to use as a separator between the returned values. The default is a comma.

dependency clause

The DEPENDENCY LIST clause identifies the attributes on which *expression_string* depends. The default is all attributes named in the *expression_strings*.

mapping_table_specification

A mapping table specification most commonly maps a list of descriptive character strings to a corresponding list of integers. For example, assume an object type has an integer attribute called country in which each value represents a different country. If you display that attribute in a dialog box, you want users to see a name for each country rather than an integer value. Using a mapping table specification, you can map the integer values to corresponding country names.

The syntax is:

```
MAPPING TABLE (map_element {,map_element})
```

where *map_element* is:

```
VALUE=value_string
[DISPLAY=display_string]
```

```
[COMMENT=description_string]
```

For example:

```
MAPPING TABLE (VALUE=4
DISPLAY=Spain
COMMENT='Added to list in first quarter 98')
```

The default for DISPLAY is the data value as a character string literal. The default for COMMENT is a NULL string.

default_specification

A default specification provides a default value for an attribute. When a user creates a new instance of the type, Content Server automatically sets the attribute to the default value if no other value is specified by the user.

The default value can be specified as:

- A literal value
- One of the following keywords: USER, NOW, TODAY, TOMORROW, YESTERDAY
- NULL (This option is not valid for repeating attributes.)

For a single-valued attribute, the syntax is:

```
DEFAULT[=]default_value
```

For a repeating attribute, the syntax is:

```
DEFAULT[=](default_value {,default_value})
```

If you specify the default value with a keyword that indicates a date or time, the keyword is specified using the DATE function:

```
DATE(keyword)
```

For example, the following statement creates object type mytype with one single-valued date attribute and one repeating string attribute and defines a default value for each:

```
CREATE TYPE "mytype" ("single_attr" date (default=date(NOW)),
"rep_attr" string(32) repeating (default=(USER)))
WITH SUPERTYPE "dm_document"
```

The default value must be appropriate for the datatype of the attribute. For example, you cannot specify the keyword USER for a date attribute. You cannot specify NULL for a repeating attribute.

constraint_specification

A *constraint_specification* defines constraints for an attribute. For example, you can define a check constraint to provide data validation for the attribute. Or you can make the attribute a unique key for the type. Constraints are not enforced by Content Server. If you define a constraint for an attribute, it must be enforced by your client application.

You can define five kinds of constraints in an attribute modifier list:

- Primary key
- Unique key
- Foreign key
- Not null
- Check

If you define a primary key, unique key, foreign key, or check constraint in an attribute modifier list, only the single attribute for which the constraint is defined can be part of the constraint. If the constraint must include multiple attributes, define it in the type modifier list.

For each constraint, you can define an error message to display if the constraint is violated. You can also indicate whether the constraint is enforced by the application or disabled. Enforcement information is stored in the data dictionary but not used by Content Server.

Any constraint you define is inherited by all the type's subtypes.

Refer to Constraints, page 53 in *Content Server Fundamentals* for expanded information about the constraints and what they do.

Primary Key

You can define a primary key constraint only on single-valued attributes. The attribute must also have a NOT NULL constraint. You can define only one primary key for a type. (A type may have more than one primary key if it inherits a primary key from its supertype.)

To define a primary key constraint in an attribute modifier list, use:

```
PRIMARY KEY
[REPORT 'message_string '[ON VIOLATION]]
[DISABLE|ENFORCE [BY] APPLICATION]
```

message_string is a string literal that contains the error message you want displayed if the constraint is violated. You can include the following special tokens in the message to parameterize it:

- $value(*attribute_name*)

  When the message is displayed, $value(*attribute_name*) is replaced with the value entered by the user in the field associated with the attribute. The attribute name must be enclosed in parentheses with no spaces before the opening parenthesis. For example:

  ```
  The security type must be folder" or cabinet and you
  entered $value(security_val).
  ```

- $$

  Use $$ in a message string to display a single dollar sign in the message.

The DISABLE|ENFORCE clause lets you specify that the constraint is either disabled or enforced by the client application. Content Server does not enforce constraints.

Unique Key

The syntax for a unique key constraint specification in an attribute modifier list is:

```
UNIQUE KEY
[REPORT 'message_string '[ON VIOLATION]]
[DISABLE|ENFORCE [BY] APPLICATION]
```

Refer to the preceding description of a primary key constraint for information about the error message and enforcement clauses.

Foreign Key

You must have Sysadmin or Superuser user privileges to create a foreign key. To define a foreign key constraint in an attribute modifier list, use:

```
FOREIGN KEY REFERENCES type_name [(attribute_name)]
[REPORT 'message_string '[ON VIOLATION]]
[DISABLE|ENFORCE [BY] APPLICATION]
```

type_name identifies the foreign object type that contains the referenced attribute. *attribute_name* identifies the referenced attribute in *type_name*. The referenced attribute must have the same datatype as the attribute for which you are defining the constraint.

If *attribute_name* is not included, the server assumes that the referenced attribute is r_object_id. In this case, the attribute for which you are defining the foreign key must have an ID datatype.

Refer to Primary Key, page 75 for information about the error message and enforcement clauses.

Not Null

A NOT NULL constraint means that the attribute can't have a NULL value. To define a NOT NULL constraint specification, use:

```
NOT NULL
[REPORT 'message_string '[ON VIOLATION]]
[DISABLE|ENFORCE [BY] APPLICATION]
```

Refer to Primary Key, page 75 for information about the error message and enforcement clauses.

Check

Check constraints are used most commonly for data validation. The constraint consists of an expression or routine that must return TRUE. If it returns FALSE, the attribute's value violates the constraint. To define a check constraint, use:

```
CHECK(expression_string)
```

expression_string can be an expression or a complete user-defined routine. The expression or routine must be written in Docbasic and return a Boolean value. Any string literals in the expression or routine must consist of ASCII characters. To specify a routine, you must have Sysadmin or Superuser user privileges.

For expressions, the syntax is:

```
expression [LANGUAGE docbasic]
[REPORT 'message_string' [ON VIOLATION]]
```

For user-written routines, the syntax is:

```
routine_name
([routine_parameter {,routine_parameter}])
FROM OBJECT(object_id |
PATH folder_path[VERSION version_label])
[LANGUAGE docbasic]
[REPORT 'message_string' [ON VIOLATION]]
```

routine_parameter is an attribute name.

The FROM clause must identify an object whose content file contains the source code for the routine. Identify the object using the object's ID or a folder path for an object. If you use a folder path, you can include a version label. If you don't include a version label, the label CURRENT is assumed.

Refer to Primary Key, page 75 for information about the error message clause.

Type Modifiers

Type modifiers define data dictionary information for the type. For example, you can set the type's default lifecycle or define constraints for the type. Type modifier set attributes in the data dictionary objects for the type. No attributes are set in the dm_type or dm_type_info objects for the type.

update_modifier

You can use the same update modifiers for types as for attributes (refer to Attribute Modifiers, page 71). In addition, you can use the following to set the default lifecycle for a type:

```
SET DEFAULT BUSINESS POLICY[=]
chronicle_id [VERSION version_label]|
NULL|
NONE
```

chronicle_id is the object ID of the root version of the lifecycle. The VERSION clause identifies which version of the lifecycle you want to use. The default is the CURRENT version.

If you set the default lifecycle to NULL, the type inherits the default lifecycle from its supertype.

If you set the default lifecycle to NONE, the type has no default lifecycle.

mapping_table_specification

A mapping table specification is typically used to map a user-friendly character string value to an underlying value that may not be as readable or easy to understand. A mapping table can also be used to map localized or type-specific values to an underlying value.

For example, Desktop Client uses a mapping table defined at the type level to display a list of the display config objects appropriate for the object type. The actual names of the display config objects are mapped to more user-friendly strings. (Display config objects represent subsets of type attributes that have a common display definition.)

If you define a mapping table at the type level, the mappings apply to that object type and its subtypes.

constraint_specification

You can define four kinds of constraints in a type modifier list:

- Primary key
- Unique key
- Foreign key
- Check

For each constraint, you can define an error message to display if the constraint is violated. You can also indicate whether the constraint is enforced by the application or disabled. Enforcement information is stored in the data dictionary but not used by the Content Server.

Constraints are defined in a type modifier list if the constraint includes multiple attributes. For example, if two attributes make up the primary key for a type, the primary key constraint must be defined in the type modifier list. If only a single attribute defines the constraint, the constraint is typically defined in the attribute modifier list for the attribute rather than the type modifier list.

Refer to Constraints, page 53 in *Content Server Fundamentals* for more information about the constraints and their use.

Primary Key

You can define only one primary key for a type. (A type may have more than one primary key if it inherits a primary key from its supertype.) To define a primary key constraint in a type modifier list, use:

```
PRIMARY KEY (attribute_list)
[REPORT 'message_string '[ON VIOLATION]]
[DISABLE|ENFORCE [BY] APPLICATION]
```

attribute_list is a comma-separated list of the attributes that make up the key. The attributes must be single-valued attributes and each must have a NOT NULL constraint. Additionally, all the attributes must be defined for the same object type.

message_string is a string literal that contains the error message you want displayed if the constraint is violated. You can include two special tokens in the message to parameterize it:

- $value(*attribute_name*)

  When the message is displayed, the server replaces $value(*attribute_name*) with the value entered by the user in the field associated with the attribute. The attribute name must be enclosed in parentheses with no space before the opening parenthesis. For example:

  ```
  The security type must be folder or cabinet and you
  entered $value(security_val).
  ```

- $$

  Use $$ in a message string to display a single dollar sign in the message.

The DISABLE|ENFORCE clause lets you specify that the constraint is either disabled or enforced by the client application. Content Server does not enforce constraints.

Unique Key

To define a unique key constraint, use:

```
UNIQUE KEY (attribute_list)
[REPORT 'message_string '[ON VIOLATION]]
[DISABLE|ENFORCE [BY] APPLICATION]
```

attribute_list is a comma-separated list of the attributes that make up the key. The attributes can be either single-valued or repeating attributes, but all must be defined for the same object type.

Refer to the description of a primary key constraint on Primary Key, page 77 for information about the error message and enforcement clauses.

Foreign Key

You must have Sysadmin or Superuser user privileges to define a foreign key constraint. Use the following syntax:

```
FOREIGN KEY (attribute_list)
REFERENCES type_name [(attr_list)]
[REPORT 'message_string'[ON VIOLATION]]
[DISABLE|ENFORCE [BY] APPLICATION]
```

attribute_list is a comma-separated list of the attributes that make up the key.

type_name identifies the foreign object type that contains the referenced attributes and *attr_list* identifies the attributes in the foreign type that are referenced by those in the foreign key. The attributes defined in *attribute_list* and *attr_list* must match in number and datatypes.

If an *attr_list* is not included, the server assumes that the referenced attribute is r_object_id. In this case, only one attribute can be specified in *attribute_list* and it must have an ID datatype.

Refer to the description of a primary key constraint on Primary Key, page 77 for information about the optional error message and enforcement clauses.

Check

Check constraints are used most commonly for data validation. The constraint consists of an expression or routine that must return TRUE. If it returns FALSE, the attribute's value violates the constraint. Check constraints are defined at the type level when *expression_string* references two or more attributes. To define a check constraint, use:

```
CHECK(expression_string)
```

expression_string can be an expression or a complete user-defined routine. The expression or routine must be written in Docbasic and return a Boolean value. To specify a routine, you must have Sysadmin or Superuser user privileges.

For expressions, the syntax is:

```
expression [LANGUAGE docbasic]
[REPORT 'message_string' [ON VIOLATION]]
```

For user-written routines, the syntax is:

```
routine_name
([routine_parameter {,routine_parameter}])
FROM OBJECT (object_id |
PATH folder_path[VERSION version_label])
[LANGUAGE docbasic]
[REPORT 'message_string' [ON VIOLATION]]
```

routine_parameter is an attribute name.

The FROM clause must identify an object whose content file contains the source code for the routine. Identify the object using the object's ID or a folder path for an object. If you use a folder path, you can include a version label. If you don't include a version label, the label CURRENT is assumed.

Refer to Primary Key, page 77 for information about the error message clause.

mapping_table_specification

A mapping table specification is typically used to map a user-friendly character string value to an underlying value that may not be as readable or easy to understand. A mapping table can also be used to map localized or type-specific values to an underlying value.

For example, Desktop Client uses a mapping table defined at the type level to display a list of the display config objects appropriate for the object type. The actual names of the display config objects are mapped to more user-friendly strings. (Display config objects represent subsets of type attributes that have a common display definition.)

If you define a mapping table at the type level, the mappings apply to that object type and its subtypes.

component_specification

Component specifications identify which components can operate on objects of the type. The syntax is:

```
COMPONENTS (component_id_list)
```

component_id_list contains one or more entries with the following syntax:

```
component_classifier=object_id|NONE
```

component_classifier is a character string that represents a qualified component (a dm_qual_comp object). It consists of the component's class name and an acronym for the build technology used to build the component. For example, an component classifier might be Checkin.ACX or Checkin.HTML.

object_id is the object ID of the qualified component represented by the classifier. If you specify NONE instead of an object ID, the component is not available for the type.

Related Statements

Alter Type, page 43
Drop Type, page 89

Examples

The following example creates a new subtype called employee, sets the label text for most attributes and constraints in the attribute modifier list and the type modifier list, and publishes the data dictionary information.

```
CREATE TYPE "employee"
(
"emp_ssn" string(10)
(PRIMARY KEY,
 CHECK ('Len(emp_ssn)=10'
 LANGUAGE docbasic)
REPORT 'The SSN must have exactly 10 digits.'
ENFORCE BY APPLICATION,
 SET "label_text"='Social Security Number'),
"emp_first_name" string(32)
(SET "label_text"='First Name',
 NOT NULL),
"emp_middle_name" string(32)
(SET "label_text"='Middle Name'),
"emp_last_name" string(32)
(SET "label_text"='Last Name',
 NOT NULL),
"emp_disambiguator" integer,
"emp_department" integer
(FOREIGN KEY REFERENCES "department" ("department_id")
REPORT'The dept. code must be a valid code.',
 NOT NULL,
 SET "label_text"='Department Code'),
"emp_job_title" string(64)
(CHECK (validate_job_title ("emp_job_title")
FROM '\Admin\Scripts'
VERSION 'approved'
LANGUAGE docbasic
REPORT '$value is not a valid job title.')
)
WITH SUPERTYPE NULL
UNIQUE ("emp_first_name","emp_middle_name",
  "emp_last_name","emp_disambiguator"),
SET "label_text"='Employee Record'
PUBLISH
```

The following example creates a subtype of dm_document and publishes its information in the data dictionary:

```
CREATE TYPE "legal"
("lawyer" CHAR(30),"case_number" INT,
  "defendants" CHAR(30) REPEATING)
```

```
WITH SUPERTYPE "dm_document" PUBLISH
```

The following example creates a user-defined type with no supertype:

```
CREATE TYPE "my_base_type"
("author" CHAR(30), "title" CHAR(145), "doc_id" ID)
WITH SUPERTYPE NULL
```

The following example creates a subtype of the user-defined type my_base_type:

```
CREATE TYPE "acctg" ("accounting" CHAR(30) REPEATING)
WITH SUPERTYPE "my_base_type"
```

Delete

Purpose

Removes rows from a registered table.

Syntax

```
DELETE FROM table_name WHERE qualification
```

Syntax Description

Table 2–9. DELETE Syntax Description

Variable	Description
table_name	Identifies the registered table from which you are removing rows. Use the name of table in the underlying RDBMS.
qualification	Defines the conditions used to restrict the rows that are deleted. Refer to The WHERE Clause, page 130 for a description of the valid forms of a qualification.

Return Value

The DELETE statement returns a collection whose result object has one attribute, rows_deleted, that contains the number of rows deleted.

Permissions

To delete a row, the following conditions must be true:

- Your object-level permission for the dm_registered object in the Docbase that represents the RDBMS table must be at least Browse.
- Your table permission for the dm_registered object that represents the table must be DM_TABLE_DELETE.
- The user account under which Content Server is running must have the appropriate RDBMS permission to delete from the specified table. (The actual name of this permission will depend on your RDBMS.)

(For more information about security and object-level and table permissions, refer to Chapter 4, Security Services, in the *Content Server Administrator's Guide*.)

General Notes

The DELETE statement deletes rows from a registered table. A registered table is a table in the underlying RDBMS that has been registered with Content Server. (Refer to Register, page 102 for information about registering tables.) All rows for which the WHERE clause qualification evaluates to TRUE are deleted.

Related Statements

Insert, page 99
Register, page 102
Select, page 109
Unregister, page 137
Update, page 139

Example

The following example deletes the rows in the registered table authors_table that contain the name of any author who is not also found in the authors attribute of a document:

```
DELETE FROM "authors_table"
WHERE NOT EXISTS (SELECT * FROM "dm_document"
       WHERE ANY "authors" = authors_table.name)
```

Delete...Object

Purpose

Deletes objects from the Docbase.

Syntax

```
DELETE [PUBLIC]type_name[(ALL)]
[correlation_variable] OBJECT[S]
[IN ASSEMBLY document_id [VERSION version_label]
[NODE component_id][DESCEND]]
[SEARCH fulltext search condition]
[WHERE qualification]
```

Syntax Description

Table 2–10. DELETE...OBJECT Syntax Description

Variable	Description
type_name	Identifies the type of object to remove from the Docbase. Valid *type_names* are: dm_assembly and its subtypes dm_user and its subtypes dm_group and its subtypes dm_sysobject and its subtypes user-defined types with NULL supertypes and their subtypes
correlation_variable	Defines a qualifier for the type name that is used to clarify attribute references.

Variable	Description
IN ASSEMBLY clause	Restricts the statement to objects in a particular assembly. *document_id* identifies the document with which the assembly is associated. Use a literal object ID: `ID('`*`object_id`*`')` *version_label* specifies a particular version of the document. Use a symbolic or an implicit version label. If you do not include a version label, the server uses the version identified by the *document_id* argument. *component_id* specifies a particular component in the assembly. Including the NODE option restricts the statement to the specified component. Use a literal object ID to identify the component: `ID('`*`object_id`*`')` The DESCEND keyword directs the server to delete not only all directly contained node components, but also any indirectly contained components that meet the criteria. If you do not include this keyword, the server deletes only the directly contained components that meet the criteria in the statement.
SEARCH clause	Restricts the statement to objects that meet the SEARCH clause *fulltext search condition*. Refer to The SEARCH Clause, page 128 for a detailed description of the SEARCH clause.
WHERE clause	Restricts the statement to objects that meet the *qualification*. The qualification is an expression that evaluates to TRUE or FALSE. Refer to The WHERE Clause, page 130 for a description of the valid forms of a qualification.

Return Value

The DELETE...OBJECT statement returns a collection whose result object has one attribute, objects_deleted, that contains the number of objects deleted.

Permissions

You must have Delete permission on an object to delete the object.

General Notes

Deleting objects is subject to the following conditions:

- The object cannot belong to a frozen assembly or a frozen or immutable virtual document.
- If the compound_integrity attribute in the server's server config object is set to TRUE, the object cannot belong to any virtual document, regardless of the whether the document is immutable or not.

The *type_name* argument specifies the type of object to delete. The server searches all objects of the specified type and any subtypes for objects that meet any additional criteria you define in the statement.

The keyword PUBLIC restricts the query to those objects with the r_is_public attribute set to TRUE. If the query contains a SEARCH clause, the full-text search is restricted to those documents for which ISPUBLIC is TRUE. When the server queries or searches only public objects, it uses only the setting of r_is_public for security checks.

The keyword ALL directs the server to consider all versions of each object. If you do not include ALL, the server only considers versions with the symbolic label CURRENT. You must enclose ALL in parentheses.

After the server finds all objects that meet the defined criteria, it deletes those for which you have Delete permission. If any of the objects that are otherwise eligible for deletion belong to a frozen assembly or an unchangeable virtual document, then the entire statement is rolled back and no objects are deleted.

The optional clauses, IN ASSEMBLY, SEARCH, and WHERE, restrict the statement's scope. The IN ASSEMBLY clause restricts the operation to a particular virtual document assembly or node (component) within the virtual document. The SEARCH clause restricts the operations to indexed objects that meet the fulltext search condition. The WHERE clause restricts the operations to objects that meet the specified qualification. The clauses are applied in the following order:

- SEARCH
- IN ASSEMBLY
- WHERE

You cannot use DELETE...OBJECT to destroy record objects. Use the Destroy API command instead.

The IN ASSEMBLY Clause

Including the IN ASSEMBLY clause generates an error if the compound_integrity attribute in the server's server config object is set to TRUE. This attribute controls whether objects contained in a virtual document may be deleted from the Docbase.

If the document identified by *document_id* does not have an associated assembly, the statement fails with an error.

Related Statements

Change...Object, page 59
Create...Object, page 65
Update...Object, page 141

Examples

This example deletes all old documents owned by Joe:

```
DELETE "dm_document" OBJECTS
WHERE "r_modify_date" < date('01/01/1970')
AND "owner_name" = 'joe'
```

The following statement deletes all documents that contain the word yeast but not the word rolls:

```
DELETE "dm_document" OBJECTS
SEARCH DOCUMENT CONTAINS 'yeast' AND NOT'rolls'
```

This statement deletes all documents that contain the words yeast and bread and those that contain the words cake and wedding:

```
DELETE "dm_document" OBJECTS
SEARCH DOCUMENT CONTAINS 'yeast' AND 'bread'
OR 'cake' AND 'wedding'
```

The following statement deletes all workflows that have either Janine or Jeremy as a supervisor:

```
DELETE "dm_workflow" OBJECTS
WHERE "supervisor_name" = 'janine' OR
"supervisor_name" = 'jeremy'
```

This example also deletes all workflows that have Janine or Jeremy as a supervisor:

```
DELETE "dm_workflow" OBJECTS
WHERE "supervisor_name" IN ('janine','jeremy')
```

Drop Group

Purpose

Removes a group from the Docbase.

Syntax

```
DROP GROUP group_name
```

Syntax Description

Table 2–11. DROP GROUP Syntax Description

Variable	Description
group_name	Identifies the group to remove from the Docbase. You can specify the name as an identifier or as a character string literal.

Permissions

You must be the owner of a group or have Superuser user privileges to drop a group.

General Notes

When you drop a group, Content Server also removes any registry objects that identify the group as the subject of an audit or event notification request.

Related Statements

Alter Group, page 41
Create Group, page 63

Example

This example drops the group called rons_baby_gift:

```
DROP GROUP rons_baby_gift
```

Drop Type

Purpose

Removes a user-defined object type from the Docbase.

Syntax

```
DROP TYPE type_name
```

Syntax Description

Table 2–12. DROP TYPE Syntax Description

Variable	Description
type_name	Identifies the object type to remove.

Permissions

You must own the type or have Superuser privileges to drop a type.

General Notes

The type you identify must meet the following conditions:

- No objects of the type can exist in the Docbase.
- The type cannot have any subtypes.

Dropping a type also removes data dictionary information for the type and its attributes. The information is removed when one of the following occurs:

- The publish_dd method is executed for the entire Docbase.
- The Data Dictionary Publisher job runs.

Note: After you drop a type, you may not be able to create a type by the same name immediately. Allow time for the system to synchronize the type cache before attempting to recreate the type.

Related Statements

Alter Type, page 43
Create Type, page 68

Example

This example removes the user-defined type my_base_type from the Docbase:

```
DROP TYPE "my_base_type"
```

Execute

Purpose

Executes administration methods.

Syntax

```
EXECUTE admin_method_name [[FOR] object_id]
[WITH argument = value {,argument = value}]
```

Syntax Description

Table 2–13. EXECUTE Syntax Description

Variable	Description
admin_method_name	Specifies which administration method you want to execute. Table 2–14, page 92 lists the methods. Administration method names are not case sensitive.
object_id	Identifies the object on which you want the function to operate. Use the object's object ID.
WITH clause	Defines one or more arguments and their values for the administration method. Valid arguments differ for each method. Refer to Chapter 3, Administration Methods, for a description of each administration method and their arguments.

Return Value

The EXECUTE statement returns a collection. The attributes of the query result object in the collection depend on which administration method was executed. Refer to the description of each method in Chapter 3, Administration Methods, for details.

Permissions

The privileges required depend on the administration method you are executing. Refer to the description of the individual method in Chapter 3, Administration Methods, for information.

General Notes

The EXECUTE statement is the DQL equivalent of the API Apply method. You can use EXECUTE to invoke any of the administration methods that you can invoke using the Apply method except PING and WEBCACHE_PUBLISH. These cannot be executed using the EXECUTE statement.

Table 2–14, page 92 lists the administration methods that you can invoke with the EXECUTE statement and briefly describes the task that each performs.

Table 2–14. Administration Methods by Category for the EXECUTE Statement

Category of Operation	Function	Description
Process Management	CHECK_SECURITY, page 166	Checks a user or group's permissions level for one or more objects.
	GET_INBOX, page 201	Returns items in user's Inbox.
	MARK_AS_ARCHIVED, page 239	Sets the i_is_archived attribute of a dm_audittrail, dm_audittrail_acl, or dm_audittrail_group object to T.
	PURGE_AUDIT, page 252	Deletes audit trail entries from the Docbase.
	ROLES_FOR_USER, page 278	Returns the roles assigned to a user in a particular client domain.
Execute procedures	DO_METHOD, page 180	Executes system-defined procedures such as lpq or who or user-defined procedures.
	HTTP_POST, page 211	Directs the execution of a method to an application server.
Content storage management	CAN_FETCH, page 160	Determines whether content in a distributed storage area component can be fetched by the server.
	CLEAN_LINKS, page 170	Provides maintenance for linked store storage areas. On Windows platforms, cleans up unneeded linkrecord objects and resets file storage object security. On UNIX platforms, cleans up unneeded linkrecord objects, directories, and links associated with linked storage areas.
	DELETE_REPLICA , page 176	Removes a replica from a distributed storage area.

Category of Operation	Function	Description
	DESTROY_CONTENT, page 178	Removes a content object and its associated file from the Docbase. (Do not use this for archiving; use PURGE_CONTENT instead.)
	GET_FILE_URL, page 197	Returns the URL to a content file.
	GET_PATH, page 205	Returns the path to a particular content file in a particular distributed storage area component.
	IMPORT_REPLICA, page 216	Imports an external file as a replica of content already in the Docbase.
	MIGRATE_CONTENT, page 242	Moves content files from one storage area to another.
	PURGE_CONTENT, page 260	Deletes a content file from a storage area. (Used as part of the archiving process.)
	PUSH_CONTENT_ATTRS, page 261	Sets the content metadata in a content-addressed storage system for a document stored in that storage system.
	REGISTER_ASSET, page 264	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. This method is only available or useful if you have Documentum Media Services running.
	REPLICATE, page 269	Copies content in one component of a distributed storage area to another area.
	RESTORE_CONTENT, page 273	Moves a file or files from archived storage to the original storage location.
	SET_CONTENT_ATTRS, page 283	Sets the content_attr_name and content_attr_value attributes in the content object associated with the content file.
	SET_STORAGE_STATE, page 290	Sets the state of a storage area to off-line, on-line, or read-only.

Category of Operation	Function	Description
	SYNC_REPLICA_RECORDS, page 297	Synchronizes the replica record objects with the current definition of a distributed storage area. **Note:** This method is not available through Documentum Administrator.
	TRANSCODE_CONTENT, page 300	Queues a request for a content transformation to Media Server. This method is only available or useful if you have Documentum Media Services running.
Database Methods	DB_STATS, page 174	Provides database operation statistics for a session.
	DROP_INDEX, page 188	Drops an index.
	EXEC_SQL, page 194	Executes SQL statements.
	FINISH_INDEX_MOVES, page 196	Completes an interrupted object type index move operation.
	MAKE_INDEX, page 235	Creates an object type index.
	MOVE_INDEX, page 249	Moves an object type index from one tablespace to another. **Note:** Not supported on DB2.
	REORGANIZE_TABLE, page 266	Reorganizes a database table for query performance.
	UPDATE_STATISTICS, page 309	Updates the statistics for a database table.
Full-Text Methods	ADOPT_FTINDEX, page 156	Swaps attribute values between fulltext index objects, effectively making specified standby index the searchable index for a storage area.
	CHECK_FTINDEX, page 164	Reports on the status of the most recent index update.
	CLEAN_FTINDEX, page 168	Removes unwanted files and entries from an index. Unwanted files are temporary files that are created as part of the updating process and old versions of the index. Unwanted entries are entries associated with deleted documents.

Category of Operation	Function	Description
	CLEAR_PENDING, page 172	Recovers from an update that cannot be completed by marking all the content in the update as bad.
	DUMP_FTINDEX, page 190	Creates a dump file containing a specified full-text index. (Supports shadow indexes for replicated Docbases.)
	ESTIMATE_SEARCH, page 192	Returns the number of results matching a particular SEARCH condition.
	GET_FTINDEX_SIZE, page 199	Returns the size, in bytes, of an index.
	GETSTYLE_FTINDEX, page 208	Retrieves the topic set or style file associated with an index.
	LOAD_FTINDEX , page 230	Loads a dump file created by DUMP_FTINDEX into an index. (Supports shadow indexes for replicated Docbases.)
	MARK_ALL, page 237	Finds all content objects that are not indexed but should be (the a_full_text attribute set for the associated object is set to T) and marks these content object for indexing.
	MARK_FOR_RETRY, page 240	Finds all content objects that have a particular negative update_count and marks them as awaiting indexing.
	MODIFY_TRACE, page 247	Sets the tracing level for full-text indexing operations.
	RESET_FTINDEX, page 271	Reinitializes an index. Use when adding or dropping indexed attributes or to reinitialize a corrupted index.
	RMSTYLE_FTINDEX, page 275	Removes a topic set or a user-defined style file.
	SETSTYLE_FTINDEX, page 292	Associates a specified topic set or style file with an index.
	UPDATE_FTINDEX, page 303	Adds new entries to a full-text index and marks for removal entries associated with deleted documents.

Category of Operation	Function	Description
Session Management	CHECK_CACHE_CONFIG, page 161	Requests a consistency check on a particular cache config object.
	GET_LAST_SQL, page 204	Returns the last SQL statement issued.
	GET_SESSION_DD_ LOCALE, page 207	Returns the locale in use for the current session.
	LIST_AUTH_PLUGINS, page 218	Lists the authentication plugins loaded by Content Server.
	LIST_RESOURCES, page 220	Provides information about the server operating system environment.
	LIST_SESSIONS, page 224	Provides information about current, active sessions.
	LIST_TARGETS, page 228	Lists the DocBrokers defined as targets for the server. The information is returned in a collection with one result object whose attributes list the DocBrokers defined as targets for the server.
	LOG_ON, page 233 and LOG_OFF, page 232	Turn server logging of information about RPC calls on or off.
	SET_APIDEADLOCK, page 280	Sets a deadlock trigger on a particular API method or operation.
	SET_OPTIONS, page 287	Turn various tracing options on or off.
	SHOW_SESSIONS, page 296	Provides information about current, active sessions and a user-specified number of timed-out sessions.

Unlike the Apply method, the EXECUTE statement is not case sensitive. Also, you do not have to specify the datatype of the arguments for each function. The statement determines the datatype from the value you assign to the argument.

Examples

Refer to the individual descriptions of the method in Chapter 3, Administration Methods, for examples.

Grant

Purpose

Gives one or more user privileges to one or more users.

Syntax

```
GRANT privilege {,privilege} TO users
```

Syntax Description

Table 2–15. GRANT Syntax Description

Variable	Description
privilege	Identifies the privilege you want to grant. Valid privileges are SUPERUSER SYSADMIN CREATE TYPE CREATE CABINET CREATE GROUP CONFIG AUDIT PURGE AUDIT VIEW AUDIT
users	Identifies the users to whom you want to a privilege or privileges. The user name must belong to an individual user. You can specify one or more users as a comma-separated list of user names or use a SELECT statement to identify the user names. (Refer to Examples, page 98 for the use of a SELECT statement.) The user name must be the value is found in the user_name attribute of the dm_user object associated with the user.

Permissions

To grant Superuser or Sysadmin user privileges, you must have Superuser privileges.

To grant Create Group, Create Type, or Create Cabinet user privileges, you must have Superuser or Sysadmin user privileges.

To grant Config Audit, Purge Audit, Or View Audit privileges, you must be the Docbase owner or a Superuser. Docbase owners and Superusers cannot grant these privileges to themselves.

General Notes

Granting a privilege to a user who already has that particular privilege does not generate an error.

Related Statements

Revoke, page 107

Examples

The following example grants the Create Type user privilege to the users donna and carol:

```
GRANT CREATE TYPE TO donna,carol
```

This example grants the Create Cabinet user privilege to all individual users (that is, to those user names that do not represent a group):

```
GRANT CREATE CABINET TO
(SELECT "user_name" FROM "dm_user"
 WHERE "r_is_group" = FALSE)
```

The final example grants two privileges to the users jim and mike:

```
GRANT SYSADMIN,SUPERUSER TO jim,mike
```

Insert

Purpose

Inserts a row into a registered table.

Syntax

```
INSERT INTO table_name [(column_name {,column_name})]
VALUES (value {,value}) | dql_subselect
```

Syntax Description

Table 2–16. INSERT Syntax Description

Variable	Description
table_name	Identifies the registered table in which you want to insert new rows.
column_name	Identifies a column in the table to receive an assigned value.
value	Specifies the value assigned to a column. The number of values specified must equal the number of columns named or the number of columns in the table. Refer to Assigning Values to Columns, page 100 for more information.
dql_subselect	Specifies a SELECT statement. The returned values are inserted into the table.

Return Value

The INSERT statement returns a collection whose result object has one attribute, rows_inserted, that contains the number of rows inserted into the table.

Permissions

To use the INSERT statement, the following conditions must be true:

- Your object-level permission for the dm_registered object representing the RDBMS table must be at least Browse.
- Your table permission for the dm_registered object representing the table must be DM_TABLE_INSERT.
- The user account under which Content Server is running must have the appropriate RDBMS permission to insert data into the specified table. (The actual name of this permission will depend on your RDBMS.)

For more information about security and object-level and table permissions, refer to Chapter 4, Security Services, in the *Content Server Administrator's Guide*.

Assigning Values to Columns

Which value is inserted in each column you specify is determined by position. That is, the first column you specify in the statement receives the first value. The second column receives the second value, and so forth. Consequently, if you are inserting a value in every column, it is not necessary to specify the columns; the server automatically inserts the values in each column in turn. However, if you omit the column names, the number of values must equal the number of columns in the table.

If you specify column names, you can insert values into a subset of the table's columns. Also, it is not necessary to specify the column names in the same order in which they appear in the table. Columns that are not specified in the INSERT statement receive default values. (Refer to the documentation for your underlying RDBMS for information about the default values assigned to columns in this situation. Different systems have different rules. For example, some database management systems only allow you to default nullable columns. In such systems, all non-nullable columns must be specified.)

Defining Values

There are two possible ways to define the values to assign to columns. You can use the VALUES clause or you can use a DQL subselect statement.

The VALUES clause has the syntax:

```
VALUES value {,value}
```

One value must be specified for each column named in the statement. Each value must be a literal value appropriate for the datatype of the column to which it is being assigned. For example, assume there is a registered table called customer_accounts and that you want to insert values into four of its columns: customer_name, acct_number, open_date, and balance. The customer_name and acct_number columns are character string datatypes. The open_date column is a date datatype, and the balance column is a floating point datatype. The following statement inserts values into these columns:

```
INSERT INTO "customer_accounts" ("customer_name","account_number","open_date,
balance") VALUES ('Henrietta Hornsby','01264',date('4/2/1993'),125.00)
```

The value Henrietta Hornsby is inserted into the customer_name column, the value 01264 is inserted into the account_number column, and so forth.

Similarly, when you use a subselect statement, the returned values must be equal in number to the number of columns named in the INSERT statement and the values must be appropriate for the column into which they will be inserted. (Refer to Select, page 109 for the subselect statement's syntax.)

Related Statements

Delete, page 82
Register, page 102
Select, page 109
Unregister, page 137
Update, page 139

Example

This example saves the object names and creation dates of the contents of the flowers folder into the objects table:

```
INSERT INTO "objects" ("o_name", "c_date")
SELECT "object_name", "r_creation_date"
FROM "dm_document"
WHERE FOLDER ('/public/subject/flowers')
```

Register

Purpose

Registers a table from the underlying RDBMS with the Docbase.

Syntax

```
REGISTER TABLE [owner_name.]table_name
(column_def {,column_def})
[[WITH] KEY (column_list)]
[SYNONYM [FOR] 'table_identification']
```

Syntax Description

Table 2–17. REGISTER Syntax Description

Variable	Description
owner_name	Identifies the table's owner. Use the owner's RDBMS user name. If the owner is a Docbase user, this value may be found in the user's user_db_name attribute. If the RDBMS is Oracle or DB2 and the owner is the DBA, you can use the alias dm_dbo. If the RDBMS is MS SQL Server or Sybase and the owner is the DBA, you can use either dbo or dm_dbo as an alias. *owner_name* is optional if the current user is the table's owner. The default value is the current user.

Variable	Description
table_name	Identifies the RDBMS table to register with the Docbase. You can specify the table's actual name or a synonym. The name must consist of ASCII characters. If you specify the table's actual name, do not include the SYNONYM clause in the statement. For Oracle or DB2, if you specify a synonym, that synonym must be previously defined through the RDBMS. Including the SYNONYM clause in the REGISTER statement is optional. For MS SQL Server or Sybase, if you specify a synonym, you must include the SYNONYM clause in the statement. For MS SQL Server, you cannot register a remote table. (Refer to The SYNONYM Clause, page 105 for details.) If you do not have Superuser user privileges, you must be the owner of the table.
column_def	Describes columns in the table. The format for a column definition is: `column_name datatype [(length)]` where *column_name* is the name of the column in the table and *datatype* is the DQL datatype that corresponds to the column's RDBMS datatype. The column name must consist of ASCII characters. Valid datatypes in a column definition are: float, double integer, int char, character, string date, time You must also specify a length for columns that have a character, char, or string datatype (for example, char(24)).

Variable	Description
column_list	Identifies the columns in the table on which indexes have been built. Use a comma-separated list of column names.
table_ identification	The name of the table in the underlying RDBMS. You must include the SYNONYM clause if you run against MS SQL Server or Sybase and you specify a synonym as the table name in the statement. Including the SYNONYM clause is optional if you are running against Oracle or DB2 and specify a synonym as the table name in the statement. (Refer to The SYNONYM Clause, page 105 for details.)

Return Value

When issued through IDQL, the REGISTER statement returns the object ID of the dm_registered object for the table. If you issue the statement through IAPI (using the Query method), the statement returns a collection whose result object has one attribute, called new_object_id, that contains the object ID of the dm_registered object for the table.

Permissions

To register a table, you must own the table or have Superuser user privileges. If folder security is enforced in the Docbase, you must also have at least Write permission on the System cabinet.

General Notes

When you execute the REGISTER statement, the server creates an object of type dm_registered (a SysObject subtype) that represents the RDBMS table in the Docbase. This object is automatically linked to the system (/system) cabinet.

A dm_registered object can be manipulated like any other SysObject or SysObject subtype with one exception: you cannot version a dm_registered object.

You do not have to include all of the columns in the underlying RDBMS table in the statement. You can specify a subset of the underlying table's columns. The column definitions you provide need not match the column definitions for the underlying table. When the server creates the dm_registered object for the table, it uses your column definitions.

The REGISTER statement automatically assigns a default ACL to the dm_registered object. (The default is determined by the value in the default_acl attribute of the server's server config object.)

The statement also sets default table permissions. The table permissions (owner, group, and world table permits) are set to SELECT. You can change these by setting the attributes directly. Note that table permissions for registered tables are not hierarchical. (For a complete description of registered table permits, refer to The Table Permits, page 344, in the *Content Server Administrator's Guide*.)

Changes to the definition of an underlying table are not automatically reflected in its corresponding dm_registered object. If the underlying table is modified and you want the dm_registered object to match the underlying table definition, you must unregister the table and then register it with new column definitions.

The SYNONYM Clause

Use the SYNONYM clause to register RDBMS tables that have synonyms in the underlying tables. (A synonym for an RDBMS table must be created independently in the RDBMS before you register the table. The Register statement does not create a synonym.)

The SYNONYM clause records the actual name of the table that corresponds to the table's synonym. The name is stored in the synonym_for attribute of the table's dm_registered object.

When you run against Oracle or DB2, Content Server passes the synonym directly to the database server. The actual table name, as specified in the SYNONYM clause and recorded in the synonym_for attribute is purely informational. For example, suppose johndoe has a table called myremotetable in the remote Oracle database londonremote, and that this table has the synonym remote1. To register this table, he uses:

```
REGISTER TABLE johndoe."remote1" ("columnA" int)
SYNONYM FOR johndoe.myremotetable@londonremote
```

After he registers the table, he can use the synonym in DQL statements and it is passed directly to the database server. For example, issuing the following DQL:

```
SELECT * FROM johndoe."remote1"
```

generates the following SQL:

```
SELECT * FROM johndoe."remote1"
```

When you run against MS SQL Server or Sybase, Content Server substitutes the value you specified in the SYNONYM clause (recorded in the synonym_for attribute) for the table name in the SELECT, INSERT, UPDATE and DELETE statements. For example, suppose johndoe issues the following REGISTER statement:

```
REGISTER TABLE johndoe."remote1" ("columnA" int)
SYNONYM FOR londonserver.londonremote.johndoe.myremotetable
```

After he registers the table, he issues the following SELECT statement:

```
SELECT * FROM johndoe."remote1"
```

Content Server substitutes the actual table name for the synonym and the following SQL is generated:

```
SELECT * FROM londonserver.londonremote.johndoe.myremotetable
```

Related Statements

Delete, page 82
Insert, page 99
Unregister, page 137
Update, page 139

Example

The following statement registers the RDBMS table named departments:

```
REGISTER TABLE "departments"
("dept_name" CHAR(30), "dept_code" INT)
KEY ("dept_code")
```

Revoke

Purpose

Removes one or more user privileges from one or more users.

Syntax

```
REVOKE privilege {,privilege} FROM users
```

Syntax Description

Table 2–18. REVOKE Syntax Description

Variable	Description
privilege	Specifies the privilege to revoke. Valid privileges are: SUPERUSER SYSADMIN CREATE TYPE CREATE CABINET CREATE GROUP CONFIG AUDIT PURGE AUDIT VIEW AUDIT
users	Specifies the users from whom you are revoking the specified privilege or privileges. The user name must belong to an individual user. You can specify one or more users as a comma-separated list of user names or use a SELECT statement to identify the user names. (Refer to Examples, page 108 for the use of a SELECT statement.) The user name must be the value is found in the user_name attribute of the dm_user object associated with the user.

Permissions

To revoke the Sysadmin or Superuser user privilege, you must have Superuser user privileges.

To revoke the Create Group, Create Type, or Create Cabinet user privilege, you must have either Superuser or Sysadmin user privileges.

To revoke Config Audit, Purge Audit, Or View Audit privileges, you must be the Docbase owner or a Supersuser. Docbase owners and Superusers cannot revoke these privileges from themselves.

General Notes

The statement succeeds even if a user does not have the privilege being revoked.

Related Statements

Grant, page 97

Examples

The following example revokes the Create Cabinet and Create Type privileges from the users john and howard:

```
REVOKE CREATE CABINET, CREATE TYPE FROM john, howard
```

The next example demonstrates the use of a subselect statement to identify the users. This example revokes the Superuser user privilege from all users except haroldp:

```
REVOKE SUPERUSER FROM
(SELECT "user_name" FROM "dm_user"
WHERE "user_privilege" >= 16 AND "user_name" != 'haroldp')
```

Select

Purpose

Retrieves information from the Docbase, the database, or both.

Syntax

```
SELECT [FOR base_permit_level][ALL|DISTINCT] value [AS
name] {,value [AS name]}
FROM [PUBLIC] source_list
[IN DOCUMENT clause | IN ASSEMBLY clause]
[SEARCH [FIRST|LAST]fulltext_search_condition
[IN FTINDEX index_name{,index_name}]
[WITH|WITHOUT FT_OPTIMIZER]]
[WHERE qualification]
[GROUP BY value_list]
[HAVING qualification]
[UNION dql_subselect]
[ORDER BY value_list]
[ENABLE hint_list]
```

Syntax Description

Table 2–19. SELECT Syntax Description

Variable	Description
base_permit_level	Defines a minimum permission level for the returned objects. If specified, the query returns only those objects for which the user has at least the specified permit level. Valid values are: NONE, BROWSE, READ, NOTE, VERSION, WRITE, and DELETE. If unspecified, the default is BROWSE.

Variable	Description
value	Identifies the information to retrieve. Valid values are: • Attribute and column names • Scalar and aggregate functions • MFILE_URL function • Arithmetic expressions • The keywords CONTAIN_ID, CONTENTID, DEPTH, HITS, ISCURRENT, ISPUBLIC, ISREPLICA, MCONTENTID, MHITS, MSCORE, OBJTYPE, OFFSET, PAGE_NO, PARENT, POSITION, SCORE, SUMMARY, SYSOBJ_ID, TAG, TEXT, TEXTPAGE, THUMBNAIL_URL, USER • An asterisk (*) Multiple values can appear in any order. If the SELECT statement is a subselect or a subquery, you can only include one value.
name	Defines a name for the result object attribute that will contain the retrieved value. This argument is optional. If you omit it, the system provides a default name. (Refer to The AS Clause, page 122 for information about the default name.) *name* must consist of ASCII characters

Variable	Description

source_list	Identifies the object types and RDBMS tables to search. You can specify any combination of object types and RDBMS tables.
	
	Types are specified using the following syntax:
	
	`type_name [(ALL)|(DELETED)] [correlation_variable] [WTH passthrough_hint_list]`
	
	Tables are specified using the following syntax:
	
	`[owner_name.]table_name [correlation_variable] [WITH passthrough_hint_list]`
	
	passthrough_hint_list is a list of hints for one or more databases. The hint list for an individual database has the following format:
	
	`db_keyword('hint'{,'hint'})`
	
	Valid *db_keywords* are: ORACLE, SQL_SERVER, SYBASE, and DB2. *hint* is any valid hint accepted by the particular RDBMS.
	
	If you include hints for multiple databases, the hint lists for each must be separated by commas and the entire set (for all databases) enclosed in parentheses. Passthrough Hints, page 323 describes using passthrough hints fully.
	
	On Oracle and DB2, a passthrough hint in the source list in a subquery is applied to the entire SELECT statement.
	
	The RDBMS table must be a registered table unless the user issuing the SELECT is a superuser. Only superusers can query unregistered RDBMS tables. Additionally, there are several constraints on specifying more than one type in the source list.
	
	For complete information about specifying type and table names, refer to Source List, page 123.

IN DOCUMENT clause	Identifies the target of the SELECT statement as a particular virtual document. This clause can assemble a virtual document or identify the components of a virtual document. If you include this clause, you cannot specify the IN ASSEMBLY clause. Refer to The IN DOCUMENT Clause, page 125 for the syntax and usage of the IN DOCUMENT clause.

Variable	Description
IN ASSEMBLY clause	Identifies the target of the SELECT statement as an assembly of a virtual document. The server uses the components of the assembly as the definition of the virtual document and applies any other specified conditions to those components, rather than searching the document's hierarchy for components.
fulltext_search _condition	Defines criteria for searching of the full-text index. Refer to The SEARCH Clause, page 128 for a description of its syntax and use.
index_name	Identifies one or more indexes to be searched. Use the index name as defined in the associated fulltext index object. The index may be a searchable or standby index. You can include the IN FTINDEX clause only if the SELECT statement includes a SEARCH clause.
FT_OPTIMIZER clause	Indicates whether to optimize the full-text search query. You can include this clause only when the SELECT statement includes a SEARCH clause. The default is WITH FT_OPTIMIZER.
qualification	Restricts returned results to particular information. Refer to The WHERE Clause, page 130 for a full description of the valid forms of a qualification for a WHERE clause or a HAVING clause.
dql_subselect	Defines an additional DQL SELECT statement. The columns returned by the statement must correspond to those returned by the outermost SELECT statement in number and datatype.

Variable	Description
value_list	Defines an order in which to group or sort the returned results. The values in this list must also be selected values. Refer to The GROUP BY Clause, page 131 for the specific use of this in each clause.
hint_list	One or more standard or passthrough DQL hints. Refer to the description of the source list for the format of a passthrough hint. Standard hints are the following: SQL_DEF_RESULT_SET N FORCE_ORDER RETURN_TOP N OPTIMIZE_TOP N FETCH_ALL_RE- SULTS N where N is an integer value. For a brief description of the standard hints, refer to Table 2–20, page 134. Appendix A, Using DQL Hints, contains full information about using standard hints.

General Notes

The SELECT statement retrieves information from object types and RDBMS tables. By default, the statement returns only objects for which the user has at least Browse permission. If you want to enforce a higher level of permission on the returned objects, you can include the FOR *base_permit_level* clause. For example, suppose you issue the following SELECT statement:

```
SELECT FOR VERSION "r_object_id","object_name","owner_name"
FROM "dm_document" WHERE "subject"='budget_proposal'
```

The query returns all budget proposal documents for which the currently logged-in user has at least Version permission.

You can execute the statement as a stand alone statement and indirectly, as part of a variety of other DQL statements. For example, the following CREATE GROUP statement uses a SELECT statement to select the users that will populate the new group:

```
CREATE GROUP supers MEMBERS
(SELECT "user_name" FROM "dm_users"
 WHERE "user_privilege" >= 16)
```

(For more information about CREATE GROUP, refer to Create Group, page 63 .)

The ALL and DISTINCT Keywords

The optional ALL and DISTINCT keywords determine whether the SELECT statement returns duplicate rows. ALL returns all rows, including any duplicates. DISTINCT does not return duplicate rows. If neither keyword is included, the default is determined by the server's default behavior. (The server's default behavior is determined by the distinct_query_results flag in the server's server.ini start-up file.)

The DISTINCT keyword is ignored if the SELECT statement also includes an IN DOCUMENT or IN ASSEMBLY clause. The server issues a warning if the statement includes both the DISTINCT keyword and an IN DOCUMENT or IN ASSEMBLY clause. The warning is also issued if the server's default setting is not to return duplicates and the query contains an IN DOCUMENT or IN ASSEMBLY clause.

Selecting Attribute Values

Select object attribute values by specifying the attribute's name as a value. The attributes you specify must belong to the object type specified in the FROM clause. The attributes can be either single-valued or repeating attributes. However, selecting repeating attributes is subject to the following limitations:

- You cannot select a repeating attribute and use the keyword TAG or POSITION.
- You cannot select a repeating attribute and the name of a column from a registered table.
- You cannot select a repeating attribute if there are multiple object types identified in the FROM clause.
- You cannot select any of the following attributes if you are querying audit trail entries unless you have Superuser or View_audit privileges:

  - acl_name
  - acl_domain
  - attribute_list
  - attribute_list_id
  - chronicle_id
  - object_name
  - object_type
  - owner_name
  - session_id
  - version_label

The following statement returns the value of title, a single-valued attribute, for all documents in the Docbase:

```
SELECT "title" FROM "dm_document"
```

This next example returns the value of authors, a repeating attribute, from all documents that have bread as their subject:

```
SELECT "authors" FROM "dm_document"
WHERE "subject"='bread'
```

You can select both repeating and single-valued attributes in the same statement. By default, the server returns a separate result object for each value in the selected repeating attribute. For example, suppose a document has three authors, Jean, Connie, and Corwin. The following statement returns three result objects:

```
SELECT "object_name","authors" FROM "dm_document"
WHERE "title" = 'Breads of France'
```

Assuming that the object_name is French_breads, the results would be:

object_name	authors
French_breads	Jean
French_breads	Connie
French_breads	Corwin

If you want to return one result object that includes all of the repeating attribute values, include the r_object_id attribute as a selected value and order the results with that attribute as the primary sort element. For example, the following statement returns one result object that contains the names of all three authors:

```
SELECT "object_name","r_object_id","authors"
FROM "dm_document"
WHERE "title" = 'Breads of France'
ORDER BY "r_object_id"
```

This statement returns the following object:

r_object_id	object_name	authors
object_id	French_breads	Jean
		Connie
		Corwin

(Refer to The ORDER BY Clause, page 133 for more information.)

You can use an asterisk (*) to select a predefined set of system-defined attributes for the object. Refer to The Asterisk (*) as a Selected Value, page 121 for details.

Selecting Column Values

Select column values by specifying column names as values. The columns must belong to a RDBMS table that you identify in the FROM clause. You can use column names to select some or all of the column values, or you can use an asterisk to retrieve all column values.

Note: The account from which Content Server was installed must have SELECT privileges on the underlying table in the RDBMS before you can use DQL to select from an RDBMS table. Additionally, to query a registered table, you must have DM_TABLE_SELECT table permission and at least Browse object-level permission for the dm_registered object representing the table. If the SELECT statement includes a SEARCH clause, your object-level permission must be at least Read.

Scalar and Aggregate Functions as Selected Values

You can include scalar and aggregate functions to manipulate returned values.

A scalar function operates on one value. The DQL SELECT statement accepts three scalar functions:

- UPPER, which returns the uppercase form of a character string
- LOWER, which returns the lowercase form of a character string
- SUBSTR, which returns a subset of a specified character string

An aggregate function operates on a set of values and returns one value. The DQL SELECT statement accepts five aggregate functions:

- COUNT, which returns the number of values in a group of values
- MIN, which returns the minimum value in a group of values
- MAX, which returns the maximum value in a group of values
- AVG, which returns the average value of a group of values
- SUM, which returns the total of all values in a group of values

For example, the following statement returns the number of documents owned by the user horace:

```
SELECT COUNT(*) FROM "dm_document"
WHERE "owner_name"='horace'
```

You cannot include an aggregate function if your statement includes an IN DOCUMENT clause unless the statement also includes the USING ASSEMBLIES clause and an assembly exists for the virtual document itself. (Refer toAssemblies, page 148 in *Content Server Fundamentals*for information about assemblies.) However, in this case, you may prefer to use the IN ASSEMBLY clause instead of the IN DOCUMENT clause with USING ASSEMBLIES. Refer to The IN ASSEMBLY Clause, page 127 for more information.

MFILE_URL Function as a Selected Value

The MFILE_URL function returns URLs for the content files and renditions associated with the objects returned by the SELECT statement. The function has three arguments that are used to control which URLs are returned. For a description of the function and its arguments, refer to The MFILE_URL Function, page 26.

Arithmetic Expressions as Selected Values

You can include arithmetic expressions to perform calculations on the returned values. Arithmetic expressions are expressions that use arithmetic operators to form the expression, such as:

```
attribute A + attribute B
column C * column D
(2 * column B) - attribute F
attribute X + 4
```

The following statement returns the total of all rental charges paid in June, including regular rents and late charges:

```
SELECT "rent_received" + "late_charge_total" AS month_total
FROM "yearly_rent_records"
WHERE "mon" = 'june'
```

Keywords as Selected Values

Keywords are words that have a special meaning for the server. The majority are full-text keywords. That is, the values they return are attribute values stored in the full-text index or values computed during a full-text search. Three keywords return information about an object's relationships to other objects in a virtual document. The remaining keywords determine whether an object is a replica, return the current user name, and return the URL to an object's thumbnail file.

It isn't necessary to include a SEARCH clause in the query to include a full-text key in the selected values list. However, for some keywords such as TEXT or POSITION, the returned values are not useful unless a SEARCH clause is included.

Full-Text Keywords

The following keywords return information about full-text indexes.

Note: Any of the full-text keywords can also be specified in a WHERE clause.

- CONTENTID and MCONTENTID

  The CONTENTID and MCONTENTID keywords return the object ID of the object representing the content file that matches the select criteria. (A content object links a content file to all documents that contain the file.)

  Use CONTENTID if the document has only one content file. Use MCONTENTID if the document has multiple content files (multiple pages). MCONTENTID is returned as a repeating attribute. The value at each index position is the object ID of the content object for the content file at that position in the document.

- HITS and MHITS

  The HITS and MHITS keywords return an integer number representing the number of places in the document where the search criteria were matched. For example, the following statement counts the number of times the words workflow and flowchart occur within each document and returns the total as the HITS value:

  ```
  SELECT "r_object_id", "object_name",HITS FROM "dm_document"
  SEARCH DOCUMENT CONTAINS 'workflow' OR 'flowchart'
  ```

- Use the HITS keyword when the document has only one content file. Use MHITS when the document has multiple content files (multiple pages). The results obtained using MHITS are returned in a repeating attribute. The number at each index level corresponds to the number of hits in the content file at that position in the document.

  If you include HITS or MHITS as part of the WHERE clause qualification, you must also include it in the values list.

- ISCURRENT

  The ISCURRENT keyword is a Boolean keyword that returns 1 (TRUE) if the object is the current version and 0 (FALSE) if the object is not the current version.

- ISPUBLIC

  The ISPUBLIC keyword is a Boolean keyword that returns 1 (TRUE) if the object is a public object (its r_is_public attribute is TRUE) and 0 (FALSE) if the object is not a public object.

- OBJTYPE

  The OBJTYPE keyword returns the r_object_type of the selected object.

- OFFSET

  The OFFSET keyword works in conjunction with TEXTPAGE to identify the location of a particular word in a document. It returns an integer value that represents the location of the word on the document page, expressed as the number of words from the beginning of the page. This keyword is only useful against documents that are indexed from PDF renditions (not PDFText renditions). If the returned value is formatted in an XML file, it can be used by Adobe's web-highlighting plug-in.

- PAGE_NO

  The PAGE_NO keyword returns the page number of particular content. For example:

  ```
  SELECT "r_object_id", "object_name",
  PAGE_NO FROM "dm_document"
  SEARCH DOCUMENT CONTAINS 'workflow' OR 'flowchart'
  ```

  For each document, the query returns the page number of the document content that contains the specified words (in addition to the document's object ID and object name).

- POSITION and TAG

  The POSITION and TAG keywords return values relating to the indexing entries found in a full-text indexed document. The return value is the offset of each index entry from the beginning of the file.

  POSITION works only for content in PDFText format. (TAG is currently unsupported by the Full-Text Engine and is included for future enhancements.)

  You can use these keywords together or use only the keyword POSITION. In either case, you must also select the r_object_id attribute and order the statement results using that attribute as the primary element:

  ```
  SELECT "r_object_id", "object_name",
  POSITION FROM "dm_document"
  SEARCH DOCUMENT CONTAINS 'zyx'
  ORDER BY "r_object_id"
  ```

  Additionally, you cannot include a repeating attribute in the values list if you include TAG, POSITION, or both.

- SCORE and MSCORE

  The SCORE and MSCORE keywords return a document's relevance ranking.

  Use SCORE and MSCORE in conjunction with the ACCRUE concept operator. Each returns a number that indicates how many of the specified words were found in each returned document. For example, look at the following statement:

  ```
  SELECT "object_name", SCORE FROM "dm_document"
  SEARCH TOPIC '<ACCRUE>("document","management","workflow")'
  WHERE SCORE >=75
  ```

  The statement returns all documents that score from .7 to 1. The more words found in a document, the higher the document's score.

  Use the SCORE keyword when the document has only one content file. Use MSCORE when the document has multiple content files (pages). The results obtained using MSCORE are returned in a repeating attribute. The value at each index level corresponds to the relevance ranking assigned to the content at that position in the document.

- SUMMARY

  The SUMMARY keyword returns a summary of each document returned by the SELECT statement. The summary is four sentences from the document, chosen by the Full-Text Engine. The Full-Text Engine chooses the four sentences most indicative of the overall theme of the document, based on an analysis of word and phrase frequency, density of keywords in the sentence, location of the sentence in the document, and other lexical analysis techniques.

For example, the following statement returns a summary of the document whose name is Company History:

```
SELECT SUMMARY FROM "dm_document"
WHERE "object_name"='Company History'
```

Note: Sentences vary in length. The summary may be truncated if the total length exceeds the maximum of 255 characters that may be returned by keyword.

- TEXT

  The TEXT keyword returns the actual text of a matched word. Use this keyword when you include a SEARCH clause that does not specify exact word matches in its full-text search qualification. For example, you can use TEXT if you specify a thesaurus, STEM, or SOUNDEX query.

- TEXTPAGE

  The TEXTPAGE keyword returns the number of the page in a PDF document that contains a particular word or phrase. This keyword is generally selected in conjunction with OFFSET for use by Adobe's web-highlighting plug-in. (You must format the returned values in an XML file and make it available to Adobe through a web server.)

Virtual Document Keywords

The following keywords return information about relationships in a virtual document.

- CONTAIN_ID

  The CONTAIN_ID keyword returns the object IDs of the containment objects associated with the components of a virtual document. The CONTAIN_ID keyword cannot return containment IDs for components that are found by searching an assembly. Consequently, if the SELECT statement includes the USING ASSEMBLIES clause, the containment IDs of any components found using an assembly will be zeros ('0000000000000000').

  For example, suppose you have a virtual document, A, with one component, B. B is also a virtual document with four components and an assembly. If you execute a SELECT statement to retrieve the containment IDs of the components of A and include the DESCEND keyword, you receive the containment IDs for A's components at all levels. If the SELECT statement includes DESCEND and USING ASSEMBLIES, you receive only the containment ID for B. (B's containment object links it to A.) Because the direct components of B are found using an assembly, their containment IDs are not returned.

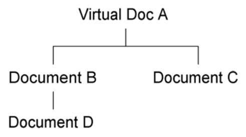
- DEPTH

  The DEPTH keyword returns a component's level within a virtual document. You can only use the DEPTH keyword if the SELECT statement includes the IN DOCUMENT clause and the clause includes the DESCEND keyword.

  To illustrate its use, suppose the following query is executed against the virtual document shown in :

```
SELECT "r_object_id", DEPTH FROM "dm_document"
IN DOCUMENT ID('object_id_A') DESCEND
```

Sample Virtual Document



The statement returns

Object ID	DEPTH
A	0
B	1
D	2
C	1

- PARENT

  The PARENT keyword returns the object ID of the object that directly contains a direct or indirect component of a virtual document. You can only include the PARENT keyword in the value list if the SELECT statement contains an IN DOCUMENT clause.

  For example, suppose the following query is executed against the virtual document shown in :

```
SELECT "r_object_id", PARENT FROM "dm_document"
IN DOCUMENT ID('object_id_A') DESCEND
```

  The statement returns the object IDs and parents of each component of the virtual document:

Object ID	PARENTS
A	A
B	A
D	B
C	A

  Note that a virtual document is always considered to be its own parent.

Miscellaneous Keywords

The following keywords return miscellaneous information.

- ISREPLICA

  The ISREPLICA keyword is a Boolean keyword that returns 1 (TRUE) if the returned object is a replica and 0 (FALSE) if the object is not a replica. (*Replica* is the term used to describe any object in the Docbase that has been placed in the Docbase as a copy of an object from another Docbase. Only environments using object replication have replicas.)

  You can only include the ISREPLICA keyword in the value list of the outermost SELECT statement. You cannot use ISREPLICA in a subquery.

- THUMBNAIL_URL

  The THUMBNAIL_URL keyword returns the URL to a thumbnail file. Use THUMBNAIL_URL in a SELECT statement's selected values to return the URL to the thumbnail associated with the returned objects. For example:

  ```
  SELECT object_name,THUMBNAIL_URL FROM dm_document
  WHERE FOLDER('/Corporate Objectives')
  ```

  The statement returns the documents in the folder Corporate Objectives. If the content of any of returned document has an associated thumbnail, the URL to that thumbnail is returned also. If a document has multiple content pages with thumbnails, the keyword returns only the thumbnail associated with the first content page that has a thumbnail.

  The URL contains the following information:

  — The base URL defined for the thumbnail storage area

  — A path relative to the storage area identified in the store attribute that points to the thumbnail file

  — A store attribute that identifies the storage area containing the thumbnail

  If the storage area's require_ticket attribute is TRUE, the URL also contains a ticket that contains an encryption of the path plus a time stamp. The ticket is valid for five minutes, which means that the URL is good only for five minutes.

  For example:

  ```
  http://myserver.documentum.com:8080/getThumbnail?
  path=00232803/80/00/01/Ob.jpg&store=thumbnail_store_01&ticket=8002DWR670X
  ```

  `http://myserver.documentum.com:8080/getThumbnail?` is the base URL.

  `path=00232803/80/00/01/Ob.jpg` specifies the path to the file.

  `store=thumbnail_store_01` identifies the storage area.

  `ticket=8002DWR670X` is the optional ticket.

- USER

  The USER keyword returns the current user's user name.

The Asterisk (*) as a Selected Value

If the FROM clause references only registered tables or only object types, you can use an asterisk (*) in the values list.

For registered tables, the asterisk returns all columns in the table. For object types, the asterisk returns the following set of attributes:

- All read/write single-valued attributes
- The r_object_id attribute
- If the object is a SysObject or SysObject subtype:

  — r_object_type

  — r_creation_date

  — r_modify_date

  — a_content_type

If you select from more than one type, the * applies only to the first type in the FROM clause.

For example, here is a statement that uses an asterisk:

```
SELECT * FROM "dm_outputdevice"
```

The AS Clause

The AS clause lets you name the attributes of the query result objects that represent the returned results. For example, suppose you issue the following SELECT statement:

```
SELECT "a"+"b" AS total FROM "accts"
```

Total is assigned as the name of the query result object attribute that contains the returned a+b values.

When the statement returns the value of an attribute or column name, providing a name for the query result attribute is optional. The system assigns the attribute or column name as the default name. If you provide a name, the name must consist of ASCII characters.

When the statement includes a function, arithmetic expression, or a keyword, you must specify a name if you are running against MS SQL Server or Sybase, and we recommend that you also do so if you are running against Oracle or DB2.

MS SQL Server and Sybase do not provide default names when values are functions, arithmetic expressions, or keywords. For example, suppose you issue the following statement:

```
SELECT COUNT(*) FROM "dm_document"
WHERE "subject" = 'cake'
```

Your return value would be:

```
____________
integer
```

Notice that there is no name above the integer return value.

For DB2, if you don't provide a name, the returned column is named with its position in the selected values list. For example, in the above SELECT example, the count would be returned in a column named 1.

Because the server does not allow duplicate attribute names in an object, if you assign the same name to more than one attribute, the system ignores all subsequent occurrences of the name and assigns those attributes different names.

Oracle provides a default name, but it is difficult to predict what forms the names for selected functions, arithmetic expressions, or keywords will have. Consequently, explicitly assigning a name is recommended.

The FROM Clause

The FROM clause defines which items are searched. Both the PUBLIC keyword and the source list restrict the search.

PUBLIC Keyword

The PUBLIC keyword restricts the search to objects for which the r_is_public attribute is set to TRUE. If the query contains a SEARCH clause, the full-text search is restricted to those documents for which ISPUBLIC is TRUE. When the server searches public objects only, it uses the setting of r_is_public for security checks.

Including PUBLIC increases the search performance.

Source List

A source list defines which object types and which RDBMS tables to search. The source list can include:

- One or more object types, to a maximum of 10 types
- One or more RDBMS table names

The total number of object types and tables that you can include in the FROM clause is constrained by the RDBMS. Each relational database has a limit on the number of tables that can joined in one query. In the underlying RDBMS query, for each object type in the FROM clause, all the _s tables in the type's hierarchy plus any needed indexes are candidates for inclusion in the underlying query. If the query references repeating attributes, the _r tables are included also.

For registered tables, all object types included in the view's definition are included in the query.

DQL passthrough hints included in the source list are applied only to the type or table after which the hint appears. For portability, you can include hints for multiple databases. If you include hints for multiple databases, only the hints that are applicable to the database receiving the request are used.

For more information about using passthrough hints, refer to Passthrough Hints, page 323.

Including Type Names in the Source List

Use the following syntax to specify a type name:

```
type_name [(ALL)|(DELETED)] [correlation_variable]
```

The *type name* argument identifies the object type to search. The server searches objects of the specified type and all subtypes. For example, to search all SysObjects, specify dm_sysobject. The server searches dm_sysobject and all of its subtypes, such as dm_document, dm_process, and so forth.

The keyword ALL directs the server to search all versions of each object. The keyword DELETED directs the server to search all versions of each object including any versions for which i_is_deleted is set to TRUE. (This attribute is set to TRUE if you delete the object and it is the root version of a version tree.) You must enclose ALL and DELETED in parentheses.

If the FROM clause includes neither ALL nor DELETED, the server searches only the CURRENT version.

The value of *correlation_variable* is a qualifier for the type name that is used to clarify attribute references in the query.

Any user can specify any object type in a query with one exception. The exception is the type dmi_audittrail_attrs. To specify that type, you must have either superuser or View Audit privileges.

The server searches only objects within that specified type that are owned by the user or objects to which the user has access. Whether a user has access depends on the object-level permissions of the objects. The user must have at least Browse permission on an object for the object to be included in a query run by the user. If the query includes a SEARCH clause, the user must have at least Read permission on the object. (For a full description of server security, refer to Chapter 4, Security Services, in the *Content Server Administrator's Guide*.)

Including multiple object types is subject to the following constraints:

- The selected values list cannot contain repeating attributes.
- If the selected values list contains an asterisk (*), only the first type's attributes will be expanded.
- The query cannot include a SEARCH clause.
- Unqualified attribute names in the query are disambiguated from left to right, as the object types appear in the FROM clause.
- It the query includes an IN DOCUMENT clause, the object types must join in one-to-one relationship.
- If the query includes an IN DOCUMENT or IN ASSEMBLY clause or a TYPE or FOLDER predicate, the clause or predicate is applied to only one type in the join, and that type will be the first object type identified in the FROM clause.

Including Table Names in the Source List

You can include one or more RDBMS table names in the FROM clause. The syntax is:

```
[owner_name.]table_name [correlation_variable]
```

The *owner_name* argument identifies the owner of the table. You must include the owner name if you are not the owner of the table. If the owner is the DBA of the database, you can use the alias dm_dbo as the owner name. (Using dm_dbo when appropriate makes your statement portable across databases.) If the owner name is unspecified, the current user is assumed.

You must be a superuser to include an unregistered RDBMS table in the list. Any user with appropriate permissions can include a registered table.

table_name is the name of the table that you want to search.

correlation_variable is a qualifier for the table name used to clarify column references in the query.

Clarifying Type and Table Names

Type and table names can be the same. In such instances, the server must distinguish which name identifies a type and which identifies a table. The server uses the following rules for distinguishing between type and table names:

- If (ALL) or (DELETED) is present, the name is a type name.
- If *ownername* is present, the name is a table name.
- All other cases are ambiguous references. In such cases, the server checks to see whether the name is a type name. If it is, the name is assumed to be a type.

The values in the *source_list* argument are processed from left to right. Consequently, if you include tables and types with the same name in the FROM clause, be sure to qualify the table name with its owner's name. This will ensure that the server does not mistakenly assume that the table name is the type name.

To illustrate, suppose you have an object type named legal_docs and a registered table of the same name. The following statement shows one correct way to include them both in one SELECT statement:

```
SELECT a."r_object_id", b."client"
FROM "legal_docs" a, jolenef."legal_docs" b
```

The first legal_docs (with correlation name a) is a type name. The server knows the second is a registered table because an owner name is specified. Note that correlation variables should be used in this case to clarify column references.

The following statement is valid because billing_docs is not found to be a type and so is assumed to be a registered table:

```
SELECT a."r_object_id", b."client"
FROM "dm_document" a, "billing_docs" b
```

In the following statement, the server assumes that the query references the legal_docs type, not the registered table, because no owner name is specified and there is a type called legal_docs:

```
SELECT a."object_id", b."client"
FROM "legal_docs" b, "dm_document" a
```

The IN DOCUMENT Clause

The IN DOCUMENT clause lets you use the SELECT statement to perform the following operations:

- Identify only the directly contained components of a virtual document
- Identify the indirectly contained components that reside in the current Docbase with the virtual document

  If a component is a folder that doesn't reside in the current Docbase, the query returns the object ID of the folder's mirror object in the Docbase, but not any folders or documents that the remote folder contains.
- Assemble a virtual document
- Identify the virtual documents that directly contain a particular object

If the value list for the SELECT statement contains a repeating attribute and the statement contains an IN DOCUMENT clause, the server automatically removes any duplicates of the repeating attribute.

You cannot include an ORDER BY clause when you include an IN DOCUMENT clause.

The syntax of the IN DOCUMENT clause is:

```
IN DOCUMENT object_id [VERSION version_label]
[DESCEND][USING ASSEMBLIES]
[WITH binding condition]
[NODESORT BY attribute {,attribute} [ASC|DESC]]]
```

Specifying the Virtual Document

To identify which virtual document to search, use either the document's object ID or the combination of an object ID and a version label. When you use only the object ID, the server searches the version identified by that object ID. When you use a version label in addition to an object ID, the server searches that version of the object. Note that this version may not be the same as the version identified by the object ID.

If the SELECT statement is not a subquery, specify the *object_id* using the special ID function and the actual object ID of the virtual document. For example:

```
IN DOCUMENT ID('object_id')...
```

If the SELECT statement is a subquery, you can also specify the *object_id* using a correlated r_object_id attribute reference. For example:

```
SELECT "r_object_id" FROM "dm_document" x
```

```
WHERE EXISTS
(SELECT * FROM "dm_document"
IN DOCUMENT x."r_object_id"
WHERE "object_name"='Chapt1')
```

The VERSION option lets you specify a version label. This label can be assigned to any version in the version tree that contains the specified virtual document. If you do specify a label, the server searches the specified version rather than the version identified by the object ID.

The DESCEND Option

The keyword DESCEND directs the server to search the virtual document's hierarchy, including all components directly or indirectly contained in the virtual document. Otherwise, the server searches only those components that are directly contained. You cannot use the DESCEND keyword in the IN DOCUMENT clause if the SELECT statement is a subquery.

The USING ASSEMBLIES Option

An assembly defines the components of a particular version of a virtual document at a particular time (the time the assembly was created). Creating an assembly ensures that a particular version of a virtual document always contains the same material. (Refer to Assemblies, page 148 in *Content Server Fundamentals* for information about creating assemblies.)

If a component selected for inclusion is a virtual document and the USING ASSEMBLIES clause is included, the server determines whether an assembly is defined for that component. If so, the server uses the information in the assembly as the definition of the component's composition.

Refer to Assemblies, page 148 in *Content Server Fundamentals* for a full discussion of assemblies.

The WITH Option

The WITH option lets you identify which version of a virtual document's component you want to include in the document. The syntax of the WITH option is:

`WITH` *`binding condition`*

where *binding condition* is one or more expressions that resolve to a single Boolean TRUE or FALSE. The expressions can include comparison operators, literals, attribute or column names, scalar and date functions, arithmetic expressions, and any predicates except SysObject predicates. Multiple expressions in the condition are joined using logical operators.

The condition defined by the WITH option is applied to any component of the virtual document that was added without a version being specified at the time of its addition. For example, suppose you add component C to virtual document A without specifying a particular version of component C. Later, when you assemble virtual document A, you can use the WITH clause condition to determine which version of component C to include in the document. You may want to include the version that has the word accounting as a keyword or the version that carries the symbolic label CURRENT. Each time you assemble the document, you can select a different version of component C. Choosing a version for inclusion at the time of actual assembly is called late binding. (For more information about late binding, refer to Late Binding, page 140 in *Content Server Fundamentals*.)

More than one version of a component may qualify for inclusion in the virtual document. To resolve this, you can include the NODESORT BY option in the IN DOCUMENT clause or you can allow the server to make a default choice. If you allow the server to make the choice, the server includes the version with the smallest object ID; that is, the earliest version of the component.

The NODESORT BY Option

The NODESORT BY option works in conjunction with the WITH option to identify one version of a particular component for inclusion in a virtual document. (In some instances, more than one version of a component may fulfill the condition imposed by the WITH option.) The NODESORT BY option sorts the versions by the value of one or more attributes. The server includes the first version.

The syntax of the NODESORT BY option is:

```
NODESORT BY attribute {,attribute} [ASC|DESC]
```

where *attribute* is any attribute belonging to the component. You can sort in ascending (ASC) or descending (DESC) order. If you do not specify a sort order, the default is ascending (ASC).

If you use the NODESORT BY option and two or more versions still qualify for inclusion, the server picks the version having the smallest object ID. This situation could occur if two or more versions have exactly the same values in the attributes specified in the NODESORT BY option.

The IN ASSEMBLY Clause

The IN ASSEMBLY clause allows you to identify an assembly associated with a particular document. An assembly is a snapshot of a virtual document at a particular point in time and under conditions defined when the assembly was created. Using the IN ASSEMBLY clause means that the SELECT statement queries the virtual document represented by the assembly; however, the server uses only the components found in the assembly, rather than recursively searching the virtual document's hierarchy.

You can also use this clause to identify a single component of a virtual document and any components contained by that component. The component must be contained in the assembly.

The IN ASSEMBLY clause is more flexible than the IN DOCUMENT clause. For example, you can include aggregate functions such as COUNT or MAX in the value list when you use the IN ASSEMBLY clause. You cannot include aggregate functions when a SELECT statement includes an IN DOCUMENT clause.

The syntax of the IN ASSEMBLY clause is:

```
IN ASSEMBLY [FOR] document_id [VERSION version_label]
[NODE component_id] [DESCEND]
```

where *document_id* identifies the document with which the assembly is associated. Use the document's object ID for this argument, specified as an ID literal:

```
ID('object_id')
```

Note: You can create an assembly for a virtual document and assign that assembly to another document. In such cases, the value of the *document_id* argument identifies the document to which the assembly is assigned, *not* the virtual document that the assembly represents.

If the specified document does not have an associated assembly, the statement fails with an error.

The VERSION option lets you specify a particular version of a document. If you include a version label, the server finds the version of the document carrying that label and uses the assembly associated with that version. You can specify either a symbolic label or an implicit label.

The NODE option allows you to identify a particular component of a virtual document. Use the component's object ID, specified as an ID literal:

```
ID('component_id')
```

The component must be contained in the assembly.

You cannot include the NODE option if the SELECT statement contains an aggregate function in the value list or if the SELECT statement is a subselect or subquery.

The DESCEND keyword directs the server to return not only directly contained components but also any indirectly contained components that meet the criteria in the statement. If you do not include this keyword, the server returns only directly contained components of the document or component.

The SEARCH Clause

The SEARCH clause identifies some subset of the full-text indexed documents. The syntax of the SEARCH clause is:

```
SEARCH [FIRST|LAST]fulltext_search_condition
[IN FTINDEX index_name {,index_name}]
[WITH|WITHOUT FT_OPTIMIZER]
```

FIRST and LAST Keywords

The FIRST and LAST keywords determine when the search is conducted.

If you include FIRST, the server runs the full-text search query first and then applies the SELECT statement. For example, suppose legal_documents is a subtype of dm_document. The following statement first searches the full-text index for all documents that have the word fiduciary in their content and then finds the members of that group that are also legal_documents:

```
SELECT "object_name" FROM "legal_documents"
SEARCH FIRST DOCUMENT CONTAINS 'fiduciary'
```

If you include LAST, the server first searches the Docbase for all objects that meet the SELECT criteria and then applies the criteria in the SEARCH clause. For example, the following statement first finds all legal_documents authored by G. Oliphant and then finds the members of that group that contain the word fiduciary:

```
SELECT "object_name" FROM "legal_documents"
SEARCH LAST DOCUMENT CONTAINS 'fiduciary'
WHERE ANY "authors"='G.Oliphant'
```

The default behavior is to search the full-text index first. If you include a full-text keyword in the selected values list, the full-text search is conducted first even if LAST is specified in the SEARCH clause.

The Fulltext Search Condition

The *fulltext_search_condition* argument has two valid formats:

```
DOCUMENT CONTAINS [NOT] word {AND|OR [NOT] word}
```

and

```
TOPIC 'topic_query_string'
```

- The DOCUMENT CONTAINS format lets you specify one or more *words* on which to conduct the full-text index search. A *word* can be any character string that does not include spaces or punctuation. For example, the following statement finds all indexed documents that contain the word yeast:

```
SELECT * FROM "dm_documents"
SEARCH DOCUMENT CONTAINS 'yeast'
```

Multiple words are joined together using the logical operators AND and OR. (For more information about logical operators and the order of precedence, refer to Chapter 1, DQL Language Elements.)

- The TOPIC format lets you specify a query in the Boolean Plus Language of Verity. Use this format also to query indexed attributes. When the server receives a Verity query as a *fulltext_search_condition*, it passes the query to the Full-Text Engine. (For information about how the query syntax when querying indexed attributes, refer to Querying Indexed Attributes, page 277 in *Content Server Fundamentals*.)

  You cannot specify the FREETEXT or LIKE operators, nor can you include references to previous queries. However, all other Boolean Plus Language operators are allowed.

  Warning: Documentum provides the passthrough syntax but does not provide support for the Boolean Plus Language beyond what is described in Querying Indexed Attributes, page 277. If you are knowledgable about Verity and can construct the query, you can use the passthrough mechanism.

The IN FTINDEX Option

The IN FTINDEX option allows you to identify which index (or indexes) to search. Identify each index by its name, which is the name of its fulltext index object. If the index name begins with a digit, enclose the name in single quotes. You can specify a searchable or standby index.

If you do not specify an index, the statement searches all indexes visible to the server.

The FT_OPTIMIZER Option

Including WITH FT_OPTIMIZER turns on optimization for full-text searches as long as the following conditions are true:

- All attributes included in the value list and the WHERE clause are indexed attributes or full-text keywords.
- If a WHERE clause is included, the server must be able to convert the clause to a full-text search qualification.

A WHERE clause must meet the following conditions to be converted to a full-text search clause:

- The clause can reference only document object types or subtypes.
- The clause cannot reference a registered table or any repeating attributes.
- The clause can reference only single-valued attributes that are indexed in the full-text index.
- The clause can contain only one simple expression—compound expressions are not allowed. The expression cannot include:
  - Aggregate, scalar, or date functions
  - The ID function
  - Any repeating attribute predicate
  - Arithmetic operators

- — A quantified predicate of the format ANY | ALL | SOME (subquery)
- — Any of the following predicates for single-valued attributes: EXISTS, IN, BETWEEN, NOT LIKE, LIKE with an ESCAPE clause, and ONLY
- — The FOLDER or TYPE predicate
- — A subquery
- — A UNION clause
- — The ID function
- — The ORDER, ISREPLICA, or USER keywords

WITH FT_OPTIMIZER is on by default if a SEARCH clause is part of the SELECT statement and you do not specify WITHOUT FT_OPTIMIZER. If you do specify WITHOUT FT_OPTIMIZER, the WHERE clause is not converted to a full-text search query even if it satisfies the constraints.

Notes:

- Including the keyword PUBLIC in the FROM clause provides additional optimization. Only public documents are searched. Refer to The FROM Clause, page 122 for details.
- If WITH FT_OPTIMIZER is on and the query does not contain a GROUP BY, HAVING, or ORDER BY clause or the DISTINCT keyword and PUBLIC is not specified, the query searches public documents first and then non-public documents.

The WHERE Clause

Use the WHERE clause to restrict what objects are returned. For example, the following statement selects the title of every document but only returns the titles of documents owned by the current user:

```
SELECT "title" FROM "dm_document"
WHERE "user_name" = USER
```

The syntax of the WHERE clause is:

```
WHERE qualification
```

The *qualification* argument consists of one or more expressions that resolve to a single Boolean TRUE or FALSE. The expressions can include comparison operators, literals, attribute or column names, scalar and date functions, arithmetic expressions, predicates, and full-text keywords. Multiple expressions are joined together using the logical operators. (Chapter 1, DQL Language Elements, contains descriptions of all accepted comparison and arithmetic operators, predicates, functions, and logical operators. Full-Text Keywords are described in Full-Text Keywords, page 117.)

A qualification cannot reference any of the following attributes of audit trail objects unless you have Superuser or View_audit privileges:

- acl_name
- acl_domain
- attribute_list
- attribute_list_id
- chronicle_id
- object_name
- object_type
- owner_name
- session_id
- version_label

The qualification can be as complex as necessary. For example, the following WHERE clause is simple:

```
SELECT * FROM "dm_workflow"
WHERE "supervisor_name" = 'horace'
```

This next example contains a more complex qualification:

```
SELECT "r_object_id", "title," FROM "dm_document"
WHERE "r_creation_date" >= DATE('01/01/99','mm/dd/yy')
AND "r_modify_date" <= NOW
```

If the qualification references a repeating attribute, you must include the ANY keyword. For example, the following query selects the title and subject of all documents that have gillian as an author:

```
SELECT "r_object_id","title","subject" FROM "dm_document"
WHERE ANY "authors" IN ('gillian')
```

A qualification can also include a subquery:

```
SELECT "r_object_id"  FROM "dm_document"
WHERE ANY "authors" IN
(SELECT "user_name" FROM "dm_user"
 WHERE "r_is_group" = TRUE)
```

If you are referencing a repeating attribute and including a subquery, you can use the IN or EXISTS keyword after the ANY keyword to control the generated SQL query. The syntax is:

```
WHERE ANY [IN|EXISTS] attr_name IN subquery
```

Including IN or EXISTS may change the generated SQL query and consequently, enhance performance. IN and EXISTS, page 322 contains an example that shows the differences between the options in the generated query.

For more information about referencing repeating attributes in WHERE clause qualifications, refer to Repeating Attributes in Queries, page 269 in *Content Server Fundamentals*.

Under certain conditions, a WHERE clause can be converted into a full-text search clause to provide performance optimization for the SELECT statement. Refer to The FT_OPTIMIZER Option, page 129 for complete information about writing WHERE clauses that can be optimized for a full-text search.

The GROUP BY Clause

The GROUP BY clause groups results. Use it when you include a function in the value list for the SELECT statement. The *value_list* argument for the GROUP BY clause must contain all of the values in the value list for the SELECT statement that are not aggregate function values.

For example, the following statement returns the names of all documents, their owners, and a count of the documents that each owner owns. The results are grouped by owner name:

```
SELECT "owner_name", count(*)
FROM "dm_document"
GROUP BY "owner_name"
```

This next example selects the names of all documents, their owners, and their subjects, and groups them first by owner and then, within each owner group, by subject:

```
SELECT "owner_name", "subject", count (*)
FROM "dm_document"
GROUP BY "owner_name", "subject"
```

When you include a GROUP BY clause in a SELECT statement, you cannot include the FT_OPTIMIZER clause.

The HAVING Clause

The HAVING clause restricts which groups are returned. It is most commonly used in conjunction with the GROUP BY clause.

For example, the following statement returns owner names and the number of documents each owner owns, grouped by owner names, for all owners having more than 100 documents:

```
SELECT "owner_name", count(*)
FROM "dm_document"
GROUP BY "owner_name"
HAVING count(*) > 100
```

This statement first finds and counts all documents for each owner name. It returns only those groups (owner's name and count) for which the count is greater than 100.

You can also use the HAVING clause in a SELECT statement that does not include the GROUP BY clause when a value in the value list is an aggregate function. For example, the following statement returns a count of the workflows supervised by haroldk if that count is greater than or equal to 25:

```
SELECT COUNT(*) FROM "dm_workflow"
WHERE "supervisor_name" = 'haroldk'
HAVING COUNT (*) >=25
```

If the number of workflows supervised by haroldk is less than 25, the statement returns nothing.

Note that if you do not include a GROUP BY clause when you use a HAVING clause, the only value permitted in the value list for the SELECT statement is one aggregate function.

The UNION Clause

The UNION clause lets you obtain results from more than one SELECT statement. The syntax is:

UNION *dql_subselect*

The *dql_subselect* argument must return the same number of attributes or columns as the first SELECT statement, and each attribute or column must have the same datatype as its corresponding attribute or column in that SELECT statement. For example, if the first SELECT statement returns three attributes, then the subselect argument in the UNION clause must also return three attributes. The datatypes of the first attributes returned by each must be the same, as must the datatypes of the second and third attributes.

Neither the first SELECT statement nor any subsequent UNION SELECT statements can contain an IN DOCUMENT clause. The IN ASSEMBLY clause can be included only if the clause is in the first SELECT statement and all unioned subselect statements.

For all databases except DB2, when you union two or more SELECT statements, the attribute names that are returned are derived from the first SELECT statement. For example, suppose you issue the following statement:

```
SELECT "name", "address" FROM "current_emp"
UNION SELECT "ret_name", "ret_address" FROM "retired_emp"
```

The query result objects for this statement have two attributes, name and address, taken from the value list of the first SELECT in the statement.

For DB2 , if corresponding selected values in the unioned SELECT statements don't have the same name or alias, the returned attribute name is an number representing the attribute's position in the selected values list. For example, suppose you issue the following query:

```
SELECT "name", "address" FROM "current_emp"
UNION SELECT "ret_name", "ret_address" FROM "retired_emp"
```

The query result objects for this statement have two attributes. The first attribute is named 1, representing selected values from name and ret_name. The second attribute is named 2, representing selected values from address and ret_address.

The server does not return duplicate rows when you union two or more SELECT statements. This is true even if the ALL keyword is included in the first SELECT statement.

The ORDER BY Clause

Use the ORDER BY clause to sort the results of the SELECT statements. The syntax of this clause is:

```
ORDER BY value [ASC|DESC] {,value [ASC|DESC]}
```

The *value* argument must be an attribute or column name that appears in the value list. You can sort in ascending (ASC) or descending (DESC) order. If you do not specify a sort order, the default is ascending.

The primary sort is on the first value specified, the secondary sort is on the second value specified, and so forth. To illustrate, the following statement returns the owner name, subject, and title of all documents in the system in ascending order by owner name and the subject:

```
SELECT "owner_name", "subject", "title"
FROM "dm_document"
ORDER BY "owner_name", "subject"
```

You can specify a value either explicitly, by name, or by position. For example, the following statement returns the same results as the previous example, but notice that the ORDER BY clause specifies values by their position in the SELECT value list:

```
SELECT "owner_name", "subject", "title"
FROM "dm_document"
ORDER BY 1, 2
```

The only exception occurs when you union two or more SELECT statements. In such cases, you can only specify a value by its position in the value list.

When you include an ORDER BY clause in a SELECT statement, you cannot include the FT_OPTIMIZER clause.

Including DQL Hints

You can include processing hints for DQL and the RDBMS servers in SELECT statements. The hints for DQL are called standard hints and can be added at the end of the SELECT statement, using the ENABLE keyword. Hints to the RDBMS server are called passthrough hints and can be added in the FROM clause, after a table or type name, and at the end of the statement, using the keyword ENABLE.

Table 2–20, page 134 lists the standard hints and provides brief guidelines for their use. For detailed information about their implementation, refer to Appendix A, Using DQL Hints .

Table 2–20. DQL Standard Hints

Standard Hint	Description
SQL_DEF_RESULT_SET *N*	Defines a maximum number (*N*) of rows to return. Set *N* to a positive integer number. If set to 0, all rows are returned. **Guidelines** • On a SQL Server database, this hint forces the use of default result sets instead of a cursor to return the rows. On all other databases, the hint only sets the number of rows to return. • It is recommended that you set *N* to a maximum limit, rather than specifying it as 0. • If you include SQL_DEF_RESULT_SET and other hints that control the number of rows returned, the last one listed in the hints is used.
FORCE_ORDER	Controls the join order for the tables and types listed in the source list. If this hint is included, the tables are joined in the order they are listed. **Guideline** • This hint is ignored on DB2.

Standard Hint	Description
RETURN_TOP *N*	Limits the number of results returned by a query. Set *N* to a positive integer number. **Guidelines** • If you include RETURN_TOP and other hints that control the number of rows returned, the last one listed in the hints is used. • This hint is useful because it can limit the effect of a bad (unbounded) query on the database. • Sorting the results or including the DISTINCT keyword reduces or eliminates the benefits of using RETURN_TOP. • On a SQL Server database, this hint reduces the number of rows touched by the query. • On a DB2 database, using OPTIMIZE_TOP with RETURN_TOP is recommended.
OPTIMIZE_TOP *N*	Returns *N* rows very quickly, then continues with the remainder of the rows. Set *N* to positive integer. **Guidelines** • This hint is most useful in conjunction with RETURN_TOP. • On an Oracle database, *N* is ignored. • This hint is ignored on a Sybase database.
FETCH_ALL_RESULTS *N*	Fetches all results from the cursor immediately and then closes the cursor. Set *N* to a positive integer number or 0 to return all rows. **Guideline** • If you include FETCH_ALL_RESULTS and other hints that control the number of rows returned, the last one listed in the hints is used.

Examples

The following example returns the names and titles of all documents owned by the current user:

```
SELECT "object_name", "title" FROM "dm_document"
WHERE "owner_name" = USER
```

The next example selects all documents and their authors with the subject employee_benefits:

```
SELECT "r_object_id", "authors" FROM dm_document
WHERE "subject" = ’employee_benefits’
ORDER BY 1
```

The following example returns the names of all objects in the New Books cabinet:

```
SELECT "object_name" FROM "dm_sysobject"
WHERE CABINET ('/New Books')
ORDER BY "object_name"
```

There are additional examples demonstrating the use of specific clauses in the descriptions of the clauses.

Unregister

Purpose

Removes a registered table from the Docbase.

Syntax

```
UNREGISTER [TABLE] [owner_name.]table_name
```

Syntax Description

Table 2–21. UNREGISTER Syntax Description

Variable	Description
owner_name	Identifies the table's owner. The default value is the current user. Use the owner's RDBMS user name. If the owner is a Docbase user, this value is found in the user's user_db_name attribute. If the RDBMS is Oracle and the owner is the DBA, you can use the alias dm_dbo. If the RDBMS is MS SQL Server or Sybase and if the owner is the DBA, you can use the alias dbo.
table_name	Identifies a registered table to remove from the Docbase. Use the table name as it appears in the RDBMS.

Permissions

You must be the owner of the registered table or have Superuser privileges to unregister a table.

General Notes

Unregistering a table removes the object of type dm_registered that represents the table in the Docbase. It does not remove the table from the underlying RDBMS.

If you attempt to unregister a table that is not registered, you receive an error message.

Related Statements

Delete, page 82

Insert, page 99
Register, page 102

Example

The following example unregisters the table called departments:

```
UNREGISTER TABLE "departments"
```

Update

Purpose

Updates the rows of a registered table.

Syntax

```
UPDATE table_name SET column_assignments
[WHERE qualification]
```

Syntax Description

Table 2–22. UPDATE Syntax Description

Variable	Description
table_name	Identifies the registered table to update. (Refer to Register, page 102 for information about registering tables in the Docbase.)
SET clause	Specifies the columns to update and the new values to assign to them. The syntax for *column_assignments* is: `column_name = value expression` Refer to Identifying Columns to Update, page 140 for a full description of the syntax.
WHERE clause	Restricts the rows that are updated to those that meet the criteria in the *qualification*. Refer to The WHERE Clause, page 130 for a full description of WHERE clauses and qualifications.

Return Value

The UPDATE statement returns a collection whose result object has one attribute, rows_updated, that contains the number of updated rows.

Permissions

To use the UPDATE statement, the following conditions must be true:

- Your object-level permission for the dm_registered object in the Docbase that represents the RDBMS table must be at least Browse
- Your table permission for the dm_registered object representing the table must be DM_TABLE_UPDATE
- The user account under which Content Server is running must have the appropriate

RDBMS permission to update the specified table. (The actual name of this permission depends on your RDBMS.)

(For more information about security, object-level and table permissions, refer to Chapter 12, Protecting Docbase Objects, in the *Content Server Administrator's Guide*.)

General Notes

The SET clause identifies which columns are updated and the WHERE clause determines which rows are updated. The server searches for all rows that meet the qualification and updates the specified columns. If a WHERE clause is not included, then all rows are updated.

Identifying Columns to Update

In the SET clause, *column_assignment* has the format:

```
column_name = value_expression
```

where *column_name* is the name of a column in the table.

The *value_expression* can be a simple or complex expression but must resolve to a single value. It can include literal values, other column names, special keywords, date and scalar functions, and arithmetic expressions. The resulting value must be appropriate for the datatype of the specified column.

Indentifying Rows to Update

The WHERE clause *qualification* determines which rows are updated. The *qualitication* consists of one or more expressions that resolve to a single Boolean TRUE or FALSE. The server updates only those rows for which the qualification is TRUE. The expressions in a qualification can include comparison operators, literals, scalar and date functions, column names, arithmetic expressions, and column predicates. Multiple expressions are joined together using logical operators.

Related Statements

Delete, page 82
Insert, page 99
Register, page 102
Unregister, page 137

Example

This example updates the column called chef_name:

```
UPDATE "recipes" SET "chef_name" = 'carol'
WHERE "chef_name" = 'carole'
```

Update...Object

Purpose

Updates an object in the Docbase.

Syntax

```
UPDATE [PUBLIC]type_name [(ALL)][correlation_var]OBJECT[S] update_list
[,SETFILE filepath WITH CONTENT_FORMAT=format_name]
{,SETFILE filepath WITH PAGE_NO=page_number}
[IN ASSEMBLY document_id [VERSION version_label]
[NODE component_id][DESCEND]]
[SEARCH fulltext search condition]
[WHERE qualification]
```

Syntax Description

Table 2–23. UPDATE...OBJECT Syntax Description

Variable	Description
type_name	Identifies the type of the object that you want to update. Valid values are: dm_assembly and its subtypes dm_user and its subtypes dm_group and its subtypes dm_relation_type dm_sysobject and its subtypes If you have Sysadmin or Superuser user privileges, you can also update the dmr_content object type.
correlation_var	Defines a qualifier for the type name that is used to clarify attribute references in the statement.
update_list	Specifies the update operations to perform on the object. Valid formats are: `set attribute_name = value set attribute_name[[index]] = value append [n] attribute_name = value insert attribute_name[[index]] = value remove attribute_name[[index]] truncate attribute_name[[index]] [un]link 'folder path' move [to] 'folder path'` If you specify more than one operation, use commas to separate them.
SETFILE clause WITH CONTENT_ FORMAT option	Adds the first content file to the new object. Refer to WITH CONTENT_FORMAT Option, page 144 for details.

Variable	Description
SETFILE clause WITH PAGE_NO option	Adds additional content to the object. Refer to WITH PAGE_NO Option, page 144 for details.
IN ASSEMBLY clause	Restricts the statement to objects in a particular assembly. *document_id* identifies the document with which the assembly is associated. Use a literal object ID: `ID('object_id')` *version_label* specifies a particular version of the document. Use a symbolic or an implicit version label. If you do not include a version label, the server uses the version identified by the *document_id* argument. *component_id* specifies a particular component in the assembly. Including the NODE option restricts the statement to the specified component. Use a literal object ID to identify the component: `ID('object_id')` The DESCEND keyword directs the server to update not only all directly contained node components, but also any indirectly contained components that meet the criteria. If you do not include this keyword, the server updates only the directly contained components that meet the criteria in the statement.
SEARCH clause	Restricts the statement to objects that meet the SEARCH clause *fulltext search condition*. Refer to The SEARCH Clause, page 128 for a detailed description of a SEARCH clause.
WHERE clause	Restricts the statement to objects that meet the *qualification*. The qualification is an expression that evaluates to TRUE or FALSE. It can include comparison operators, literals, attribute names, scalar and date functions, special keywords, predicates, and arithmetic expressions. Multiple expressions can be joined together using logical operators. Refer to The WHERE Clause, page 130 for a detailed description of the clause.

Return Value

The UPDATE...OBJECT statement returns a collection whose result object has one attribute, called objects_updated, that contains the number of objects updated.

Permissions

To update an object, you must have Write permission on the object.

To include the SETFILE clause in the statement, you must have Superuser privileges.

General Notes

To update an object, the object cannot belong to a frozen assembly or a frozen or immutable virtual document.

The *type_name* argument specifies the type of objects to update. The server searches all objects of the specified type and any subtypes for objects that meet any additional criteria you define in the statement.

The keyword PUBLIC restricts the query to those objects with the r_is_public attribute set to TRUE. If the query contains a SEARCH clause, the full-text search is restricted to those documents for which ISPUBLIC is TRUE. When the server queries or searches only public objects, it uses only the setting of r_is_public for security checks.

The keyword ALL directs the server to consider all versions of each object. If you do not include ALL, the server only considers the version with the symbolic label CURRENT. You must enclose ALL in parentheses.

The IN ASSEMBLY, SEARCH, and WHERE clauses restrict the scope of the statement. The IN ASSEMBLY clause restricts the operation to a particular virtual document assembly or node (component) within the virtual document. The SEARCH clause restricts the operations to indexed objects that meet the fulltext search condition. The WHERE clause restricts the operations to objects that meet the specified qualification. The clauses are applied in the following order:

- SEARCH
- IN ASSEMBLY
- WHERE

If you do not include any of these clauses, the updates are applied to all objects of that type for which you have Write permission.

If any of the objects that are otherwise eligible for updating belong to a frozen assembly or an unchangeable virtual document, then the entire statement is rolled back and no objects are updated.

The SETFILE Clauses

A SETFILE clause adds new content to the end of the sequence of content files in the object or replaces an existing content file. You cannot use SETFILE to insert a content file between two current files. You cannot store the content file you add in a turbo store storage area.

You must have Superuser privileges to include the clause in the statement.

Any object capable of having content may have multiple associated content files. All files must have the same content format and be ordered within the object.

The content format is defined when you add the first content file to the object. All subsequent additions must have the same format. Consequently, specifying the format for content additions after the first file is not necessary. Instead, you must specify the content's position in the ordered list of content files for the object.

To add the first content, use the SETFILE clause with the WITH CONTENT_FORMAT option.

To add additional content, use the SETFILE clause with the PAGE_NO option.

You can't include both options in a single SETFILE clause.

WITH CONTENT_FORMAT Option

Use this SETFILE option to add the first content file to an object. The syntax is:

```
SETFILE 'filepath' WITH CONTENT_FORMAT='format_name'
```

The filepath must identify a location that is visible to Content Server.

The format name is the name found in the name attribute of the format's dm_format object.

WITH PAGE_NO Option

Use this SETFILE option to add additional content to an object or replace existing content. The syntax is:

```
SETFILE 'filepath' WITH PAGE_NO=page_number
```

The filepath must identify a location that is visible to Content Server. The file must have the same file format as the existing content associated with the object.

The page number identifies the file's position of the content file within the ordered contents of the new object. You must add content files in sequence. For example, you cannot add two files and specify their page numbers as 1 and 3, skipping 2. Because the first content file has a page number of 0, page numbers for subsequent additions begin with 1 and increment by 1 with each addition.

To replace a content file, specify the page number of the file you want to replace.

The Update Operations

The *update_list* defines the operations to perform. You can:

- Set the value of a single-valued or repeating attribute
- Add a value for a repeating attribute
- Insert a value into the list of values for a repeating attribute
- Remove a value from a repeating attribute
- Truncate a repeating attribute (remove all values)
- Link an object to a folder or cabinet or unlink an object from a folder or cabinet
- Move an object to a new folder or cabinet

Unless you have Superuser user privileges, you can only update read/write attributes. These attributes have names beginning with a_ or have no prefix at all. If you have Superuser user privileges, you can update read-only attributes. These attributes have names beginning with r_.

You can specify more than one update operation in a single statement. When an UPDATE...OBJECT statement includes multiple operations, use commas to separate them. For example, the following statement sets two attributes and inserts a value into a repeating attribute. Notice the commas at the end of each update operation clause:

```
UPDATE "dm_document" OBJECTS
SET "title" = 'A Cake Primer',
SET "subject" = 'cake',
INSERT "authors"[3] = 'georgette'
WHERE "r_object_id" = '090007354140004e'
```

The server processes the update operations in the order listed.

Setting an Attribute Value

To set the value of a single-valued or repeating attribute, use the following syntax:

```
set attribute_name[[index]] = value
```

The *attribute_name* argument is the name of the attribute. If the attribute is a repeating attribute, the *index* identifies the position of the new value in the attribute's list of values. The positions of the values for a repeating attribute are numbered from zero. Enclose the *index* value in square brackets.

The *value* argument is the value to assign to the attribute. The value can be a literal or a subquery. If it is a subquery, it cannot return more than one row. If it returns more than one row, the statement returns an error.

Appending a Value to a Repeating Attribute

To append a value to a repeating attribute, use the following syntax:

```
append [n]attribute_name = value
```

The *attribute_name* argument is the name of the attribute to which to append a new value. The *value* argument specifies what value to add to the attribute's ordered list of values. The value is automatically added to the end of the repeating attribute's list of values.

The value can be either a literal value or a subquery. The subquery can return any number of rows. By default, the system appends a maximum of 20 rows. To override the default, use the [*n*] option to define how many rows to append. You can use any integer number (to append that number of rows) or an asterisk (*) (to append all rows).

Inserting a Value into a Repeating Attribute

To insert a value into the list of values for a repeating attribute, use the following syntax:

```
insert attribute_name[[index]] = value
```

The *attribute_name* argument identifies the attribute. The *index* argument defines where to insert the new value. The positions of all values for a repeating attribute are numbered from zero. If you do not include the index, the server automatically inserts the new value in position zero, as the first element in the ordered list. You must enclose the index in square brackets.

The *value* defines the value that you want to insert. The *value* can be either a literal value or a subquery. If it is a subquery, it cannot return more than one row. If it returns multiple rows, the statement returns an error.

When you insert a value, all values that follow the inserted value are renumbered. For instance, when you insert a value at position [5], the value formerly at position [5] is moved up to position [6]. Consequently, if you are inserting more than one value in the attribute (for example, if the statement is in a program loop), be sure to increase the index number each time you insert a value.

Removing Values from Repeating Attributes

To remove a value from a repeating attribute, use the following syntax:

```
remove attribute_name[[index]]
```

The *attribute_name* argument identifies the attribute. The *index* argument indicates the position of the value to remove in the attribute's ordered list of values. The positions of all values for a repeating attribute are numbered from zero. If you do not include the index, the server removes the first value (position 0) in the attribute. Enclose the index in square brackets.

When you remove a value, the remaining values are renumbered. For instance, when you remove the value at position [5], the value formerly at position [6] is moved up to position [5]. Consequently, if you are removing more than one value in the attribute (for example, if the statement is in a program loop), be sure to start with the highest index number and decrement the index number each time you delete a value.

For example, if you want to remove the values at positions 6 and 7, remove 7 first and then 6. If you remove 6 first, the value at 7 is moved into position 6 and the value at 8 is moved into position 7. When you remove 7, you are actually removing the value formerly in the position 8.

Truncating a Repeating Attribute

To truncate a repeating attribute (remove its values), use the following syntax:

```
truncate attribute_name[[index]]
```

The *attribute_name* argument identifies the attribute. The *index* argument specifies where in the attribute's list of values to begin the truncation. For example, if you specify attribute_name[4], the server removes all values beginning with attribute_name[4]. If you do not specify an index level, the server removes all values.

Linking or Unlinking an Object

To link an object to a folder or cabinet, use the following syntax in the update_list:

```
link 'folder path'
```

Linking an object adds a new link for the object. Current links are not affected.

To unlink an object from a cabinet or folder use the following syntax:

```
unlink 'folder path'
```

Unlinking removes only the specified link. Other links are not affected.

The *folder path* argument specifies the folder or cabinet to which you are linking the object. A folder path has the following format:

```
cabinet_name{/folder_name}
```

Moving an Object to a New Folder or Cabinet

To move an object to a new folder or cabinet, use the following syntax:

```
move [to] 'folder path'
```

The *folder path* argument specifies the folder or cabinet to which to link the object. A folder path has the following format:

```
cabinet_name{/folder_name}
```

Moving an object removes all current links and adds a link to the specified cabinet or folder.

Related Statements

Change...Object, page 59
Create...Object, page 65
Delete...Object, page 84

Examples

The following statement deletes all indexed documents that contain the word *yeast*:

```
UPDATE "dm_document" OBJECTS
SET "keywords" = 'yeasted'
SEARCH DOCUMENT CONTAINS 'yeast'
```

This next statement updates all documents that contain the word *yeast* but not the word *rolls*:

```
UPDATE "dm_document" OBJECTS
SET "keywords" = 'pastries'
SEARCH DOCUMENT CONTAINS 'yeast' AND NOT 'rolls'
```

The following statement updates all cabinets that have either Janine or Jeremy as their owner:

```
UPDATE "dm_cabinet" OBJECTS
SET "is_private" = TRUE
WHERE "owner_name"='janine' OR owner_name='jeremy'
```

Chapter 3

Administration Methods

Administrative methods are methods that perform a variety of administrative and monitoring tasks. (Table 2–14, page 92 lists the methods by task categories.) They are executed in an application by invoking either the DQL EXECUTE statement or the API Apply method. You can also execute them interactively through Documentum Administrator.

This chapter first describes how to invoke the methods. Then it provides an alphabetical reference to the methods. For each method, it provides:

- Invoking syntax
- Arguments
- Return Value
- Permissions
- General Notes
- Related Administration Methods
- Examples

Invoking Administration Methods

To execute an administration method manually, use Documentum Administrator.

To execute an administration method in an application, use either the API Apply method or the DQL EXECUTE statement.

You can also use Apply or EXECUTE to invoke an administration method through IAPI or IDQL if for any reason you can't access Documentum Administrator.

Using Documentum Administrator

All the methods (except DO_METHOD, SYNC_REPLICA_RECORDS, WEBCACHE_PUBLISH, and ROLES_FOR_USER) are available through the Methods facility in Docbase Management. Many of the methods are also available through category-specific pages. For example, you can run UPDATE_FTINDEX to update a fulltext index through the Methods facility or through the Fulltext Index management facilities.

Note: Because DO_METHOD is used to execute user-defined procedures, this method is implemented as part of the Attributes page for user-defined methods.

Using the EXECUTE Statement

The EXECUTE statement is the DQL equivalent of the API Apply method. You can use it to execute any of the administration methods except PING or WEBCACHE_PUBLISH.

The EXECUTE statement is not case sensitive. You can enter the administration method name and argument names in uppercase, lowercase, or any combination.

You must enclose character string values (including an object ID in a FOR clause) in single quotes when they are included in the EXECUTE statement.

Using the Apply Method

You can use the Apply method to invoke any of the administration methods. The Apply method is case sensitive. The administration method name, argument names, and datatypes must be in uppercase.

Scope of the Administration Methods

Administrative methods execute in the context of the current docbase scope. You cannot connect to a Docbase and execute an administration method against a remote Docbase.

Executing Full-text Methods

Do not execute two or more of the following full-text methods simultaneously on the same index:

- CLEAN_FTINDEX
- CLEAR_PENDING
- DUMP_FTINDEX
- MARK_ALL
- MARK_FOR_RETRY
- RESET_FTINDEX
- RMSTYLE_FTINDEX
- SETSTYLE_FTINDEX
- UDPATE_FTINDEX

Additionally, before you manually invoke one of these methods on an index, you must inactivate the FullTextMgr administration tool. This tool runs every five minutes by default. To avoid conflict, inactivate it before running the method.

Administration Method Operations

Table 3–1, page 151 lists the administration methods by category and describes the operation that you can perform with each method.

Table 3–1. Administration Methods by Category

Category of Operation	Administration Method	Description
Process Management	BATCH_PROMOTE, page 158	Promotes multiple objects to their next lifecyle states. **Note:** This method is not available through Documentum Administrator or using DQL EXECUTE.
	CHECK_SECURITY, page 166	Checks a user or group's permissions level for one or more objects.
	GET_INBOX, page 201	Returns items in user's Inbox.
	MARK_AS_ARCHIVED, page 239	Sets the i_is_archived attribute of a dm_audittrail, dm_audittrail_acl, or dm_audittrail_group to T.
	PURGE_AUDIT, page 252	Removes audit trail entries from the Docbase.
	ROLES_FOR_USER, page 278	Returns the roles assigned to a user in a particular client domain.
Execute methods	DO_METHOD, page 180	Executes system-defined procedures such as lpq or who or user-defined procedures.
	HTTP_POST, page 211	Directs the execution of a method to an application server.
Content storage management	CAN_FETCH, page 160	Determines whether content in a distributed storage area component can be fetched by the server.

Category of Operation	Administration Method	Description
	CLEAN_LINKS, page 170	Provides maintenance for linked store storage areas. On UNIX, this method cleans up unneeded linkrecord objects and directories and links associated with linked storage areas. On Windows, this method cleans up unneeded linkrecord objects and resets file storage object security.
	DELETE_REPLICA , page 176	Removes a replica from a distributed storage area.
	DESTROY_CONTENT, page 178	Removes a content object and its associated file from the Docbase. (Do not use this for archiving; use PURGE_CONTENT instead.)
	GET_FILE_URL, page 197	Returns the URL to a content file.
	GET_PATH, page 205	Returns the path to a particular content file in a particular distributed storage area component.
	IMPORT_REPLICA, page 216	Imports an external file as a replica of content already in the Docbase.
	MIGRATE_CONTENT, page 242	Moves content files from one storage area to another.
	PURGE_CONTENT, page 260	Deletes a content file from a storage area. (Used as part of the archiving process.)
	PUSH_CONTENT_ ATTRS, page 261	Sets the content metadata in a content-addressed storage system for a document stored in that storage system.
	REGISTER_ASSET, page 264	Queues a request for the creation of a thumbnail, proxies, and metadata for a rich media content file. The request is queued to the Media Server. This method is only available or useful if you have Documentum Media Services running.
	REPLICATE, page 269	Copies content in one component of a distributed storage area to another area.

Category of Operation	Administration Method	Description
	RESTORE_CONTENT, page 273	Moves a file or files from archived storage to the original storage location.
	SET_CONTENT_ATTRS, page 283	Sets the content_attr_name and content_attr_value attributes in the content object associated with the content file.
	SET_STORAGE_STATE, page 290	Sets the state of a storage area to off-line, on-line, or read-only.
	SYNC_REPLICA_ RECORDS, page 297	Synchronizes the replica record objects with the current definition of a distributed storage area. **Note:** This method is not available through Documentum Administrator.
	TRANSCODE_ CONTENT, page 300	Queues a request for a content transformation to the Media Server. This method is only available or useful if you have Documentum Media Services running.
Database Methods	DB_STATS, page 174	Provides database operation statistics for a session.
	DROP_INDEX, page 188	Drops an index.
	EXEC_SQL, page 194	Executes SQL statements.
	FINISH_INDEX_ MOVES, page 196	Completes an interrupted move operation for an object type index.
	MAKE_INDEX, page 235	Creates an object type index.
	MOVE_INDEX, page 249	Moves an object type index from one tablespace or segment to another. **Note:** This is not supported on DB2.
	REORGANIZE_TABLE, page 266	Reorganizes a database table for query performance.
	UPDATE_STATISTICS, page 309	Updates the statistics in a database table.
Full-Text Methods	ADOPT_FTINDEX, page 156	Swaps attribute values between fulltext index objects, effectively making specified standby index the searchable index for a storage area.
	CHECK_FTINDEX, page 164	Reports on the status of the most recent index update.

Category of Operation	Administration Method	Description
	CLEAN_FTINDEX, page 168	Removes unwanted files and entries from an index. Unwanted files are temporary files that are created as part of the updating process and old versions of the index. Unwanted entries are entries associated with deleted documents.
	CLEAR_PENDING, page 172	Recovers from an update that cannot be completed by marking all the content in the update as bad.
	DUMP_FTINDEX, page 190	Creates a dump file containing a specified full-text index (supports shadow indexes).
	ESTIMATE_SEARCH, page 192	Returns the number of results matching a particular SEARCH condition.
	GET_FTINDEX_SIZE, page 199	Returns the size, in bytes, of an index.
	GETSTYLE_FTINDEX, page 208	Retrieves the topic set or style file associated with an index.
	LOAD_FTINDEX , page 230	Loads a dump file created by DUMP_FTINDEX into an index (supports shadow indexes).
	MARK_ALL, page 237	Finds all content objects that are not indexed but should be (the a_full_text attribute set for the associated object is set to T) and marks these content object for indexing.
	MARK_FOR_RETRY, page 240	Finds all content objects that have a particular negative update_count and marks them as awaiting indexing.
	MODIFY_TRACE, page 247	Sets the tracing level for full-text indexing operations.
	RESET_FTINDEX, page 271	Reinitializes an index. Use when adding or dropping indexed attributes or to reinitialize a corrupted index.
	RMSTYLE_FTINDEX, page 275	Removes a topic set or a user-defined style file.
	SETSTYLE_FTINDEX, page 292	Associates a specified topic set or style file with an index.

Category of Operation	Administration Method	Description
	UPDATE_FTINDEX, page 303	Adds new entries to a full-text index and marks for removal entries associated with deleted documents.
Session Management	CHECK_CACHE_ CONFIG, page 161	Requests a consistency check on a particular cache config object.
	GET_LAST_SQL, page 204	Returns the last SQL statement issued.
	GET_SESSION_DD_ LOCALE, page 207	Returns the locale in use for the current session.
	LIST_AUTH_PLUGINS, page 218	Lists the authentication plugins loaded by Content Server.
	LIST_RESOURCES, page 220	Provides information about the server operating system environment.
	LIST_SESSIONS, page 224	Provides information about current, active sessions.
	LIST_TARGETS, page 228	Lists the DocBrokers defined as targets for the server. The information is returned in a collection with one result object whose attributes list the DocBrokers defined as targets for the server.
	LOG_ON, page 233 and LOG_OFF, page 232	Turn server logging of information about RPC calls on or off.
	PING, page 251	Determine if a client has an active server connection.
	SET_APIDEADLOCK, page 280	Sets a deadlock trigger on a particular API method or operation.
	SET_OPTIONS, page 287	Turn various tracing options on or off.
	SHOW_SESSIONS, page 296	Provides information about current, active sessions and a user-specified number of timed-out sessions.
Web Publishing Management	WEBCACHE_PUBLISH, page 312	Invokes the dm_webcache_publish method to publish documents to a Web site.

ADOPT_FTINDEX

Purpose

Swaps the attribute values between the specfied standby index and the current searchable index for a storage area.

Syntax

```
EXECUTE adopt_ftindex [[FOR]] standby_index_obj_id]
[WITH name='standby_index_name']
```

With EXECUTE, do not include the FOR clause if the NAME argument is included.

```
dmAPIExec("apply,session,standby_index_obj_id,ADOPT_FTINDEX
[,NAME,S,standby_index_name]")
```

With Apply, specify *standby_index_obj_id* as NULL if you include the NAME argument.

Arguments

Table 3–2. ADOPT_FTINDEX Arguments

Argument	Datatype	Value	Description
NAME	S	standby_index_name	Name of the standby index with which to swap values. Use the name of the index's fulltext index object.

Return Value

The method returns a collection with a single query result attribute that indicates the success or failure of the method.

General Notes

You must have at least Sysadmin privileges to execute this method.

A storage area may have multiple indexes associated with it. However, users are allowed to use only one for searching. The ADOPT_FTINDEX method is used to swap between the current searachable index and one of the associated standby indexes. The method swaps the attribute values in the fulltext index object of the specified standby index with the attribute values in the fulltext index object of the current searchable index. In this manner, the previous standby index becomes the current searchable index and the previous searchable index becomes a standby index. (Some attributes, such as r_object_id, is_standby, index_name, are not switched.) Swapping values in this manner allows you to swap indexes for a storage area without requiring you to reset attributes in all the content objects representing content in the storage areas.

Related Admininstration Methods

None

Example

These examples adopt the standby index identified by 3b0000019348ac2e as the searchable index:

```
EXECUTE adopt_ftindex FOR 3b0000019348ac2e
```

```
dmAPIExec("apply,S0,3b0000019348ac2e,ADOPT_FTINDEX")
```

The next two examples use the NAME argument instead of specifying the index object ID:

```
EXECUTE adopt_ftindex WITH NAME='cust_index_1'
```

```
dmAPIExec("apply,S0,NULL,ADOPT_FTINDEX,
  NAME,S,cust_index_1")
```

BATCH_PROMOTE

Purpose

Promotes multiple objects to their next state.

Syntax

```
dmAPIGet("apply,session,NULL,BATCH_PROMOTE,
ARGUMENTS,S,'object_ID{,object_ID}' ")
```

This method cannot be executed using the DQL EXECUTE statement.

Arguments

Table 3–3. BATCH_PROMOTE Arguments

Argument	Datatype	Value	Description
ARGUMENTS	S	*comma-separated list of object IDs*	Identifies the objects to promote. Use the objects' object IDs. You must enclose the list of object IDs in single quotes.

Return Value

BATCH_PROMOTE launches a method, dm_bp_batch. If the method executes successfully, BATCH_PROMOTE returns a collection with one result object. If the method is not successfully launched, BATCH_PROMOTE returns nothing. If BATCH_PROMOTE returns nothing, use Getmessage to retrieve the error message.

If BATCH_PROMOTE returns a collection, check the value of the method_return_val attribute of the result object. A value of 36 indicates that all objects were successfully promoted. Any value other than 36 means that an error occurred and processing stopped at some point. Use Getmessage to retrieve the error mesesage.

Permissions

The user issuing the method must have the necessary permissions to promote the objects listed in the argument.

General Notes

BATCH_PROMOTE allows you to promote multiple objects with one command. The objects can be attached to different lifecycles and can be in different states.

The method checks the permissions on each object specified to ensure that the user issuing the method has the correct permissions to promote the object. Then the method checks that associated lifecycle or lifecycles are valid. Finally, the method executes any entry criteria specified for the objects' next states. If BATCH_PROMOTE encounters an error at this stage, the method exits and returns nothing.

If the permissions are correct, the lifecycles are valid, and any entry criteria are fulfilled, BATCH_PROMOTE launches the dm_bp_batch method. The method performs any action procedures needed, as a single transaction. If an error occurs, the method exits and reports an error. Any objects promoted before the action procedure error remain promoted; the object whose procedure caused the error and any objects in the list after it are not promoted. For example, if the argument list includes 6 objects and an error occurs on an action procedure for object 4 in the list, objects 1, 2, and 3 are promoted. Objects 4, 5, and 6 are not.

There are two limitations on the use of BATCH_PROMOTE:

- BATCH_PROMOTE cannot run actions on behalf of the lifecycle_owner. If the value in the a_bpaction_run_as attribute in the docbase config object is set to lifecycle_owner, BATCH_PROMOTE exits with an error.
- You can specify a maximum of 200 objects in each execution of BATCH_PROMOTE.

Related Administration Methods

None

CAN_FETCH

Purpose

Determines if the server can fetch a specified content file.

Syntax

```
EXECUTE can_fetch FOR 'content_object_id'

dmAPIGet("apply,session,content_obj_id,CAN_FETCH")
```

Arguments

CAN_FETCH has no arguments.

Return Value

CAN_FETCH returns a collection with one query result object. The object has one Boolean attribute that is set to TRUE if the fetch is possible or FALSE if it is not.

Permissions

Anyone can use this method.

General Notes

If a Docbase has a distributed storage area, it is possible to configure the servers so that they can fetch from all distributed storage area components, from a subset of the components, or only from the local component. (For information about configuring this capability, refer to the *Distributed Configuration Guide*.) In such Docbases, you can use the CAN_FETCH method to determine whether a server can fetch from a particular distributed storage area component.

In a content server configuration, the CAN_FETCH method is executed by the content server.

Related Administration Methods

GET_PATH, page 205

Examples

The following examples determine whether Content Server can fetch the content file associated with the content object 06000002472185e1:

```
EXECUTE can_fetch FOR '06000002472185e1'

dmAPIGet("apply,s0,06000002472185e1,CAN_FETCH")
```

CHECK_CACHE_CONFIG

Purpose

Requests a consistency check on a particular cache config object.

Syntax

```
EXECUTE check_cache_config [FOR 'cache_config_id']
[WITH argument = value ][,argument = value]
```

With EXECUTE, do not include the FOR clause if you include the CONFIG_NAME argument.

```
dmAPIGet("apply,session,cache_config_id,CHECK_CACHE_CONFIG
[,argument,datatype,value ][,argument,datatype,value]")
```

With Apply, specify the cache config ID as NULL if you include the CONFIG_NAME argument.

Arguments

Table 3–4. CHECK_CACHE_CONFIG Arguments

Argument	Datatype	Value	Description
CONFIG _NAME	S	*name of cache config object*	Identifies the cache config object on which to perform the consistency check. Use the cache config's object name. This is an optional argument. If you do not include it, you must include the object ID of the cache config object.
FORCE _CHECK	B	T (TRUE) or F (FALSE)	TRUE directs Content Server to re-execute the queries regardless of the server_check_interval value. (Refer to the General Notes for a description of the use of the server_check_interval in the context of CHECK_CACHE_CONFIG.) This is an optional argument. The default is F (FALSE).

Return Value

The method returns a collection with one query result object. The object has the attributes listed in Table 3–5, page 162.

Table 3–5. Attributes in the CHECK_CACHE_CONFIG Result Object

Attribute	Datatype	Description
r_object_id	ID	Object ID of the cache config object
r_last_changed_date	Date/time	The date and time at which Content Server last validated the query results and found that something had changed.
r_last_checked_date	Date/time	The date and time at which Content Server last ran the queries to determine whether the results had changed.
server_check_interval	integer	How long the server waits between validations of the query results. The time is expressed in seconds.
client_check_interval	integer	The consistency check interval used by clients when issuing Fetch or Query_cmd methods that incude this cache confi object as an argument.
server_time	Date/time	Current server time
server_time_secs	integer	Current server time in seconds
cache_config_exists	Boolean	F (FALSE) if the specified cache config object is not found. T (TRUE) otherwise.

If an error occurs, the method returns an error rather than the collection.

Permissions

The cache config object must be owned by a superuser.

You must have at least Browse permission on the cache config object to issue this method. To issue the method with FORCE_CHECK set to TRUE, you must be a Superuser or have Execute Procedure permission on the cache config object.

General Notes

CHECK_CACHE_CONFIG directs Content Server to check whether the data defined by a particular cache config object is current. (Cache config objects define queries whose results are cached persistently on the client.) To determine if the data is current, the server compares the current time to the date and time in the object's r_last_checked_date attribute. If the difference is less than the interval defined in server_check_interval, indicating the interval has not expired, the data is considered current and the server does not recompute the data. If the interval has expired, the data is considered out of date. The server executes the dm_CheckCacheConfig method to recompute the data. The method executes the queries defined in cache_element_queries, computes a hash of the results, and stores the hash in the i_query_result_hash attribute. The method returns the new computation date and time in the query result object's r_last_checked_date attribute.

You can use the FORCE_CHECK argument to override the server check interval and force Content Server to execute the queries.

For information about persistent client caching and cache config objects and their use, refer to Persistent Client Caches, page 35 in *Content Server Fundamentals*. You can execute the dm_CheckCacheConfig method manually, using a DO_METHOD administration method. For an example of using DO_METHOD to call dm_CheckCacheConfig, refer to Automating Cache Config Data Validation , page 172, in the *Content Server Administrator's Guide*. However, explicit calls to the method are not normally necessary. The calls occur automatically, as a side effect of referencing cache config objects in Fetch and Query_cmd methods.

Related Administration Methods

None

Examples

The following example identifies the cache config object by its object ID and forces the recomputation of the data:

```
EXECUTE check_cacke_config FOR '08000002723ae36b'
WITH force_check = true
```

This next example identifies the cache config object by its owner and name.

```
dmAPIGet("apply,s0,NULL,CHECK_CACHE_CONFIG
 ,CONFIG_NAME,S,johndoe.report_app_config")
```

CHECK_FTINDEX

Purpose

Returns the status of a full-text index update operation.

Syntax

```
EXECUTE check_ftindex [FOR 'fulltext_index_obj_id']
[WITH argument = value [,argument = value]]
```

With EXECUTE, do not include the FOR clause if you include the NAME argument.

```
dmAPIGet("apply,session,fulltext_index_obj_id,CHECK_FTINDEX
[,argument,datatype,value [,argument,datatype,value]]")
```

With Apply, specify the fulltext index object ID as NULL if you include the NAME argument.

Arguments

Table 3–6. CHECK_FTINDEX Arguments

Argument	Datatype	Value	Description
NAME	S	*name*	Identifies the index you want to check. Use the name associated with the index's fulltext index object. This is an optional argument. If you do not use it, provide the object ID of the index's fulltext index object.
SERVER _PATH	S	*server_path*	This is an optional argument. If you specify this argument, the method copies the status file to the specified directory. *server_path* must identify a directory visible to the server.

Return Value

CHECK_FTINDEX returns a collection with one query result object that has three attributes: current_status, update_count, and server_path.

The current_status attribute contains one of the following integer values:

- 0, indicating the update has crashed
- 1, indicating the update is finished
- 2, indicating the update is in progress

The update_count value tells you which update operation is the subject of the status report and the current_status value. (Each time an index is updated, its r_update_count attribute is incremented by one.) If there is no update in progress, the update_count represents the last completed or attempted update.

If you include the SERVER_PATH argument, the server_path result attribute contains the value specified in the argument. If you don’t include SERVER_PATH, the attribute identifies the location of the update operation’s log file.

Permissions

Anyone can use this method.

General Notes

None

Related Administration Methods

CLEAR_PENDING, page 172

Examples

The following EXECUTE statements check the status of an update operation on the index named storage1_index and direct the results to the index_log file. The first example identifies the index by name and the second identifies the index by its object ID.

```
EXECUTE check_ftindex
WITH name = ’storage1_index’,
server_path = ’c:\documentum\data\index_log’
```

```
EXECUTE check_ftindex FOR ’0e0000012756241f’
WITH server_path = ’c:\documentum\data\index_log’
```

These examples check the status of an update on the storage1_index full-text index. The first identifies the index by name and the second example identifies the index by the object ID of its fulltext index object. In both examples, the log file is stored in the default location.

```
dmAPIGet("apply,c,NULL,CHECK_FTINDEX,
   NAME,S,storage1_index")
```

```
dmAPIGet("apply,c,0e0000012756241f,CHECK_FTINDEX")
```

CHECK_SECURITY

Purpose

Checks a user's or group's access permissions on one or more objects or checks a user's or group's permission level in one or more ACLs.

Syntax

```
EXECUTE check_security WITH user_name='name'|group_name='name',
level=security_level,object_list='list_of_objectids'
```

```
dmAPIGet("apply,session,NULL,CHECK_SECURITY,
USER_NAME|GROUP_NAME,S,name,LEVEL,I,security_level,
OBJECT_LIST,S,list_of_objectids
```

Arguments

Table 3–7. CHECK_SECURITY Arguments

Argument	Datatype	Value	Description
USER_NAME	S	*name*	Name of the user for whom you are checking permissions. If you include this argument, do not include GROUP_NAME.
GROUP_NAME	S	*name*	Name of a group for which you are checking permissions. If you include this argument, do not include USER_NAME.
LEVEL	I	*security_ level*	Minimum access permission level for which you are checking. Valid values are: 1, for None 2, for Browse 3, for Read 4, for Relate 5, for Version 6, for Write 7, for Delete
OBJECT_LIST	S	*list of object IDs*	The objects being checked. This can be a list of SysObject object IDs, a list of ACL object IDs, or both. Use a space to delimit the individual items.

Return Value

CHECK_SECURITY returns a collection of query result objects. The objects have one attribute, r_object_id. The attribute contains the object IDs of those objects submitted in the OBJECT_LIST argument for which the user or group has at least the permission level identified in the LEVEL argument.

Permissions

You must have Superuser privileges to use this method.

General Notes

There are two uses for CHECK_SECURITY:

- You can use it to determine whether a user or group has a particular access permission or better for a group of SysObjects
- You can use to determine whether a user or group has entries in one or more ACLs that give the user or group a particular access permission (or better).

To determine whether a user or group has at least the permission identified in the LEVEL argument for a particular document, include the document's object ID in the OBJECT_LIST argument.

To determine whether a user or group has entries in an ACL that give at least the permission level identified in the LEVEL argument or better to the user or group, include the object ID of the ACL in the OBJECT_LIST argument.

The method ignores all object IDs that do not represent SysObjects or ACLs or object IDs that are incorrectly identified (too few characters, too many characters, and so forth).

Related Administration Methods

None

Examples

This example checks to determine whether the user LibbyLoo has at least Write permission on three documents:

```
EXECUTE check_security WITH user_name='LibbyLoo',
level=5,object_list='09000001734912ac
    0900000153813f2b 0900000116572af3'
```

This example determines whether the group Engineering has accessor entries that give the group at least Read permission in the ACLs included in the object list:

```
dmAPIGet("apply,s0,NULL,CHECK_SECURITY,
GROUP_NAME,S,Engineering,LEVEL,I,3,
OBJECT_LIST,S,4500000112562e4c 450000017351213c
```

CLEAN_FTINDEX

Purpose

Merges partitions in a fulltext index.

Syntax

```
EXECUTE clean_ftindex [[FOR] 'fulltext_index_obj_id']
[WITH name='value']
```

With EXECUTE, do not include the FOR clause if you include the NAME argument.

```
dmAPIGet("apply,session,fulltext_index_obj_id,CLEAN_FTINDEX
[,NAME,S,value]")
```

With Apply, specify the fulltext index object ID as NULL if you include the NAME argument.

Argument

Argument Name	Datatype	Value	Description
NAME	S	*name*	Identifies the full-text index that you want to clean up. Use the name associated with the index's fulltext index object. This is an optional argument. If you do not use it, provide the object ID of the index.

Return Value

CLEAN_FTINDEX returns a collection with one query result object. The object has one Boolean attribute whose value indicates the success (TRUE) or failure (FALSE) of the operation.

Permissions

You must have Sysadmin or Superuser privileges to use this method.

General Notes

CLEAN_FTINDEX goes through the collections in an index and merges the partitions within the collections. Merging partitions makes it easier for the Verity Fulltext Engine to open the collections, which enhances full-text query performance.

CLEAN_FTINDEX does not execute if the r_update_inprogress attribute for the index is set to TRUE. When that occurs, the method returns immediately and a message is logged in the session log file. Consequently, before executing this method, set the FullTextMgr job to inactive. Additionally, this method can't be executed against an index if another update, clean or reset operation is already executing against the index.

CLEAN_FTINDEX generates mkvdk log files called status_optimize, which are stored in the fulltext working directory. By default, these log files contain fatal, error, warning, and status messages generated by mkvdk. If you have executed a MODIFY_TRACE to set the tracing level to verity or all, informational, verbose, and debug messages are also included.

The mkvdk files have a time stamp and old ones are removed when the LogPurge job is run.

Related Administration Methods

None

Examples

The following examples execute CLEAN_FTINDEX against the storage1_index full-text index, using the index's name for identification:

```
EXECUTE clean_ftindex
WITH name='storage1_index'
```

```
dmAPIGet("apply,c,NULL,CLEAN_FTINDEX,
   NAME,S,storage1_index")
```

CLEAN_LINKS

Purpose

On Windows, this method removes unneeded dmi_linkrecord objects and resets security on file store objects to the original state. On UNIX, this method removes unneeded dmi_linkrecord objects and the auxiliary directories and symbolic links.

Syntax

```
EXECUTE clean_links [WITH force_active=true|false]
```

```
dmAPIGet("apply,session,NULL,CLEAN_LINKS
[,FORCE_ACTIVE,B,T|F]")
```

Arguments

Argument Name	Datatype	Value	Description
FORCE _ACTIVE	B	T (TRUE) or F (FALSE)	TRUE directs the server to clean the links in all sessions. FALSE directs the server to clean links only in inactive sessions. The default is FALSE.

Return Value

CLEAN_LINKS returns a collection with one query result object. The object has one Boolean attribute whose value indicates the success (TRUE) or failure (FALSE) of the operation.

Permissions

You must have Superuser privileges to use this method.

General Notes

Linked storage areas are handled differently on UNIX and Windows platforms. Consesquently, the behavior of CLEAN_LINKS is slightly different on each platform.

On UNIX, when a linked storage area is referenced, the server creates a dmi_linkedrecord object and auxiliary directories and symbolic links. On Windows, the server creates a dmi_linkrecord object and resets the security of the linked storage object. Generally, the server removes unneeded linkrecord objects and, on UNIX, any unneeded auxiliary directories and symbolic links. On Windows, the server typically resets the linked storage object's security as needed. However, there are conditions that do not allow the server to do this work.

You can use CLEAN_LINKS to perform this work manually. We recommend running

CLEAN_LINKS regularly. (Note that CLEAN_LINKS is run automatically whenever the server is restarted.)

To determine if you need to run CLEAN_LINKS, run LIST_SESSIONS and compare the reported session IDs (in session[*x*]) to the values in the session_id attributes of the dmi_linkrecord objects. If the session_id attribute in a link record object references an inactive session (a session not reported in LIST_SESSIONS), that link record object is not needed. Depending on how many unneeded link record objects you find, you may want to run CLEAN_LINKS to remove them and perform the other, associated clean up work.

Related Administration Methods

LIST_SESSIONS, page 224

Examples

This example runs CLEAN_LINKS against only inactive sessions because the FORCE_ACTIVE argument is defaulted to FALSE:

```
EXECUTE clean_links
```

The following example removes the unneeded link record objects for all Docbase sessions, active and inactive:

```
dmAPIGet("apply,c,NULL,CLEAN_LINKS,FORCE_ACTIVE,B,T")
```

CLEAR_PENDING

Purpose

Used to recover from index updates that do not complete successfully.

Syntax

```
EXECUTE clear_pending [[FOR] 'fulltext_index_obj_id ']
[WITH name='value']
```

With EXECUTE, do not include the FOR clause if you include the NAME argument.

```
dmAPIGet("apply,session,fulltext_index_obj_id,CLEAR_PENDING
[,NAME,S,value]")
```

With Apply, specify the fulltext index object ID as NULL if you include the NAME argument.

Argument

Argument Name	Datatype	Value	Description
NAME	S	*name*	Identifies the index that contains the pending updates you want to clear. Use the name associated with the index's fulltext index object.

Return Value

CLEAR_PENDING returns a collection with one query result object. The object has one Boolean attribute whose value indicates the success (TRUE) or failure (FALSE) of the operation.

Permissions

You must have Sysadmin or Superuser privileges to use this method.

General Notes

CLEAR_PENDING is used to recover from index updates that do not complete successfully. The most common reason for an unsuccessful update is a filter failure that causes the update operation to loop. (When an update operation is started, the server puts the process ID of the update operation in the server log file. If you encounter a looping update operation, look in the log file to determine which process to stop.)

CLEAR_PENDING marks all of the content that was participating in the unsuccessful update as bad, so that future updates of that index will not attempt to index that content.

Before executing this method, set the FullTextMgr job to inactive. Additionally, this method can’t be executed against an index if another fulltext operation, such as an update or mark all, is already executing against the index.

Related Administration Methods

MARK_FOR_RETRY, page 240

Examples

This example uses EXECUTE to invoke the method for the storage1_index:

```
EXECUTE clear_pending WITH name='storage1_index'
```

The following example uses Apply to execute CLEAR_PENDING for the storage1_index full-text index:

```
dmAPIGet("apply,c,NULL,CLEAR_PENDING,
   NAME,S,storage1_index")
```

DB_STATS

Purpose

Returns a set of statistics about database operations for the current session.

Syntax

```
EXECUTE db_stats [WITH clear = true|false]
```

```
dmAPIGet("apply,session,NULL,DB_STATS
[,CLEAR,B,T|F]")
```

Arguments

Table 3–8. DB_STATS Arguments

Argument	Datatype	Value	Description
CLEAR	B	T (TRUE) or F (FALSE)	Indicates whether you want to clear the counters. The default is FALSE.

Return Value

DB_STATS returns a collection with one result object, described in Table 3–9, page 174.

Table 3–9. Query Result Object Attributes for DB_STATS Administration Method

Attribute	Datatype	Description
updates	integer	Number of SQL update operations performed
inserts	integer	Number of SQL insert operations performed
deletes	integer	Number of SQL delete operations performed
selects	integer	Number of SQL select operations performed
rpc_ops	integer	Number of RPC calls between Content Server and the RDBMS **Note:** This may not be implemented for all databases.
ddl_ops	integer	Number of data definition operations performed Data definition operations are operations such as create table or drop table.
max_cursors	integer	Maximum number of concurrently open cursors

Permissions

Anyone can use this method.

General Notes

DB_STATS returns a set of statistics about database operations for the current session. The statistics are counts of the numbers of:

- Inserts, updates, deletes, and selects executed
- Data definition statements executed
- RPC calls to the database
- Maximum number of cursors opened concurrently during the session

Related Administration Methods

None

Examples

The following example uses EXECUTE to invoke DB_STATS. It returns the statistics for the current docbase session and resets the counters to zero:

```
EXECUTE db_stats WITH clear=true
```

This example invokes DB_STATS using the Apply method. It returns the statistics for the current docbase session and resets the counters to zero:

```
dmAPIGet("apply,c,NULL,DB_STATS,CLEAR,B,T")
```

DELETE_REPLICA

Purpose

Removes a content file from a component of a distributed storage area.

Syntax

```
EXECUTE delete_replica FOR 'content_object_id'
WITH STORE='storage_name'
```

```
dmAPIGet("apply,session,content_obj_id,DELETE_REPLICA,
STORE,S,storage_name")
```

Arguments

Table 3–10. DELETE_REPLICA Arguments

Argument	Datatype	Value	Description
STORE	S	*storage_name*	Specifies the component storage area that contains the file to remove. This is a required argument. Use the storage area's name as defined in its storage object.

Return Value

DELETE_REPLICA returns a collection with one query result object. The object has one Boolean attribute that indicates the success (TRUE) or failure (FALSE) of the operation.

Permissions

You must have Superuser privileges to use this method.

General Notes

DELETE_REPLICA removes a content file from a storage area that is a component of a distributed storage area. Using DELETE_REPLICA also modifies the replica record object associated with the content. The replica record maintains the information that allows the server to fetch the content from any of the component storage areas. When you remove a file from a component storage area using DELETE_REPLICA, this record is updated also.

To use DELETE_REPLICA, you must be using one server for both data and content requests. If the configuration is set up for content servers, you must issue a Connect method that bypasses the content server to use DELETE_REPLCA in the session.

Related Administration Methods

IMPORT_REPLICA, page 216

Examples

This example removes the content file associated with the content object identified by 06000001684537b1 from the distcomp_2 storage area:

```
EXECUTE delete_replica FOR '06000001684537b1'
WITH STORE='distcomp_2'
```

The next example uses the Apply method to perform the same operation:

```
dmAPIGet("apply,c,06000001684537b1,DELETE_REPLICA,
STORE_S,distcomp_2")
```

DESTROY_CONTENT

Purpose

Removes content objects from the Docbase and their associated content files from storage areas.

Syntax

```
EXECUTE destroy_content FOR 'content_obj_id'
dmAPIGet("apply,session,content_obj_id,DESTROY_CONTENT")
```

Arguments

DESTROY_CONTENT has no arguments.

Return Value

DESTROY_CONTENT returns a collection with one query result object. The object has one Boolean attribute that indicates the success (TRUE) or failure (FALSE) of the operation.

Permissions

You must have Sysadmin or Superuser privileges to use this method.

General Notes

DESTROY_CONTENT removes a content object from the Docbase if the object has no parent_id values. It also removes the content file associated with the content object from its storage area. This administrative function is used by the dmclean utility to clean up storage areas.

To run DESTROY_CONTENT, you must be using one server for both data and content requests. If the configuration is set up for content servers, you must issue a Connect method that bypasses the content server to use DESTROY_CONTENT in the session.

Note: Do not use this method to archive content. It removes the file's content object, rather than marking it off-line.

Destroying Content in Content-Addressed Storage

If the storage area defined in the content object is a content-addressed storage system (a CA store storage area), Content Server first determines whether the storage area allows content to be deleted. If not, DESTROY_CONTENT fails with an error. If the storage system allows deletions, DESTROY_CONTENT checks the retention period specified for the content represented by the content object. If there are multiple addresses recorded for the content, Content Server checks the retention period stored with each address. The retention period for all addresses must be expired before Content Server can destroy the content. If the period is not expired for any address, DESTROY_CONTENT returns an error.

Note: Some operations, such as those that modify the metadata or object replication, result in the generation of a new content address for a content file. These additional addresses are stored in the i_contents attribute of a subcontent object.

If that period has not expired, DESTROY_CONTENT cannot remove the file from the storage area. The method fails with an error. If the retention period has expired or there is none set, the method removes the content.

Related Administration Methods

PURGE_CONTENT, page 260

Examples

This example uses the EXECUTE statement to invoke DESTROY_CONTENT:

```
EXECUTE destroy_content FOR '06000001684537b1'
```

This example uses the Apply method to invoke DESTROY_CONTENT:

```
dmAPIGet("apply,s0,06000001684537b1,DESTROY_CONTENT")
```

DO_METHOD

Purpose

Executes an external program, a Docbasic script, or a Java method.

Syntax

```
EXECUTE do_method WITH METHOD='method_name'
{,arguments=value}
```

```
dmAPIGet("apply,session,NULL,DO_METHOD,METHOD,S,method_name,
{,arguments,datatype,value")
```

Arguments

Table 3–11. DO_METHOD Arguments

Argument	Datatype	Value	Description
SAVE_RESULTS	B	T (TRUE) or F (FALSE)	Indicates if you wish to save the results into a document.
ARGUMENTS	S	*command-line arguments*	Specifies the command-line arguments for the program. If the method is to be executed on the application server, UTF-8 characters are accepted for the argument strings. If the method is to be executed on the method server or Content Server, only characters from the server os codepage are accepted for the argument strings. Refer to Specifying the Arguments, page 184, for more information.
TIME_OUT	I	*value*	Specifies the length in seconds of the default time-out period for the program that you are executing. Refer to Defining a Time Out Period , page 185 for more information.

Argument	Datatype	Value	Description
METHOD	S	*method_name*	Name of the method object representing the script, program, or Java method to execute. This argument is required.
LAUNCH _DIRECT	B	TRUE or FALSE	Indicates whether to execute the program using the Windows (UNIX) system API or the exec API. Set this to TRUE to use the system API. By default, it is FALSE, which uses the system call. Refer to Launching Directly, page 185 for more information. This argument is ignored if the method's use_method_server attribute is TRUE.
LAUNCH _ASYNC	B	TRUE or FALSE	Indicates whether to execute the program asynchronously. TRUE executes the method asynchronously. The default is FALSE. Setting this argument to TRUE is ignored if the SAVE_RESULTS argument is also TRUE. Refer to Launching Asynchronously, page 185 for more information.

Argument	Datatype	Value	Description
RUN_AS _SERVER	B	TRUE or FALSE	Indicates whether to run the method under the server's account. If the argument is not set, the server uses the value of the method's run_as_server attribute. You must have Sysadmin or Superuser privileges to set this to TRUE on the command line if the method identified in the METHOD argument has its run_as_server attribute set to FALSE. This argument must be TRUE for methods that have their use_method_server attribute set to TRUE.
TRACE _LAUNCH	B	TRUE or FALSE	Indicates whether to generate tracing information for the method. Recording Tracing Information and Program Output, page 187 describes where the information is stored.

Return Value

A DO_METHOD function returns a collection that contains one query result object. Table 3–12, page 183 lists the attributes of this object.

Table 3–12. Attributes of the Query Result Object Returned by DO_METHOD

Attribute	Datatype	Description
result	Integer or ID	If the program you launched was dmfilescan or dmclean, this attribute holds the object ID of the script run by the utility. Otherwise, the attribute contains an integer value indicating the success or failure of the program. For methods executed on an application server, the values are: 0, meaning a status of HTTP/1.1 2xx 1, meaning a status of HTTP/1.1 5xx -1, for any other status
result_doc_id	ID	Contains the object ID of the document that contains the output of the program. (Note that the output is captured even if the program times out.) This attribute is present only when the SAVE_RESULTS argument is set to TRUE.
launch_failed	Boolean	Indicates whether the program was successfully executed. T means that the program was not launched and the method_return_val attribute is meaningless. F means that the program was launched and the return value is the exit code of the process when the method terminated.
method_return _val	Integer	Contains the return value of the executing program. For methods executed on the application server, the values are: 0, for a status of HTTP/1.1 2xx 1, for any other status For all other methods, if the program times out, this value is 0. If launch_async is T, then method_return_val is always 0.
os_system_error	String	Contains an operating system error if one occurs. This attribute can be empty.
timed_out	Boolean	Indicates whether the DO_METHOD was terminated due to a timeout. T means the method was terminated while waiting for the program to complete. F means the method received a response.
time_out_length	Integer	The length of the time-out period.

Permissions

Anyone can use this method.

If the method you are executing has the run_as_server attribute set to FALSE in its method object, you must have at least Sysadmin privileges to override that setting in the DO_METHOD command line.

General Notes

Use DO_METHOD to execute a Docbasic script, a Java method, or other executable program. (Docbasic is Documentum's interpretive language.) You can direct the execution of the method to the method server, the application server, or Content Server. Which you choose depends on the language in which the program is written and how your site is configured. For details about each of the execution agents and how to direct a method to the agents, refer to Chapter 4, Methods and Jobs, in *Documentum Content Server Administrator's Guide*.

To use DO_METHOD, the invoked script or program must be defined in the Docbase by a method object. A method object contains attributes that tell the server the program's command line and arguments and provide parameters for executing the program. If the program is a Docbasic script, the script is stored as the content of the method. (For information about creating method objects, refer to Creating a Method Object, page 91 of the *Content Server Administrator's Guide*. Information about Docbasic scripts can be found in Documentum's *Docbasic User Guide and Reference Manual*.)

Specifying the Arguments

There are no restrictions on the format of the argument list for methods written in dmbasic or methods to be executed through the Content Server or the method server.

If the method is to be executed using the application server (use_method_server is T and method_type is java), you must pass both argument names and values in the ARGUMENTS *value*. The format for *value* is:

```
-argument_name argument_value
```

Separate multiple arguments with a single space. For example:

```
dmAPIGet("apply,c,NULL,DO_METHOD,METHOD,S,payroll_report,
SAVE_RESULTS,B,T,ARGUMENTS,S,-docbase accounting
-user auditor
-ticket DM_TICKET=0000000222a02024.accounting@host01")
```

or

```
EXECUTE do_method WITH METHOD='payroll_report',
ARGUMENTS='-docbase accounting
-user auditor
-ticket DM_TICKET=0000000222a02024.accounting@host01'
```

If you are directing the DO_METHOD to the application server, Content Server encodes the arguments using application/x-www-form-urlencoded format. For example, suppose you issued the following DO_METHOD:

```
dmAPIGet("apply,c,NULL,DO_METHOD,METHOD,S,paymethod,
SAVE_RESULTS,B,T,TRACE_LAUNCH,B,T,
ARGUMENTS,S,-docbase accounting -user paymaster
-ticket DM_TICKET=0000000222a02054.accounting@host01
-document "weekly record"
```

Content Server sends the arguments in the following format:

```
docbase=accounting&user=paymaster&ticket=DM_TIICKET%
3D0000000222a02054.accounting%4Dhost01&document=weekly+record
```

Defining a Time Out Period

The TIME_OUT argument defines a time out period for the program or script you are executing. Assigning a value to this argument overrides any value assigned to the timeout_default attribute in the program's method object.

The value that you specify cannot be greater than the value assigned to the method object's timeout_max attribute or less than the value assigned to the object's timeout_min attribute. If the maximum or minimum time that you specify on the DO_METHOD command line violates this rule, the server ignores your specification and uses the timeout_max or timeout_min value specified in the method object.

Launching Directly

When you execute DO_METHOD using Content Server as the execution agent, the server calls the Windows or Unix (depending on the platform on which the method is running) system API to execute it by default. If you set LAUNCH_DIRECT to TRUE, the server calls the exec API instead. This API executes the program or script directly, instead of calling a shell script, which provides the advantage of better error reporting.

To execute with LAUNCH_DIRECT set to TRUE, the method's method_verb attribute must contain a fully qualified pathname.

Launching Asynchronously

There are two ways to launch a program or script asynchronously:

- Use the ARGUMENTS argument to DO_METHOD to append an ampersand (&) to the end of the command line arguments. If there are multiple command-line arguments, the ampersand must be the last argument in the list.

  For example,

  ```
  EXECUTE do_method
  WITH method = 'validate',arguments = '&'
  ```

- Set the DO_METHOD's LAUNCH_ASYNC argument to TRUE.

  For example,

  ```
  EXECUTE do_method
  WITH method = 'validate',launch_async = TRUE
  ```

If you launch asynchronously and the method is executing on the application server, it is not possible to capture the method's output.

Saving Results

For DO_METHOD methods that are executing on the application server, Documentum provides a simple, example interface that captures the program output so the output can be saved into the Docbase if SAVE_RESULTS is TRUE. This interface is described fully in Chapter 4, Methods and Jobs, in the *Content Server Administrator's Guide*.

Running as the Server Account

By default, the program invoked by the DO_METHOD runs as the logged-in user. If you want the program to execute under the Content Server's account, you can:

- Set the run_as_server attribute in the associated method object to TRUE and set the RUN_AS_SERVER argument in the command line to TRUE.
- Set only the RUN_AS_SERVER argument in the command line to TRUE.

By default, both the run_as_server attribute and the RUN_AS_SERVER argument are FALSE. To run as the server account, either both must TRUE or you must override the attribute setting by setting the RUN_AS_SERVER argument to TRUE. Overriding the attribute in the DO_METHOD command line by setting the RUN_AS_SERVER argument to TRUE requires Sysadmin or Superuser privileges. (Overriding the attribute if it is set to TRUE by setting the RUN_AS_SERVER argument to FALSE requires no special privileges.)

If you execute DO_METHOD using either the method server or the application server as the execution agent, you must set both the method's run_as_server attribute to TRUE and the RUN_AS_SERVER argument to TRUE.

Notes:

- If LAUNCH_DIRECT is set to TRUE, either on the command line or in the method's attribute, RUN_AS_SERVER must also be set to TRUE. If it is not, the method does not execute correctly.
- Content Server uses the assume user program to run procedures and print jobs as the logged-in user. If you disable the assume user program (by setting the assume_user_location attribute in the server config object to a blank), Content Server runs all procedures and all print jobs under its account.

Ensuring Security on the Application Server

There are two security issues to consider when using an application server to execute a DO_METHOD that invokes a Java servlet or method:

- Determining the origin of the HTTP_POST request
- Login without passwords (this is only possible on Windows platforms)

Issuing a DO_METHOD to execute on the application server sends an internal HTTP_POST request to the application server. The invoked servlet ensures that the generated request comes from a machine that hosts a Content Server by checking the IP address of the sender against a list of Docbases. This list is set up when the application server is installed and configured. If the sender's IP address doesn't match the IP address of a Docbase host, the request fails.

The application server runs as the Content Server installation owner. Consequently, the servlet it invokes to execute DO_METHOD calls also runs as the installation owner. On Windows platforms, the current operating system user is allowed to log in to the Docbase without providing a password. Consequently, a servlet or an invoked Java method can log into the Docbase with superuser privileges without providing a password.

If you write a method that logs in in that manner, you may want to ensure that the actual user who issues the DO_METHOD to invoke the method has the appropriate privileges to execute the program as a superuser. To do this, send the current user's name and a login ticket to the method in the arguments. The method can use these to log in and check the privileges of the user before connecting as the current operating system user.

Recording Tracing Information and Program Output

Trace information and program output are two different sets of information. Trace information is information about the DO_METHOD invocation and its success or failure. The program output is the results returned by the program called by the DO_METHOD.

Setting TRACE_LAUNCH to TRUE logs tracing information to the Docbase log. Setting SAVE_RESULTS to TRUE saves the execution output of the invoked program.

Related Administration Methods

EXEC_SQL, page 194
HTTP_POST, page 211

Examples

This example executes the update_accounts procedure, specifying a timeout period of 120 seconds (2 minutes):

```
EXECUTE do_method
WITH method_name='update_accounts',
time_out=120
```

The following example executes the user-defined procedure run_rpt_newaccts and saves the results to a document:

```
dmAPIGet("apply,s0,NULL,DO_METHOD,
 METHOD,S,run_rpt_newaccts,SAVE_RESULTS,B,T")
```

This example executes a method on the application server:

```
dmAPIGet("apply,s0,NULL,DO_METHOD,
METHOD,S,check_vacation,SAVE_RESULTS,B,T,
ARGUMENTS,S,-docbase HumanRSRC -user adminasst
-ticket DM_TICKET=0000000221c02052.HumanRSRC@hr025
-document "monthly payroll")
```

DROP_INDEX

Purpose

Destroys a user-defined object type index.

Syntax

```
EXECUTE drop_index [[FOR] 'dmi_index_id']
[WITH name = 'index_name']
```

With EXECUTE, do not include the FOR clause if you include the NAME argument.

```
dmAPIGet("apply,session,dmi_index_id,DROP_INDEX
[,NAME,S,index_name]")
```

With Apply, specify the index object ID as NULL if you include the NAME argument.

Arguments

Table 3–13. DROP_INDEX Arguments

Argument	Datatype	Value	Description
NAME	S	*index_name*	Identifies the index by the name of its index (dmi_index) object.

Return Value

DROP_INDEX returns a collection that contains one query result object. The object has one Boolean attribute whose value indicates the success (TRUE) or failure (FALSE) of the operation.

Permissions

You must have Superuser privileges to use this method.

General Notes

You can obtain an object type index's name or object ID from the dmi_index type table. Each row in this table represents one index in the Docbase.

Related Administration Methods

FINISH_INDEX_MOVES, page 196
MAKE_INDEX, page 235
MOVE_INDEX, page 249

Examples

These examples illustrate using EXECUTE to drop a user-defined index on the dm_user object type. The first example identifies the index by its name, user_index, and the second example identifies the index by its object ID.

```
EXECUTE drop_index WITH name='user_index'
```

```
EXECUTE drop_index FOR '1f00000011231563a'
```

These examples illustrate using the Apply method to drop a user-defined index on the dm_user object type. The first example identifies the index by its name, user_index, and the second example identifies the index by its object ID.

```
dmAPIGet("apply,s0,NULL,DROP_INDEX,NAME,S,user_index")
```

```
dmAPIGet("apply,s0,1f00000011231563a,DROP_INDEX")
```

DUMP_FTINDEX

Purpose

Creates and saves a dump record object for a full-text index.

Syntax

```
EXECUTE dump_ftindex [[FOR] 'fulltext_index_obj_id']
WITH [name = 'value',] file = 'value'
```

With EXECUTE, do not include the FOR clause if you include the NAME argument.

```
dmAPIGet("apply,session,fulltext_index_obj_id,DUMP_FTINDEX
[,NAME,S,value],FILE,S,value")
```

With Apply, specify the fulltext index object ID as NULL if you include the NAME argument.

Arguments

Table 3–14. DUMP_FTINDEX Arguments

Argument	Datatype	Value	Description
NAME	S	*index_name*	Identifies the index to dump. This is optional. You can specify the object ID of the fulltext index object instead.
FILE	S	*file_name*	Specifies the name of the resulting dump file. This must be a full path for the dump file.

Return Value

DUMP_FTINDEX returns a collection with one query result object. The object has one Boolean attribute whose value indicates the success (TRUE) or failure (FALSE) of the operation.

Permissions

You must have Sysadmin or Superuser privileges to use this method.

General Notes

DUMP_FTINDEX creates and saves a dump record object for a full-text index. This creates a dump file that contains the content of the index. You can use this file to load a shadow index. Shadow indexes are used in single-Docbase distributed configurations to improve access for users at the distributed sites. For more information about shadow indexes, refer to the *Documentum Distributed Configuration Guide*.

It is recommended that you run a CLEAN_FTINDEX administration method against the fulltext index before executing DUMP_FTINDEX on the index. Cleaning the index optimizes the index and results in a smaller dump file when the DUMP_FTINDEX method is run.

Related Administration Methods

LOAD_FTINDEX , page 230

Examples

The following examples dump the storage1_index into a file named index_dump:

```
EXECUTE dump_ftindex WITH name='storage1_index',
file='c:\adminops\index_dump'
```

```
dmAPIGet("apply,c,NULL,DUMP_FTINDEX,
NAME,S,storage1_index,
FILE,S,c:\adminops\index_dump")
```

ESTIMATE_SEARCH

Purpose

Returns the number of results matching a particular SEARCH condition.

Syntax

```
EXECUTE estimate_search [[FOR] 'fulltext_index_obj_id']
WITH [name = 'index_name'] [,type = 'object_type']
[,query = 'value']
```

With EXECUTE, do not include the FOR clause if you include the NAME argument.

```
dmAPIGet("apply,session,fulltext_index_obj_id,ESTIMATE_SEARCH
[,NAME,S,index_name] [,TYPE,S,object_type]
[,QUERY,S,value] ")
```

With Apply, specify the fulltext index object ID as NULL if you include the NAME argument.

Arguments

Table 3–15. ESTIMATE_SEARCH Arguments

Argument	Datatype	Value	Description
NAME	S	*index_name*	Identifies the index to be searched. This is optional. You can specify the object ID of the fulltext index object instead.
TYPE	S	*object_type*	Identifies the type of objects to be searched. All subtypes of the specified type are included in the search also. If not included, the method searches all object types in the index.
QUERY	S	*value*	Defines the word or phrase for which you are searching. You can use a Boolean Plus expression for the value or a particular word or phrase. If not included, the method's return value represents all objects in the index.

Return Value

ESTIMATE_SEARCH returns one of the following:

- The exact number of matches that satisfy the SEARCH condition if the user running the method is a superuser or there are more than 25 matches.
- The number 25 if there are 0-25 matches and the user running the method is not a superuser.
- The number -1 if an error occurs during execution.

Errors are logged in the session log file.

Permissions

Any user can execute this method. However, the return value is affected by the user's privilege level. Refer to the General Notes for an explanation.

To execute this method, the server.ini key, use_estimate_search, must be set to TRUE. The key defaults to FALSE.

General Notes

ESTIMATE_SEARCH is a useful tool for fine-tuning a SEARCH condition in a SELECT statement. ESTIMATE_SEARCH provides an estimate of the number of results a query will return. Use it to determine how selective or unselective a particular query is. Do not use it to determine the exact number of results a particular query will return. It is intended only as a way to tune queries. Factors such as security affect the actual number of results returned when the actual query is run.

If the user executing the method is a superuser, the method returns the exact number of matches regardless of how few or how many matches are returned.

If the user is not a superuser, the method returns the exact number of matches if the number is greater than 25. If the number of matches is 0-25, the method always returns the number 25.

Related Administration Methods

None

Example

```
EXECUTE estimate_search WITH name='filestore2_indx',
type='dm_document',query='Competitor Evaluation'
```

```
dmAPIGet("apply,c,NULL,ESTIMATE_SEARCH
,NAME,S,filestore2_indx,TYPE,S,dm_document
,QUERY,S,Competitor Evaluation")
```

EXEC_SQL

Purpose

Executes SQL statements.

Syntax

```
EXECUTE exec_sql WITH query='sql_query'
```

```
dmAPIGet("apply,session,NULL,EXEC_SQL,
 QUERY,S,sql_query]")
```

Argument

Table 3–16. EXEC_SQL Arguments

Argument	Datatype	Value	Description
QUERY	S	*sql_query*	Defines the query that you want to execute.

Return Value

EXEC_SQL returns a collection that contains one query result object. The object has one Boolean attribute whose value is TRUE if the query succeeded and FALSE if it was unsuccessful.

Permissions

You must have superuser privileges to use this method.

General Notes

EXEC_SQL executes any SQL statement with the exception of SQL Select statements.

If you use the Apply method to execute the method and the query contains commas, you must enclose the entire query in single quotes. For example:

```
dmAPIGet("apply,s0,NULL,EXEC_SQL,QUERY,S,'create table
 mytable (name char(32), address char(64))'")
```

In the EXECUTE statement, character string literals must always be single-quoted:

```
EXECUTE exec_sql
with query='create table mytable (name char(32), address char(64))'
```

Related Administration Methods

DO_METHOD, page 180

Examples

Refer to the General Notes.

FINISH_INDEX_MOVES

Purpose

Completes all unfinished object type index moves.

Syntax

```
EXECUTE finish_index_moves
dmAPIGet("apply,session,NULL,FINISH_INDEX_MOVES")
```

Arguments

FINISH_INDEX_MOVES has no arguments.

Return Value

FINISH_INDEX_MOVES returns a collection with one query result object. The object has one Boolean attribute whose value indicates the success (TRUE) or failure (FALSE) of the operation.

Permissions

You must have Superuser privileges to use this method.

General Notes

FINISH_INDEX_MOVES completes an interrupted move operation for an object type index.

Moving an object type index is not an atomic operation. Consequently, if a move operation is interrupted, the Docbase may be left with a dmi_index object that has no associated index. To resolve this situation, use FINISH_INDEX_MOVES. This method scans the dmi_index table and completes all unfinished index moves.

Related Administration Methods

DROP_INDEX, page 188
MAKE_INDEX, page 235
MOVE_INDEX, page 249

Example

Refer to the syntax description.

GET_FILE_URL

Purpose

Returns the URL to a particular content file.

Syntax

```
EXECUTE get_file_url FOR object_id
WITH format='format_name' [,page=page_number,]
[page_modifier='value']
```

```
dmAPIGet("apply,session,object_id,GET_FILE_URL,
FORMAT,S,format_name[,PAGE,I,page_number]
[,PAGE_MODIFIER,S,value]")
```

object_id is the object ID of the document that contains the content file.

Arguments

Table 3–17. GET_FILE_URL Arguments

Argument	Datatype	Value	Description
FORMAT	S	*format_name*	Name of the content format, as specified in the format object.
PAGE	I	*page_number*	Page number of the content file. Setting this to -1 directs the server to return the URLS for all primary content pages and renditions that have the specified format. The default is 0.

Return Value

GET_FILE_URL returns a collection consisting of one object with five attributes. One attribute indicates the success or failure of the method call. If the call is successful, the other attribute values can be concatenated to compose the URL.

The attributes are:

result	Boolean attribute whose value indicates whether the method was successful. T (TRUE) indicates successful completion and F (FALSE) indicates failure.
base_url	The value in the base_url attribute is the base URL used by the Web server to retrieve the content. This value is retrieved from the base_url attribute of the storage area that contains the file.
store	The value in the store attribute is the name of the storage area that contains the file.
path	The value in the path attribute is a path to the file. The path is relative to the storage area identified in the store attribute.
ticket	Contains an encryption of the path value plus a time stamp.

Permissions

You must have at least Read permission on the object or Sysadmin privileges to use this method.

General Notes

Use GET_FILE_URL in an application when you want to access content using a Web server or a streaming server.

Related Administration Methods

None

Example

```
EXECUTE get_file_url FOR 090000215400ac12
WITH format='jpeg_th',page=0
```

```
dmAPIGet("apply,s0,090000215400ac12,GET_FILE_URL,
 FORMAT,S,jpeg_th,PAGE,I,0")
```

GET_FTINDEX_SIZE

Purpose

Returns the size, in bytes, of a particular full-text index.

Syntax

```
EXECUTE get_ftindex_size [[FOR] 'fulltext_index_obj_id']
[WITH name = 'index_name']
```

With EXECUTE, do not include the FOR clause if you include the NAME argument.

```
dmAPIGet("apply,session,fulltext_index_obj_id,
GET_FTINDEX_SIZE[,NAME,S,index_name]")
```

With Apply, specify the fulltext index object ID as NULL if you include the NAME argument.

Argument

Table 3–18. GET_FTINDEX_SIZE Arguments

Argument	Datatype	Value	Description
NAME	S	*index_name*	Identifies the index to check. This is optional. You can specify the object ID of the fulltext index object instead.

Return Value

GET_FTINDEX_SIZE returns a collection with one query result object. The object has one attribute that contains the size, in bytes, of the index.

Permissions

Anyone can use this method.

General Notes

This method is useful when you want to determine how much disk space an index is using.

Related Administration Methods

None

Examples

The following EXECUTE statement determines the size of the filestore2 index:

```
EXECUTE get_ftindex_size
WITH name='filestore2_index'
```

The following example returns the size of the storage1_index full-text index:

```
dmAPIGet("apply,c,NULL,GET_FTINDEX_SIZE,
 NAME,S,storage1_index")
```

GET_INBOX

Purpose

Returns items from an Inbox.

Syntax

```
EXECUTE get_inbox [WITH name='user_name'][,category='value']
[,batch=value]{,order_by='attr_name [asc|desc]'}]
```

```
dmAPIGet("apply,session,NULL,GET_INBOX
[,NAME,S,user_name][,CATEGORY,I,value][,BATCH,I,value]
{,ORDER_BY,S,attr_name [asc|desc]}")
```

Arguments

Table 3–19. GET_INBOX Arguments

Argument	Datatype	Value	Description
NAME	S	*user_name*	User whose Inbox items you want to retrieve. The default value is the session user.
CATEGORY	I	*value*	The kind of items to retrieve. Valid values are: 1, for workflow tasks 2, for router tasks 4, for notifications 8, for completed router tasks To retrieve multiple kinds of items, use the sum of the integers representing the items. For example, to retrieve workflow tasks and notifications, specify the value as 5. The default is 3, workflow tasks and router tasks.

Argument	Datatype	Value	Description
BATCH	I	*value*	Number of items returned by each Next method issued against the collection returned by GET_INBOX. The default value is 0, meaning return all rows.
ORDER_BY	S	*attr_name*	Attribute by which to order the returned items. The attribute must be an attribute of the dmi_queue_item object type. Including asc sorts the returned items in ascending order. desc sorts the items in descending order. The default is ascending order. The default ordering is by r_object_id

Return Value

GET_INBOX returns a collection containing the Inbox items in query result objects. The query result objects have 49 attributes. The first 41 are the attributes of the dmi_queue_item object representing the Inbox item. The attribute names in the query result object match the names of the attributes of the queue items. The remaining 8 attributes identify the SysObject associated with the Inbox item, if any. Generally, these attributes have values if the Inbox item is a workflow or router task. Notification items have no values in these 8 attributes. (Refer to the General Notes for a more detailed explanation.)

Table 3–20, page 202, lists the SysObject-related attributes and their corresponding SysObject attribute.

Table 3–20. SysObject-related Attribute Names for GET_INBOX Query Result Objects

Query Result Attribute Name	Associated SysObject Attribute
pkg_object_id	r_object_id
pkg_object_name	object_name
pkg_object_type	r_object_type
pkg_content_type	a_content_type
pkg_application_type	a_application_type
pkg_link_cnt	r_link_cnt

Query Result Attribute Name	Associated SysObject Attribute
pkg_lock_owner	r_lock_owner
pkg_is_virtual_doc	r_is_virtual_doc

Permissions

Any user can issue this method to return either his or her own Inbox items or the Inbox items of another user.

General Notes

With one exception, you can specify the arguments in any order. The exception is ORDER_BY. This argument must appear last.

Generally, GET_INBOX returns one query result object for each item in user's Inbox. However, if a particular task has multiple objects associated with it, the method returns one query result object for each object associated with the task. The queue item attribute values for the multiple objects will be the same. Only the values of the eight SysObject-related attributes will be different.

The eight SysObject-related attributes contain null values in the following cases:

- The Inbox item is an event notification and therefore has no associated object.
- The Inbox item is a task, but its associated object has been deleted from the Docbase.
- The Inbox item is a task but the user who issues the method doesn't have at least Browse permission on the associated object.
- The Inbox item is a workflow task, such as a Beginning task, that has no package attached to it.

Related Administration Methods

None

Examples

This example returns all tasks and notifications for the user issuing the method:

```
EXECUTE get_inbox WITH category='7'
```

This example returns just notifications for virtual user named " my_app" in batches of 20:

```
dmAPIGet("apply,c,NULL,GET_INBOX,NAME,S,my_app,
CATEGORY,I,4,BATCH,I,20")
```

GET_LAST_SQL

Purpose

Retrieves the SQL translation of the last DQL statement issued.

Syntax

```
EXECUTE get_last_sql
```

```
dmAPIGet("apply,session,NULL,GET_LAST_SQL ")
```

Arguments

None

Return Value

GET_LAST_SQL returns a collection with one query result object. The result object has one attribute whose value is the last SQL statement. To see the statement, issue a Next on the collection and then dump the collection.

If the last SQL tracing option is turned off, this method returns the following error message:

No SQL Statement is available because Last SQL Trace is disabled.

Permissions

Any one can issue this method.

General Notes

The last SQL tracing feature is turned on by default when a server starts. If the feature is turned off, you can turn it on using the last_sql_trace option of the SET_OPTIONS method. (Refer to SET_OPTIONS, page 287, for instructions.)

Related Administration Methods

None

Examples

```
EXECUTE get_last_sql
```

```
dmAPIGet("apply,S0,NULL,GET_LAST_SQL")
```

GET_PATH

Purpose

Returns the directory location of a content file stored in a distributed storage area.

Syntax

```
EXECUTE get_path [FOR] 'content_obj_id'
[WITH store = 'value']
```

```
dmAPIGet("apply,session,content_obj_id,GET_PATH
[,STORE,S,value]")
```

Arguments

Table 3–21. GET_PATH Arguments

Argument	Datatype	Value	Description
STORE	S	*storage_component _name*	Specifies a storage area that contains the file whose full path you want to determine. This is an optional argument. Use the storage area's name as defined in its storage object.

Return Value

Returns the directory path of the specified content file.

Permissions

Anyone can use this method.

General Notes

In a content server configuration, the GET_PATH function is executed by the content server.

If you do not include the STORE argument, the method looks in the local component of the distributed storage area. If the file isn't found in the local area, the method attempts to create a replica of the file in the local area and returns the path of the local replica.

Related Administration Methods

None

Examples

The following examples return the file path for the content file represented by the content object whose object ID is 060000027435127c in the storage1 storage area:

```
EXECUTE get_path FOR '060000027435127c'
WITH store='storage1'
```

```
dmAPIGet("apply,c,060000027435127c,GET_PATH,
STORE,S,storage1")
```

GET_SESSION_DD_LOCALE

Purpose

Returns the locale in use for the current session.

Syntax

```
EXECUTE get_session_dd_locale
dmAPIGet("apply,session,NULL,GET_SESSION_DD_LOCALE")
```

Arguments

None

Return Value

The method returns a collection with one result object. The result object has one attribute, named dd_locale. The value of this attribute is the locale for the session.

Permissions

Anyone can use this method.

General Notes

None

Related Methods

None

Example

```
dmAPIGet("apply,S0,NULL,GET_SESSION_DD_LOCALE")
```

GETSTYLE_FTINDEX

Purpose

Retrieves the topic set associated with a particular index or the style files used to index a particular document.

Syntax

To obtain a topic set:

```
EXECUTE getstyle_ftindex [[FOR] 'fulltext_index_obj_id']
WITH [name='index_name',] verity_style='topic_trees'
```

- With EXECUTE, do not include the FOR clause if you include the NAME argument.

  ```
  dmAPIGet("apply,session,fulltext_index_object_id,
  GETSTYLE_FTINDEX[,NAME,S,index_name],
  VERITY_STYLE,S,topic_trees")
  ```

- With Apply, specify the fulltext index object ID as NULL if you include the NAME argument.

To obtain a style file:

```
EXECUTE getstyle_ftindex WITH verity_style='style_
type',[sample_object='object_name'],
server_path='server_path'
```

```
dmAPIGet("apply,session,NULL,GETSTYLE_FTINDEX,
VERITY_STYLE,S,style_type,[SAMPLE_OBJECT,S,object_name]
SERVER_PATH,S,server_path")
```

Arguments

Table 3–22. GETSTYLE_FTINDEX Arguments

Argument	Datatype	Value	Description
NAME	S	*name*	Identifies the index whose topic set you want to retrieve. Use this argument only when you are retrieving a topic set for an index. Specify the name of the index's fulltext index object. Optionally, you can omit the NAME argument and specify the object ID of the fulltext index object.

Argument	Datatype	Value	Description
VERITY _STYLE	S	topic_trees or *style_type*	Specifies what you want to retrieve. To retrieve the topic set for an index, use topic_trees and include the NAME argument. To retrieve the style file that will be assigned to a new collection, specify *style_type* using the file extension for the file. To retrieve a style file used for particular document, specify the *style_type* using the file extension for the file and include the SAMPLE_OBJECT argument. Valid file extensions are: *lex*, for the lex file *stp*, for the stop file *syd*, for the thesaurus file *uni*, for the style.uni file *zon*, for the zone file
SAMPLE _OBJECT	S	*object_name*	Identifies the document whose style file you want to retrieve. This must be a currently indexed document. Use the document's object name. Do not use this argument if you are retrieving a topic tree set.
SERVER _PATH	S	*server_path*	Specifies where to copy the retrieved style file. Use this argument only if you are retrieving a style file. The directory you specify must be visible to the server.

Return Value

GETSTYLE_FTINDEX returns a collection with one query result object.

If you are retrieving a topic set, the query result object has one string attribute, called Result, that contains the full server path for the topic set. If there is no topic set for the index, the string is null.

If you are retrieving a style file, the query result object has one Boolean attribute that contains TRUE if the style file was successfully copied to the location specified in SERVER_PATH or FALSE if the operation failed.

Permissions

You must have Superuser privileges to use this method.

General Notes

None

Related Administration Methods

RMSTYLE_FTINDEX, page 275
SETSTYLE_FTINDEX, page 292

Examples

These examples retrieve the directory path of the topic set in the French locale for a index named storage1_index.

```
EXECUTE getstyle_ftindex
WITH name = ’storage1_index’,
verity_style = ’topic_trees’
```

```
dmAPIGet("apply,c,NULL,GETSTYLE_FTINDEX,
NAME,S,storage1_index,VERITY_STYLE,S,topic_trees")
```

The next example copies the lex style file for the document named Site Options to the location specified in the SERVER_PATH argument.

```
EXECUTE getstyle_ftindex
WITH verity_style = ’lex’,
sample_object = ’Site Options’,
server_path = ’u01/adminops/lex_file/mycustom.lex
```

The next example retrieves the lex file used when indexing the document named mydocument (and all other documents contained in the same collection as the specified document):

```
dmAPIGet("apply,c,NULL,GETSTYLE_FTINDEX,
VERITY_STYLE,S,lex,SAMPLE_OBJECT,S,mydocument,
SERVER_PATH,S,c:\adminops\lex_file")
```

HTTP_POST

Purpose

Sends an HTTP_POST request invoking a Java servlet.

Syntax

```
EXECUTE http_post WITH app_server_name='name'
[,arguments=argument_list][,save_response=value]
[,time_out=value][,launch_asynch=value][,trace_launch=value]
```

```
dmAPIGet("apply,session,NULL,HTTP_POST,
APP_SERVER_NAME,S,name [,ARGUMENTS,S,argument_list]
[,SAVE_RESPONSE,I,value][,TIME_OUT,I,value]
[,LAUNCH_ASYNCH,B,value][,TRACE_LAUNCH,B,value]")
```

Arguments

Table 3–23. HTTP_POST Arguments

Argument	Datatype	Value	Description
APP_SERVER _NAME	S	*name*	Name of the Java servlet. This must match a servlet identified in the app_server_name attribute of the server config object.
ARGUMENTS	S	*argument list*	Defines the command line arguments passed to the servlet or Java method. UTF-8 characters are acceptable for the argument strings.
SAVE _REPONSE	I	*integer*	Indicates whether to save the response in a document in the Docbase. Value values are: 0, do not save 1, save the results -1, save the results if there is an error The default is 0, do not save.
TIME_OUT	I	*integer*	The length of time, in seconds, to wait for a response to the HTTP_POST request. The default is 60 seconds.

Argument	Datatype	Value	Description
LAUNCH ASYNCH	B	T (TRUE) or F (FALSE)	Indicates whether to execute the HTTP_POST request asynchronously. TRUE executes the HTTP_POST asynchronously. The default is FALSE. This argument is ignored if SAVE_RESPONSE is TRUE.
TRACE _LAUNCH	B	T (TRUE) or F (FALSE)	Indicates whether to generate tracing information for the HTTP_POST request. If TRACE_LAUNCH is TRUE, the information is stored in the server log. The default is FALSE.

Return Value

HTTP_POST returns a collection with one query result object that has seven attributes. Table 3–24, page 212, lists the attributes:

Table 3–24. Query Result Object Attributes for HTTP_POST

Attribute Name	Description
result	Indicates the status of the HTTP_POST. Possible values are: 0, indicating the status HTTP/1.1 2xx 1, indicating the status HTTP/1.1 5xx -1, indicating any other status
http_response_status	The response returned for the HTTP_POST. Some example values are: HTTP/1.1 2xx OK HTTP/1.1 5xx Internal Server Er- ror For a complete list of responses, refer to the HTTP protocol specification.
request_failed	Indicates whether the method sent an HTTP request to the application server. T (TRUE) means the method failed to send a request. F (FALSE) means the request was successfully sent to the application server.
response_doc_id	Object ID of the document in which the response is stored. This only has a value if SAVE_RESPONSE was set to TRUE.

Attribute Name	Description
time_taken	Length of time, in seconds, between when the request was sent and when a response was received. This represents the execution time of the Java method plus the time used for communication between Content Server and the application server.
timed_out	Indicates whether the method timed out. T (TRUE) means that the HTTP_POST method timed out while waiting for a response. F (FALSE) means that a response was received.
time_out_length	Time, in seconds, of the time out period.

Permissions

Anyone can use this method. Refer to Preserving Security, page 213, for information about security when using this method.

General Notes

Use HTTP_POST to invoke a servlet or Java method installed in an application server.

If you set LAUNCH_ASYNC to TRUE, Content Server closes the connection to the application server immediately after sending the request. If LAUNCH_ASYNC is FALSE, the server waits for a response until a response is received or the time out period is reached.

Setting TRACE_LAUNCH to TRUE logs tracing information about the invocation of the HTTP_POST method and its success or failure.

The ARGUMENTS argument can contain only UTF-8 characters. Content Server encodes the arguments using application/x-www-form-urlencoded format. For example, suppose you issued the following HTTP_POST method:

```
dmAPIGet("apply,c,NULL,HTTP_POST,APP_SERVER_NAME,S,payroll,
SAVE_RESPONSE,B,T,TRACE_LAUNCH,B,T,
ARGUMENTS,S,-docbase accounting -user paymaster
-ticket DM_TICKET=0000000222a02054.accounting@host01
-document "weekly record"
```

Content Server sends the arguments in the following format:

```
docbase=accounting&user=paymaster&ticket=DM_TIICKET%
3D0000000222a02054.accounting%4Dhost01&document=weekly+record
```

Preserving Security

The Tomcat application server, all servlets that it invokes, and the methods invoked by the servlets run as the Content Server installation owner, which is an account with Superuser privileges. Consequently, it is important that they are written and execute using adequate security precautions.

There are two primary security issues:

- Determining origin of HTTP_POST requests

- Login without passwords (this is only possible on Windows platforms)

The application server or the servlet has no inherent way to know whether the HTTP_POST request was sent from a Documentum Server. When you write a servlet, you must include security checking to ensure that the request comes from a Documentum Server. One recommended way is have the servlet ensure that the generated request comes from a machine that hosts a Content Server by checking the IP address of the sender against a list of valid IP addresses. (This is how the do_method servlet checks the origin. Refer to *Installing Content Server* for information about how that is set up.)

On Windows platforms, the current operating system user is allowed to log in to the Docbase without providing a password. Consequently, a servlet or an invoked Java method can log into the Docbase with Superuser privileges without providing a password.

If you write a servlet or method that logs in in that manner, you may want to ensure that the actual user who issues the HTTP_POST to invoke the program has the appropriate privileges to execute the program as a superuser. To do this, send the current user's name and a login ticket to the method in the arguments. The method can use these to log in and check the privileges of the user before connecting as the current operating system user.

Sample Servlet and Method

Documentum provides a sample servlet for handling HTTP_POST method calls and a sample method. The servlet is named DmSampleServlet.java and the method is named DmSampleMethod.java. The sample servlet and method are located in the classes folder under the WEB-INF folder (%DM_HOME%\tomcat\webapps\DmMethods\WEB-INF\classes or $DM_HOME/tomcat/webapps/DmMethods/WEB-INF/classes) The WEB-INF folder was set up when you installed the Apache Tomcat application server.

Recording Output

If you set SAVE_RESPONSE to TRUE, then anything that the invoked servlet writes to HttpServerletResponse.OutputStream is saved to a document in the Docbase.

Related Administration Methods

DO_METHOD, page 180

Examples

The following examples send an HTTP_POST request to the Java servlet called DevAppSrv. The content of the request passes a Docbase name, user name, a login ticket, and a document name to the Java servlet.

```
EXECUTE http_post WITH app_server_name='DevAppSrv',
save_response=1,trace_launch=T,time_out=60,
arguments='-docbase DevTest -user test_user
-ticket DM_TICKET=0000000221c02052.DevTest@dev012
-document "A Test"'
```

```
dmAPIGet("apply,c,NULL,APP_SERVER_NAME,S,DevAppSrv,
SAVE_RESPONSE,I,1,TRACE_LAUNCH,B,T,TIME_OUT,I,60,
ARGUMENTS,S,-docbase DevTest -user test_user
-ticket DM_TICKET=0000000221c02052.DevTest@dev012
-document 'A Test'")
```

IMPORT_REPLICA

Purpose

Imports files from one distributed storage area into another distributed storage area.

Syntax

```
EXECUTE import_replica FOR 'content_object_id'
WITH store='storage_name',file='file_name'
[,other_file='other_file_name']
```

```
dmAPIGet("apply,session,content_object_id,IMPORT_REPLICA,
STORE,S,storage_name,FILE,S,file_name
[,OTHER_FILE,S,other_file_name]")
```

Arguments

Table 3–25. IMPORT_REPLICA Arguments

Argument	Datatype	Value	Description
STORE	S	*storage_name*	Identifies the storage area in which to place the imported content file. This must be a component of a distributed storage area. Use the storage area's name.
FILE	S	*file_name*	Identifies the file to replicate.
OTHER_FILE	S	*other_file_name*	The OTHER_FILE argument is optional. It directs the server to copy the specified Macintosh resource fork file in addition to the data file. Specify the name of the resource fork file. Use this only when the file you are importing was created on a Macintosh.

Return Value

IMPORT_REPLICA returns a collection with one query result object. The object has one attribute whose value indicates success (TRUE) or failure (FALSE).

Permissions

You must have Sysadmin or Superuser privileges to use this method.

General Notes

IMPORT_REPLICA imports files from one distributed storage area into another distributed storage area. The files are considered replicas in the target storage area.

To use IMPORT_REPLICA, you must be using one server for both data and content requests. If the configuration is set up for content servers, you must issue a Connect method that bypasses the content server to use IMPORT_REPLCA in the session.

Related Administration Methods

CAN_FETCH, page 160
DELETE_REPLICA , page 176
GET_PATH, page 205
REPLICATE, page 269

Examples

The following examples import the file mydoc into the storage area named distcomp_2:

```
EXECUTE import_replica FOR '06000001402371e1'
WITH store='distcomp_2',FILE='mydoc'
```

```
dmAPIGet("apply,c,06000001402371e1,
IMPORT_REPLICA,STORE,S,distcomp_2,FILE,S,mydoc")
```

LIST_AUTH_PLUGINS

Purpose

Lists the authentication plugins that the server has loaded.

Syntax

```
EXECUTE list_auth_plugins
```

```
dmAPIGet("apply,session,NULL,LIST_AUTH_PLUGINS")
```

Arguments

None

Return Value

The method returns a collection with one query result object. The object has two repeating string attributes, plugin_id and plugin_filepath. The values in plugin_id are the unique plugin identifiers and the values in plugin_filepath are the full paths of the plugins. The values at corresponding index positions represent one plugin.

Permissions

You must have Sysadmin or Superuser privileges to execute this method.

General Notes

This method is useful only at sites that have a Trusted Content Services license because the ability to use authentication plugins is a feature of Trusted Content Services.

Related Administration Methods

None

Example

The following IAPI excerpt shows how this method is executed and the results obtained.

```
API>apply,c,NULL,LIST_AUTH_PLUGINS
...
q0
API>next,c,Q0
...
OK
API>dump,c,q0
```

```
...
USER_ATTRIBUTES
plugin_id[0]: customauth
[1]: custauth2
[2]: latin1auth
plugin_filepath[0]: C:\Documentum\product\5.2\bin\
  auth\customauth.dll
[1]: C:\Documentum\product\5.2\bin\auth\custauth2.dll
[2]: C:\Documentum\product\5.2\bin\auth\latin1auth2.dll

SYSTEM ATTRIBUTES
APPLICATION ATTRIBUTES

INTERNAL ATTRIBUTES
API>close,c,q0
```

LIST_RESOURCES

Purpose

Lists a variety of information about the server's operating system environment and the server.

Syntax

```
EXECUTE list_resources [WITH reset=true|false]
```

```
dmAPIGet("apply,session,NULL,LIST_RESOURCES
[RESET,B,T|F]")
```

Arguments

Table 3–26. LIST_RESOURCES Arguments

Argument	Datatype	Value	Description
RESET	B	T (TRUE) or F (FALSE)	TRUE reinitializes the file handle and heap size counters. FALSE keeps the present values of the counters.

Return Value

LIST_RESOURCES returns a collection with one query result object. On Windows platforms, the query result object has twelve attributes, described in Table 3–27, page 220.

Table 3–27. Collection Attributes for LIST_RESOURCES

Attribute	Datatype	Description
file_handles_in_use	integer	The number of file handles in use by the main thread and all session threads (Windows) or the current child process (UNIX).
file_handles_max	integer	The configured limit at the operating-system level on the number of file handles the process can open.

Attribute	Datatype	Description
file_handles_new	integer	A counter that indicates how many file handles have been created or destroyed since the last LIST_RESOURCES with RESET = T. If the number is negative, it means that there are fewer handles open than there were at the last LIST_RESOURCES call. (Issuing LIST_RESOURCES with RESET=T reinitializes file_handles_new to zero.)
session_heap_size_max	integer	Maximum size, in bytes, of a thread's session heap. When a session is started, its maximum heap size corresponds to this value. A value of -1 indicates that the size of the session heap will be unconstrained (the heap will grow to whatever size the server machine resources will allow).
current_heap_size_max	integer	Maximum size of the thread's session heap. This reflects the value that was in session_heap_size_max when the session was started, and is the size of the heap available to the session.
session_heap_size _in_use	integer	How much, in bytes, of the currently allocated heap (virtual memory) is in use by the session.
session_heap_size_new	integer	A count of the bytes that the heap has grown or shrunk since the last LIST_RESOURCES call. (Issuing LIST_RESOURCES with RESET=T reinitializes session_heap_size_new to zero.)
root_heap_size_in_use	integer	How much, in bytes, of the main server thread's heap is in use.
root_heap_size_new	integer	A count of the bytes that the heap has grown or shrunk since the last LIST_RESOURCES call. Issuing LIST_RESOURCES with RESET=T reinitializes session_heap_size_new to zero.
max_processes	integer	The maximum number of processes that can be created by the account under which the server is running.
server_init_file	string(255)	The full path to the server's server.ini file.
initial_working _directory	string(255)	The full path to the directory containing the server executable.

On UNIX, the result object has eight attributes, described in Table 3–28, page 222.

Table 3–28. Collection Attributes for LIST_RESOURCES

Attribute	Datatype	Description
file_handles_in_use	integer	The number of file handles in use by the main thread and all session threads (Windows) or the current child process (UNIX).
file_handles_max	integer	The configured limit at the operating-system level on the number of file handles the process can open.
file_handles_new	integer	A counter that indicates how many file handles have been created or destroyed since the last LIST_RESOURCES with RESET = T. If the number is negative, it means that there are fewer handles open than there were at the last LIST_RESOURCES call. (Issuing LIST_RESOURCES with RESET=T reinitializes file_handles_new to zero.)
heap_size_in_use	integer	The size, in bytes, of the session heap.
heap_size_new	integer	A count of the bytes that the heap has grown or shrunk since the last LIST_RESOURCES call. Issuing LIST_RESOURCES with RESET=T reinitializes heap_size_new to zero.
max_processes	integer	The maximum number of processes that can be created by the account under which the server is running.
server_init_file	string(255)	The full path to the server's server.ini file.
initial_working _directory	string(255)	The full path to the directory containing the server executable.

Permissions

Anyone can use this method.

General Notes

LIST_RESOURCES lists a variety of information about the server operating system environment and the server (the information is described in Return Value, page 220).

Related Administration Methods

DB_STATS, page 174
LIST_SESSIONS, page 224
LIST_TARGETS, page 228
SHOW_SESSIONS, page 296

Examples

The following examples return the resources information and reset the heap size counters:

```
EXECUTE list_resources WITH reset=true
```

```
dmAPIGet("apply,c,NULL,LIST_RESOURCES,RESET,B,T")
```

LIST_SESSIONS

Purpose

Lists information about all the currently active sessions and a user-specified number of historical sessions.

Syntax

```
EXECUTE list_sessions[WITH brief_info=true|false]
```

```
dmAPIGet("apply,session,NULL,LIST_SESSIONS
[,BRIEF_INFO,B,T|F]")
```

Arguments

Table 3–29. LIST_SESSIONS Arguments

Argument	Datatype	Value	Description
BRIEF_INFO	B	T (TRUE) or F (FALSE)	Indicates whether you want a subset of the attributes in the result object. The default is FALSE.

Return Value

LIST_SESSIONS returns a collection. Each result object in the collection describes one currently active session or one historical session.

If BRIEF_INFO is FALSE

Table 3–30, page 224 lists the information returned if BRIEF_INFO is FALSE.

Table 3–30. Complete Information Returned by LIST_SESSIONS (BRIEF_INFO=F)

Attribute	Description
root_pid	On Windows, the process ID of the Content Server process. On UNIX, the process ID of the root Content Server process.
shared_mem_id	The ID of the shared memory segment used by the servers. This attribute is returned only on UNIX platforms.

Attribute	Description
semaphore_id	The ID of the semaphore used by the servers. This attribute is returned only on UNIX platforms.
root_start	The time when the main server thread (root server process) was started.
session	The object ID of the session begun by user_name.
db_session_id	The ID of the database session.
pid	The ID of the session thread (process).
user_name	The user name of the user who started the session.
user_authentication	The user authentication state for the session. Possible values are password, ticket, trusted client, change password, or in progress. If the value is change password, the user logged in for that session can only perform the change password operation. No other operations are allowed.
client_host	The host name of the machine on which the session was started.
client_lib_ver	The version number of the dmcl in use for the session.
client_locale	A verbose description of the client's environment, including the operating system version, the character set in use, the language in use, and the date format.
start	The starting time of the session.
last_used	The last time the client session contacted the server.
session_status	The session status. Active means that the session is connected and has not timed out. Inactive means that the session has timed out.
shutdown_flag	Records the setting of the immediacy_level argument in the Kill method.

If BRIEF_INFO is TRUE

Table 3–31, page 226, lists the information returned if BRIEF_INFO is TRUE.

Table 3–31. Brief Information Returned by LIST_SESSIONS (BRIEF_INFO=T)

Attribute	Description
session	The object ID of the session begun by user_name.
db_session_id	The ID of the database session.
user_name	The user name of the user who started the session.
client_host	The host name of the machine on which the session was started.
start	The starting time of the session.
last_used	The last time the client session contacted the server.
session_status	The session status. Active means that the session is connected and has not timed out. Inactive means that the session has timed out.

Permissions

Anyone can use this method.

General Notes

LIST_SESSIONS returns a collection of query result objects whose attributes contain information about all the currently active sessions and a user-specified number of historical sessions. The historical sessions are past sessions that are not currently active.

The maximum number of historical sessions returned by LIST_SESSIONS is determined by the parameter history_sessions in the server's startup file (the server.ini file). There is also a startup parameter, called history_cutoff, to define a cutoff time for the history sessions. For example, if history_cutoff is set to 15 minutes, then LIST_SESSIONS will not return any historical sessions older than 15 minutes.

For a description of the server.ini file and the parameters you can set in it, refer to The Server.ini File, page 101, in the *Content Server Administrator's Guide*.

If you have a large number of active sessions, use SHOW_SESSIONS, page 296, instead of LIST_SESSIONS.

Related Administration Methods

SHOW_SESSIONS, page 296

Examples

This example executes LIST_SESSIONS with the BRIEF_INFO argument defaulted to FALSE:

```
EXECUTE list_sessions
```

This example executes LIST_SESSIONS with the BRIEF_INFO argument set to TRUE:

```
dmAPIGet("apply,c,NULL,LIST_SESSIONS,BRIEF_INFO,B,T")
```

LIST_TARGETS

Purpose

Lists the DocBrokers to which the server is currently projecting.

Syntax

```
EXECUTE list_targets
dmAPIGet("apply,session,NULL,LIST_TARGETS")
```

Arguments

LIST_TARGETS has no arguments.

Return Value

LIST_TARGETS returns a collection with one query result object. The Table 2-9. The values across the attributes at one index position represent the information about one DocBroker to which the server is projecting.

Table 3–32. Query Result Object Attributes for LIST_TARGETS

Attribute	Datatype	Description
projection_targets	string(80)	A repeating attribute that contains the names of the hosts on which the target DocBrokers are running.
projection_proxval	integer	A repeating attribute that contains the proximity value projected to the DocBroker identified in the corresponding index position in projection_targets.
projection_notes	string(80)	A repeating attribute that contains any user-defined note for the DocBroker defined at the corresponding index position in projection_targets and projection_proxval.
docbroker_status	string(64)	A repeating attribute that records whether the DocBroker identified in projection_targets at the corresponding index position is available. Valid values are: Available and Unavailable.
comments	string(64)	A repeating attribute that records whether the projection target entry was taken from the server config object or the server.ini file.

Permissions

Anyone can use this method.

General Notes

A server's DocBroker targets are those DocBrokers to which the server is sending checkpoint messages. Target DocBrokers are defined in the server's server config object and may also be defined in the server.ini file. This method returns a list of the targets defined in the server config object.

Related Administration Methods

None

Examples

The following examples list the current DocBroker targets for the server:

```
EXECUTE list_targets
```

```
dmAPIGet("apply,c,NULL,LIST_TARGETS")
```

LOAD_FTINDEX

Purpose

Loads a file created by the DUMP_FTINDEX function.

Syntax

```
EXECUTE load_ftindex[[FOR] 'fulltext_index_obj_id']
WITH [name='value',]file='file_name'
```

With EXECUTE, do not include the FOR clause if you include the NAME argument.

```
dmAPIGet("apply,session,fulltext_index_obj_id,LOAD_FTINDEX,
[NAME,S,value,]FILE,S,file_name")
```

With Apply, specify the fulltext index object ID as NULL if you include the NAME argument.

Arguments

Table 3–33. LOAD_FTINDEX Arguments

Argument	Datatype	Value	Description
NAME	S	*index_name*	Identifies the index where you want to put the shadow index. Use the name of the fulltext index object associated with the index. This argument is optional. If you do not include it, you must identify the fulltext index by its fulltext index object ID.
FILE	S	*file_name*	Identifies the dump file you want to load. It must be a dump file that was created with DUMP_FTINDEX. *file_name* must be a full path for the dump file.

Return Value

LOAD_FTINDEX returns a collection with one query result object. The object has one attribute whose value indicates the success (TRUE) or failure (FALSE)of the operation.

Permissions

You must have Sysadmin or Superuser privileges to use this method.

General Notes

LOAD_FTINDEX creates and saves a load record object for a dump file that was created by the DUMP_FTINDEX function. (You cannot use this method to create a load record object for any other type of dump file.) The process of saving the load record object loads the dump file into the shadow index.

Shadow indexes are a feature of Content Server that support single-Docbase distributed configurations. Shadow indexes can be searched but not updated. For more information about shadow indexes, refer to Adding Shadow Indexes, page 53 in the *Documentum Distributed Configuration Guide*.

Related Administration Methods

DUMP_FTINDEX, page 190

Examples

These examples use EXECUTE to load the index_dump file into the shadow index named shadow1_index:

```
EXECUTE load_ftindex WITH NAME='shadow1_index',
FILE='c:\adminops\index_dump'
```

or, on UNIX platforms:

```
EXECUTE load_ftindex WITH NAME='shadow1_index',
FILE='u01/adminops/index_dump'
```

The following examples perform the same operation using an Apply method.

```
dmAPIGet("apply,c,NULL,LOAD_FTINDEX,
NAME,S,shadow1_index,
FILE,S,c:\adminops\index_dump")
```

or, on UNIX platforms:

```
dmAPIGet("apply,c,NULL,LOAD_FTINDEX,
NAME,S,shadow1_index,
FILE,S,u01/adminops/index_dump")
```

LOG_OFF

Purpose

Turns off the RPC logging.

Syntax

```
EXECUTE log_off
dmAPIGet("apply,session,NULL,LOG_OFF")
```

Arguments

LOG_OFF has no arguments.

Return Value

LOG_OFF returns a collection with one query result object. The object has one Boolean attribute whose value indicates the success (TRUE) or failure (FALSE) of the operation.

Permissions

You must have Sysadmin or Superuser privileges to use this method.

General Notes

None

Related Administration Methods

LOG_ON, page 233

Example

The following example turns off RPC logging:

```
dmAPIGet("apply,c,NULL,LOG_OFF")
```

LOG_ON

Purpose

Turns on logging for RPC calls.

Syntax

```
EXECUTE log_on [WITH detail=true|false]
```

```
dmAPIGet("apply,session,NULL,LOG_ON
[,DETAIL,B,T|F]")
```

Arguments

Table 3–34. LOG_OFF Arguments

Argument	Datatype	Value	Description
DETAIL	B	T (TRUE) or F (FALSE)	Indicates whether you want information about the arguments passed with the RPC call and the results. The default is FALSE.

Return Value

LOG_ON returns a collection with one query result object. The object has one Boolean attribute whose value indicates the success (TRUE) or failure (FALSE) of the operation.

Permissions

You must have Sysadmin or Superuser privileges to use this method.

General Notes

The LOG_ON function turns on logging for RPC calls.

If you execute the LOG_ON function without including the DETAIL argument, the server logs information about the start and stop names and time information for the operation. If you set DETAIL to TRUE, the server also includes information about the arguments passed with each call and the results of the call.

Related Administration Methods

LOG_OFF, page 232

Examples

These examples turn on detailed RPC logging:

```
EXECUTE log_on WITH detail=true
```

```
dmAPIGet("apply,c,NULL,LOG_ON,DETAIL,B,T")
```

MAKE_INDEX

Purpose

Creates an index for a persistent object type.

Syntax

```
EXECUTE make_index
WITH type_name='object_type',attribute='attribute_name',
{attribute='attribute_name',}
[unique=true|false,]index_space='name'
```

```
dmAPIGet("apply,session,NULL,MAKE_INDEX
TYPE_NAME,S,object_type,ATTRIBUTE,S,attribute_name,
{ATTRIBUTE,S,attribute_name,}
[UNIQUE,B,T|F,]INDEX_SPACE,S,name")
```

Arguments

Table 3–35. MAKE_INDEX Arguments

Argument	Datatype	Value	Description
TYPE_NAME	S	*object_type*	Identifies the object type for which to create an index. This is a required argument.
ATTRIBUTE	S	*attribute_name*	Identifies the attribute or attributes on which to build the index. You can specify multiple attributes, but you cannot mix single-valued and repeating attributes in the same statement.

Return Value

MAKE_INDEX returns a collection with one query result object. The object has one attribute, named result, that contains the object ID of the dmi_index_object for the new index if successful.

If the method failed because the syntax was incorrect or the attribute specified did not exist, the result attribute contains F (FALSE).

If the specified index already exists, the result attribute contains '0000000000000000'.

Permissions

You must have Superuser privileges to use this method.

General Notes

MAKE_INDEX creates an index for a persistent object type. You can create an index for any persistent object type.

You can specify one or more attributes. However, if you specify multiple attributes, you must specify all single-valued attributes or all repeating attributes. You cannot mix single and repeating valued attributes in the same argument list. (This is because the underlying RDBMS stores an object's single and repeating attributes in separate tables.)

If you specify multiple attributes, the sort order within the index corresponds to the order in which the attributes are specified in the statement.

The attributes you specify must be defined for the type you specify. They cannot be inherited attributes. For example, if you want to create an index on the subject and title attributes, you would specify dm_sysobject in TYPE_NAME, as these attributes are defined for dm_sysobject. (Chapter 2, Object Reference, in the *Content Server Object Reference Manual* lists the attributes defined for each object type.)

Related Administration Methods

DROP_INDEX, page 188
FINISH_INDEX_MOVES, page 196
MOVE_INDEX, page 249

Examples

This example creates an index for the dm_sysobject object type, indexing on the subject and title attributes:

```
EXECUTE make_index WITH type_name='dm_sysobject',
attribute='subject', attribute='title'
```

The next example creates an index for the dm_sysobject type on the attribute r_creator_name:

```
dmAPIGet("apply,c,NULL,MAKE_INDEX,
TYPE_NAME,S,dm_sysobject,
ATTRIBUTE_NAME,S,r_creator_name")
```

The next example creates an index for the dm_sysobject type on the attributes r_creator_name and r_creation_date:

```
dmAPIGet("apply,c,NULL,MAKE_INDEX,
TYPE_NAME,S,dm_sysobject,
ATTRIBUTE_NAME,S,r_creator_name,
ATTRIBUTE_NAME,S,r_creation_date")
```

MARK_ALL

Purpose

Finds all content objects representing content files whose associated objects have the a_full_text attribute set to TRUE and marks the contents for indexing.

Syntax

```
EXECUTE mark_all [WITH trace=value]
dmAPIGet("apply,session,NULL,MARK_ALL[,TRACE,B,value]")
```

Arguments

Table 3–36. MARK_ALL Arguments

Argument	Datatype	Value	Description
TRACE	B	T (TRUE) or F (FALSE)	Turns tracing on or off for the method. TRUE turns on tracing. FALSE turns it off. The setting is FALSE by default.

Return Value

MARK_ALL returns a collection with one query result object. The object has one attribute whose value is the number of contents successfully marked or the value -1. The value -1 indicates that an error occurred and nothing was marked.

Permissions

You must have Sysadmin or Superuser privileges to use this method.

General Notes

MARK_ALL searches a Docbase for all content objects representing unindexed content files whose associated objects have the attribute a_full_text set to TRUE. When the method finds a content object that meets this condition, it marks the content for indexing.

Typically, setting the a_full_text attribute to TRUE causes the server to automatically mark an object's content for indexing. MARK_ALL is a backup for this functionality.

MARK_ALL is also very useful if you want to create an index for a storage area that was not previously indexed if the documents in the index have their a_full_text attributes set to TRUE.

Before executing this method, set the FullTextMgr job to inactive. This method can't be executed against an index if another full-text operation, such as an update or clean, is already executing against the index.

Tracing the Method

To trace the operations of the MARK_ALL method, include the TRACE argument in the MARK_ALL method. Trace messages are logged to %\DOCUMENTUM%\dba\log\markall_<*docbase_name*>.log ($DOCUMENTUM/dba/log/markall_<*docbase_name*>.log). The file is overwritten each time the method is executed.

Note: Turning on tracing using MODIFY_TRACE does not enable tracing for the MARK_ALL method. You must use the TRACE argument in the MARK_ALL method to trace the method.

Related Administration Methods

MARK_FOR_RETRY, page 240

Examples

These examples mark for indexing all content files in the Docbase that are unindexed and whose associated objects have a_full_text set to TRUE:

```
EXECUTE mark_all WITH trace=T
```

```
dmAPIGet("apply,c,NULL,MARK_ALL,TRACE,B,T")
```

MARK_AS_ARCHIVED

Purpose

Sets the i_is_archived attribute of an audit trail entry to TRUE.

Syntax

```
EXECUTE mark_as_archived FOR audit_obj_id
```

```
dmAPIExec("apply,session,audit_obj_id,MARK_AS_ARCHIVED")
```

audit_obj_id is the object ID of the audit trail entry. This is a dm_audittrail, dm_audittrail_acl, or dm_audittrail_group object.

Arguments

This method has no arguments.

Return Value

The method returns TRUE if successful or FALSE if unsuccessful.

Permissions

You must have Superuser privileges to execute this method.

General Notes

Use this method to set an audit trail entry's i_is_archived attribute to TRUE after you archive the entry.

Related Administration Methods

PURGE_AUDIT, page 252

Examples

```
EXECUTE mark_as_archived FOR 5f000012ae6217ce.
```

```
dmAPIExec("apply,S0,5f000012ae6217ce,MARK_AS_ARCHIVED")
```

MARK_FOR_RETRY

Purpose

Finds any content that has a particular negative update_count attribute value and marks such content as awaiting indexing.

Syntax

```
EXECUTE mark_for_retry
WITH NAME = 'index_name',UPDATE_COUNT = integer
```

```
dmAPIGet("apply,session,NULL,MARK_FOR_RETRY,
NAME,S,index_name,UPDATE_COUNT,I,integer")
```

Arguments

Table 3–37. MARK_FOR_RETRY Arguments

Argument	Datatype	Value	Description
NAME	S	*index_name*	Identifies the index that contains the objects to mark for a retry. Use the name associated with the index's fulltext index object.
UPDATE_COUNT	I	*integer*	Specifies which contents to check. Use the update_count value that represents the update operation that was not completely successful. Although the value is stored as a negative number, specify it as a positive integer

Return Value

MARK_FOR_RETRY returns a collection with one query result object. The object has one Boolean attribute whose value indicates the success (TRUE) or failure (FALSE) of the operation.

Permissions

You must have Sysadmin or Superuser privileges to use this method.

General Notes

Use MARK_FOR_RETRY in the recovery procedure after an UPDATE_FTINDEX is not completely successful. The method scans the Docbase to find any content objects that have the specified negative update_count attribute value and marks such content as awaiting indexing.

An UPDATE_FTINDEX operation may fail to index some content due to filter failure. Or the update operation may fail for an entire batch of documents because one document in the batch cannot be indexed for some reason. In these cases, the UPDATE_FTINDEX method changes the update_count attribute for the content object associated with the bad content to the negative complement of the update_count value in the fulltext index object. For example, if the update_count of the fulltext index object is 5, the update_count attribute of the bad content object is set to -5 (negative 5).

After the update is finished, MARK_FOR_RETRY is used as part of the recovery procedure. (Refer to Troubleshooting, page 273 in the *Content Server Administrator's Guide* for details on the recovery procedure.)

Before executing this method, set the FullTextMgr job to inactive. Additionally, this method can't be executed against an index if another fulltext operation, such as an update or mark all, is already executing against the index.

Related Administration Methods

MARK_ALL, page 237

Examples

These examples find all the content objects in the index that have an update_count value of -5 and marks them for a retry:

```
EXECUTE mark_for_retry
WITH NAME = 'storage1_index',UPDATE_COUNT = 5
```

```
dmAPIGet("apply,c,NULL,MARK_FOR_RETRY,
NAME,S,storage1_index,UPDATE_COUNT,I,-5")
```

MIGRATE_CONTENT

Purpose

Moves content files from one file store storage area to another.

Syntax

To migrate a single object:

```
EXECUTE migrate_content [FOR]content_obj_id
WITH target_store='target_storage_name'
[,remove_original=value][,log_file='log_file_path']

dmAPIGet("apply,session,content_obj_id,MIGRATE_CONTENT,
TARGET_STORE,S,target_storage_name[,REMOVE_ORIGINAL,B,T|F]
[,LOG_FILE,S,log_file_path]")
```

To migrate all objects in a particular storage area:

```
EXECUTE migrate_content WITH source_store='source_storage_name',
target_store='target_storage_name',log_file='log_file_path'
[,max_migrate_count=value][,batch_size=value]
[,remove_original=value]

dmAPIGet("apply,session,NULL,MIGRATE_CONTENT,
SOURCE_STORE,S,source_storage_name,
TARGET_STORE,S,target_storage_name,LOG_FILE,S,log_file_path
[,MAX_MIGRATE_COUNT,I,value][,BATCH_SIZE,I,value]
[,REMOVE_ORIGINAL,B,T|F]
```

To migrate a set of objects identified by a DQL query:

```
EXECUTE migrate_content WITH target_store='target_storage_name',query=
'DQL_predicate',log_file='log_file_path'
[,max_migrate_count=value][,batch_size=value]
[,remove_original=value]

dmAPIGet("apply,session,NULL,MIGRATE_CONTENT,
TARGET_STORE,S,target_storage_name,QUERY,S,dql_predicate
[,LOG_FILE,S,log_file_path][,MAX_MIGRATE_COUNT,I,value]
[,BATCH_SIZE,I,value][,REMOVE_ORIGINAL,B,T|F]
```

Arguments

Table 3–38. MIGRATE_CONTENT Arguments

Argument	Datatype	Value	Description
SOURCE_STORE	S	*source_storage_name*	Name of the storage area whose content you wish to migrate. Use the name of the file store object. The storage area must be in the read-only state.

Argument	Datatype	Value	Description
TARGET_STORE	S	*target_storage_name*	Name of the storage area to which you are migrating the content. Use the name of the file store object.
REMOVE_ ORIGINAL	B	T (TRUE) or F (FALSE)	Whether to remove the content file from the source storage area. If set to T, the content is removed. If set to F, the content is not removed. If set to F, you must include the LOG_FILE argument also. The default is T.
LOG_FILE	S	*log_file_path*	Identifies where to log messages generated by the method. You must include this argument if you are moving all content from one storage area to another or if you are using a DQL predicate to select content for migration.
MAX_MIGRATE_ COUNT	I	*value*	Defines the maximum number of content files to migrate.
BATCH_SIZE	I	*value*	Defines how many content files to include in a single transaction during the migration operation. The default is 500.
QUERY	S	*dql_predicate*	Specifies a DQL predicate to be used to select content files for migration. This must be a valid WHERE clause qualification.

Return Value

The method returns a collection with one query result object. The object has one integer attribute, named result, whose value is the number of content object successfully migrated.

Permissions

You must have Superuser privileges to execute this method.

General Notes

This method operates on file store storage areas only. You cannot use this method on content files in blob, turbo, external, or CA storage. Nor can you use this method on content stored in components of distributed storage areas (even if they are file stores).

Use this method to move content files from one storage area to another. This method moves content files associated with immutable objects as well as changeable objects. You must use this method if you want to move content files from an unencrypted storage area to an encrypted storage area. The method allows you to move one file, all files in a particular storage area, or a set of files selected by a DQL WHERE clause qualification.

If the source storage area has a full-text index, the target storage must have a full-text index. During the migration operation, the index entries in the source index are marked for deletion.

You can move content from a source storage area with no index to a target storage area with an index. You cannot move content from area with an index to an area without an index.

The method operates on dmr_content objects, resetting their storage_id attribute to the new storage area and resetting the data ticket attribute. It also updates the i_vstamp attribute. The i_vstamp attribute is also incremented for any SysObject that contains the content file as the first primary content (page 0). (Consequently, if the affected content objects are associated with replicated SysObjects, the next replication job for those objects will refresh the objects.) Similarly, if the content objects are associated with persistently cached objects, the next time the cached objects are accessed by the client, they will be refreshed.

Here are some general guidelines for using MIGRATE_CONTENT:

- Make sure that none of the objects whose content you intend to migrate are checked out. If the method attempts to migrate content associated with a checked out object, the method fails with an error.
- Back up the Docbase and the file store storage areas before using the method.
- Make sure that the target storage area has enough disk space to hold the migrated content.
- If you are moving all content from one file store to another file store, if the filestores are indexed, drop the indexes for the source storage area before executing the method.
- If you choose not to remove the original content, that content becomes an orphaned content file and can be removed using dmfilescan.
- If the target storage area has an index, it is recommended that you run UPDATE_FTINDEX on that index after completing the migration.

Controlling the Size of the Operation

If you are moving a large number of files, you can control the operation using the MAX_MIGRATE_COUNT and BATCH_SIZE arguments. MAX_MIGRATE_COUNT controls how many content files are moved with each execution of the method. Use this argument if you have a large number of files to migrate and want to perform the migration in smaller steps rather than all in one operations. BATCH_SIZE controls how many content files are moved in a single transaction within the migration operation. Setting BATCH_SIZE can help protect the operation from overrunning system and database resource limits during the operation.

If you set MAX_MIGRATE_COUNT, make sure that each execution of the method does not attempt to move content files that have already been migrated in previous executions of the method.

Including a Query

If you include a DQL predicate, the query built from the predicate is applied to dmr_content objects. It is recommended that you use a query that references a unique value in the content object. For example, the storage area ID—there is only one storage area in the Docbase with any given storeage ID.

Log File Use

You must identify a log file location if you are migrating an entire storage area or including a DQL predicate or setting REMOVE_ORIGINAL to false in the method arguments. Specifying a log file location is only optional if you are migrating just a single object. For all other cases, you must specify a log file location. If the file you specify already exists, the method appends to that file.

The method logs the following messages:

- A success message for each content object successfully migrated without errors.
- A failure message for each content object that was not successfully migrated. The message includes the reason for the failure.
- Messages for all database errors. (These are written to the server log and the session log file.)
- A success or failure message for each batch in the operation.

Note: If you are migrating a single content file, you can retrieve messages using the Getmessage method.

Related Administration Methods

None

Examples

These two examples migrate a single content file, represented by the content object 0600000272ac100d:

```
EXECUTE migrate_content FOR 0600000272ac100d
WITH target_store='filestore_02'

dmAPIGet("apply,s0,0600000272ac100d,MIGRATE_CONTENT,
TARGET_STORE,S,filestore_02")
```

The next two examples migrate all the content from filestore_01 to engr_filestore:

```
EXECUTE migrate_content
WITH source_store='filestore_01',
target_store='engr_filestore',batch_size=100,
log_file='C:\temp\migration.log'

dmAPIGet("apply,s0,NULL,MIGRATE_CONTENT,
SOURCE_STORE,S,filestore_01,
TARGET_STORE,S,engr_filestore,BATCH_SIZE,I,100,
LOG_FILE,S,C:\temp\migration.log
```

The final two examples use a DQL predicate to select the content to migrate. The query uses content size to select the content.

```
EXECUTE migrate_content
WITH target_store='engr_filestore',
query='content_size>1000',max_migrate_count=1000,
batch_size=100,log_file='C:\temp\migration.log'

dmAPIGet("apply,s0,NULL,MIGRATE_CONTENT,
TARGET_STORE,S,engr_filestore,
QUERY,S,content_size>1000,MAX_MIGRATE_COUNT,I,1000
BATCH_SIZE,I,100,LOG_FILE,S,C:\temp\migration.log
```

MODIFY_TRACE

Purpose

Turns tracing on and off for full-text index operations.

Syntax

```
EXECUTE modify_trace
WITH subsystem='fulltext',value='tracing_level'
```

```
dmAPIGet("apply,session,NULL,MODIFY_TRACE
SUBSYSTEM,S,fulltext,VALUE,S,tracing_level
```

Arguments

Table 3–39. MODIFY_TRACE Arguments

Argument	Datatype	Value	Description
SUBSYSTEM	S	fulltext	The keyword fulltext indicates that you want to turn on tracing for the full-text indexing operations.
VALUE	S	*tracing_level*	The tracing_level must be one of: *none,* to turn off tracing *documentum,* to log only Content Server messages *verity,* to log only Verity Full-Text Engine messages *all,* to log both Content Server and Verity messages

Return Value

MODIFY_TRACE returns a collection with one query result object. The object has one Boolean attribute that indicates the success (TRUE) or failure (FALSE) of the operation.

Permissions

You must have Sysadmin or Superuser privileges to use this method.

General Notes

MODIFY_TRACE turns tracing on and off for all full-text index operations except the MARK_ALL method. (MARK_ALL has its own tracing faciltity.) You can set the tracing level to log information from Content Server, the Verity Full-Text Engine, or both. Alternatively, you can turn tracing off. When tracing is on, tracing is performed for all sessions and the trace information is placed in the server's log file.

If the trace level is set to verity or all, the mkvdk tracing for the UPDATE_FTINDEX and CLEAN_FTINDEX methods includes informational, verbose, and debug messages. (mkvdk is a Verity utility used by UPDATE_FTINDEX and CLEAN_FTINDEX to build and maintain collections. The default messages logged for mkvdk include only fatal, error, warning, and status messages. For more information about mkvdk tracing, refer to the *Content Server Administrator's Guide*.

If the tracing level is set to documentum or all, the batch files created by an index update are saved in the fulltextbatches directory. The files are named using the prefix save. They are removed the next time an update is executed for the index.

All trace messages include time stamps and process and thread ID information.

If the method returns FALSE, the level of tracing remains unchanged. For example, if tracing is currently set at all and you execute MODIFY_TRACE to reset the level to documentum, but the method returned FALSE, the trace level remains at the all level.

Related Administration Methods

LOG_OFF, page 232
LOG_ON, page 233
SET_OPTIONS, page 287

Examples

The following examples turn on full tracing (Documentum and Verity):

```
EXECUTE modify_trace
WITH subsystem = 'fulltext',value = 'all'
```

```
dmAPIGet("apply,c,NULL,MODIFY_TRACE,
SUBSYSTEM,S,fulltext,VALUE,S,all")
```

MOVE_INDEX

Purpose

Moves an existing object type index from one tablespace or segment to another. (This method is not supported for servers running against DB2.)

Syntax

```
EXECUTE move_index FOR 'dmi_index_obj_id'
WITH name = 'new_home'

dmAPIGet("apply,session,dmi_index_obj_id,MOVE_INDEX,
NAME,S,new_home")
```

Arguments

Table 3–40. MOVE_INDEX Arguments

Argument	Datatype	Value	Description
NAME	S	*new_home*	Identifies the new tablespace or segment for the index. Use the name of the tablespace or segment.

Return Value

MOVE_INDEX returns a collection with one query result object. The object contains one Boolean attribute that indicates the success (TRUE) or failure (FALSE) of the operation.

Permissions

You must have Sysadmin or Superuser privileges to use this method.

General Notes

MOVE_INDEX moves an existing object type index from one tablespace or segment to another. You can obtain the index's object ID from the index's index object. Refer to Chapter 2, Object Reference, of the *Content Server Object Reference Manual* for a list of the attributes defined for the index type.

Related Administration Methods

DROP_INDEX, page 188

FINISH_INDEX_MOVES, page 196
MAKE_INDEX, page 235

Examples

These examples move the index represented by 1f0000012643124c to the index2 tablespace:

```
EXECUTE move_index FOR '1f0000012643124c'
WITH name = 'index2'
```

```
dmAPIGet("apply,s0,1f0000012643124c,MOVE_INDEX,
NAME,S,index2")
```

PING

Purpose

Determines if the client still has an active server connection.

Syntax

```
dmAPIGet("apply,c,NULL,PING[,RETRY_ON_TIMEOUT,B,T|F]")
```

You cannot use the EXECUTE statement to invoke PING.

Arguments

Table 3–41. PING Arguments

Argument	Datatype	Value	Description
RETRY_ON _TIMEOUT	B	T (TRUE) or F (FALSE)	T (TRUE) forces a connection attempt between the client and the server. The default is F.

Return Value

PING returns a collection with one query result object. The object has one attribute, called result, that is TRUE if the connection is alive. If the connection has timed out, a null collection is returned.

Permissions

Anyone can use this method.

General Notes

None

Related Administration Methods

None

Example

This example invokes PING with the RETRY_ON_TIMEOUT argument set to TRUE:

```
dmAPIGet("apply,c,NULL,PING,RETRY_ON_TIMEOUT,B,T")
```

PURGE_AUDIT

Purpose

Removes an audit trail entry from the Docbase.

Syntax

```
EXECUTE purge_audit WITH delete_mode='mode'
[,date_start='start_date',date_end='end_date']
[,id_start='start_id',id_end='end_id']
[,object_id='object_id'][,dql_predicate='predicate']
[,purge_non_archived=T|F][,purge_signed=T|F][,commit_size=value]
```

```
dmAPIGet("apply,session,NULL,PURGE_AUDIT,DELETE_MODE,S,mode[,DATE_
START,S,start_date,DATE_END,S,end_date][,ID_START,S,start_
id,ID_END,S,end_id][,OBJECT_ID,S,object_id][,DQL_
PREDICATE,S,predicate][,PURGE_NON_ARCHIVED,B,T|F][,PURGE_
SIGNED,B,T|F][,COMMIT_SIZE,I,value]
```

Arguments

Table 3–42. PURGE_AUDIT Arguments

Argument	Datatype	Value	Description
DELETE_MODE	S	*mode*	Defines how the entries to be deleted are chosen. Valid values are: • DATE_RANGE Deletes all audit trail entries generated within a specified date range. If you include this, include the DATE_START and DATE_END arguments. • ID_RANGE Deletes all audit trail entries whose object IDs fall within a specified range. If you include this, include the ID_START and ID_END arguments. • ALL_VERSIONS Deletes all audit trail entries whose audited_obj_id value corresponds to a specified object ID. ALL_VERSIONS is valid only for audit trail entries whose audited_obj_id identifies a SysObject or SysObject subtype. If you include ALL_VERSIONS, include the OBJECT_ID argument. • SINGLE_VERSION Deletes all audit trail entries whose audited_obj_id value corresponds to a specified object ID. If you include SINGLE_VERSION, include the OBJECT_ID argument. • AUDIT_RECORD Deletes the audit trail entry whose object ID corresponds to a specified object ID. If you include AUDIT_RECORD, include the OBJECT_ID argument.

Argument	Datatype	Value	Description
DELETE_MODE	S	*mode*	• PREDICATE Deletes all audit trail entries that satisfy a DQL predicate. If you include PREDICATE, include the DQL_PREDICATE argument.
DATE_START	S	*start_date*	The starting date of the entries to be deleted. The date is compared to the value in the time_stamp attribute in the audit trail entry. (time_stamp records the local time at which the entry was generated.) The date format can be any acceptable format that does not require aa pattern specification. (Refer to Date Literals, page 15, for a list of valid formats. If you include DATE_START, you must include END_DATE also. Include a start and end date only when DELETE_MODE is DATE_RANGE.

Argument	Datatype	Value	Description
DATE_END	S	*end_date*	The ending date of the entries to be deleted. The date is compared to the value in the time_stamp attribute in the audit trail entry. (time_stamp records the local time at which the entry was generated.) The date format can be any acceptable format that does not require aa pattern specification. (Refer to Date Literals, page 15, for a list of valid formats. If you include END_DATE, you must also START_DATE. The end date must be greater than the start date. Include a start and end date only when DELETE_MODE is DATE_RANGE.
ID_START	S	*start_id*	The object ID of an audit trail entry. Include ID_START if the DELETE_MODE is ID_RANGE. You must also include ID_END.
ID_END	S	*end_id*	The object ID of an audit trail entry. Include ID_END if the DELETE_MODE is ID_RANGE. You must also include ID_START. The ID_END object ID must be larger than the object ID identified in ID_START

Argument	Datatype	Value	Description
OBJECT_ID	S	*object_id*	Include this argument if DELETE_MODE is ALL_VERSIONS, SINGLE_VERSION, SINGLE_ID, or AUDIT_RECORD. For all modes except AUDIT_RECORD, *object_id* is the object ID recorded in the audited_obj_id attribute of the audit trail entries. For AUDIT_RECORD mode, *object_id* is the object ID of an audit trail entry.
PURGE_NON_ ARCHIVED	B	T (TRUE) or F (FALSE)	Determines whether unarchived audit trial entries are deleted. Setting this argument to T (TRUE) directs Content Server to delete audit trail entries whose i_is_archived attribute is set to F (FALSE). The default for this argument is F (FALSE), meaning that only entries whose i_is_archived attribute is set to T (TRUE) are removed. You cannot set this to T if the DQL_PREDICATE argument is included.

Argument	Datatype	Value	Description
PURGE_SIGNED	B	T (TRUE) or F (FALSE)	Determines whether audit trail entries for dm_adddigsignature and dm_addesignature events are considered for deletion. Setting this argument to T (TRUE) means entries for dm_adddigsignature and dm_addesignature events are considered for deletion. The default is F (FALSE), meaning the dm_adddigsignature and dm_addesignature entries are not considered for deletion. You cannot set this to T if the DQL_PREDICATE argument is included.
DQL_PREDICATE	S	*predicate*	Defines a DQL predicate to choose the audit trail entries to be deleted. A valid predicate is that portion of a DQL SELECT statement that occurs after the FROM keyword. If you include this argument, you may not include PURGE_NON_ARCHIVED or PURGE_SIGNED.
COMMIT_SIZE	I	*value*	Defines how many audit trail entries to delete in each transaction within the overall operation. The default is 1000. Setting this to 0 means that all entries are deleted in one transaction.

Return Value

The PURGE_AUDIT method returns a collection with one query result object. The query result object has two attributes, result and deleted_objects. The result attribute is set to T if the method completed successfully and F if the method did not complete successfully. deleted_objects records the number of audit trail entries that were deleted.

Permissions

You must have Purge Audit privileges to execute this method.

General Notes

Executing the PURGE_AUDIT method always generates at least one audit trail entry with the event name dm_purgeaudit. The operation generates one audit trail entry for each transaction within the operation. For example, if you set COMMIT_SIZE to 100 and the operation deletes 700 audit trail entries, the operation generates 7 audit trail entries, one for each transaction.

The entry for each transaction has the event name dm_purgeaudit. The optional attributes record the following information for the transaction:

- string_1 stores the time_stamp value of the first audit trail entry deleted in the transaction
- string_2 stores the time_stamp value of the last audit trail entry deleted in the transaction
- string_3 stores the actual number of audit trail entries deleted by the transaction
- string_5 stores the entire list of arguments from the method's command line
- id_1 stores the object ID of the first audit trail object deleted in the transaction
- id_2 stores the object ID of the last audit trail object deleted in the transaction

Related Administration Methods

None

Examples

This example deletes all archived audit trail entries generated from January 1, 2003 to January 1, 2004:

```
EXECUTE purge_audit WITH DELETE_MODE='DATE_RANGE',
date_start='01/01/2003 00:00:00 AM',
date_end='01/01/2004 00:00:00 AM'

dmAPIGet("apply,c,NULL,PURGE_AUDIT,
DELETE_MODE,S,DATE_RANGE,
DATE_START,S,01/01/2003 00:00:00 AM,
DATE_END,S,01/01/2004 00:00:00 AM")
```

This example deletes all audit trail entries generated from January 1, 2003 to January 1, 2004, including unarchived entries. The number of entries deleted in each transaction is set to 500:

```
EXECUTE purge_audit WITH delete_mode='DATE_RANGE',
date_start='01/01/2003 00:00:00 AM',
date_end='01/01/2004 00:00:00 AM'
purge_non_archived=T,commit_size=500

dmAPIGet("apply,c,NULL,PURGE_AUDIT,
DELETE_MODE,S,DATE_RANGE,
DATE_START,S,01/01/2003 00:00:00 AM,
DATE_END,S,01/01/2004 00:00:00 AM,
PURGE_NON_ARCHIVED,B,T,COMMIT_SIZE,I,500")
```

This example deletes all archived audit trail entries that identify the document 09000003ac5794ef as the audited object:

```
EXECUTE purge_audit WITH delete_mode='ALL_VERSIONS',
object_id='09000003ac5794ef'

dmAPIGet("apply,c,NULL,PURGE_AUDIT,
DELETE_MODE,S,ALL_VERSIONS,
OBJECT_ID,S,09000003ac5794ef")
```

This example deletes the single audit trail entry whose object ID is 5f0000021372ac6f:

```
EXECUTE purge_audit WITH delete_mode='AUDIT_RECORD',
object_id='5f0000021372ac6f'

dmAPIGet("apply,c,NULL,PURGE_AUDIT,
DELETE_MODE,S,AUDIT_RECORD,
OBJECT_ID,S,5f0000021372ac6f")
```

This example deletes all audit trail entries whose object IDs range from 5f1e9a8b003a901 to 5f1e9a8b003a925, including unarchived entries:

```
EXECUTE purge_audit WITH delete_mode='ID_RANGE',
id_start='5f1e9a8b003a901',id_end='5f1e9a8b003a925',
purge_non_archived=T

dmAPIGet("apply,c,NULL,PURGE_AUDIT,
DELETE_MODE,S,ID_RANGE,ID_START,S,5f1e9a8b003a901,
ID_END,S,5f1e9a8b003a925,PURGE_NON_ARCHIVED,B,T")
```

This example deletes all audit trail entries that satisfy the specified DQL predicate:

```
EXECUTE purge_audit WITH delete_mode='PREDICATE',
dql_predicate=
'dm_audittrail where event_name like ''dcm%''and
 r_gen_source=0'
```

If you need to include single-qoutes in the predicate string, (for example, 'dcm%'), escape the single-quotes with single-quotes. This is illustrated in the example above.

```
dmAPIGet("apply,c,NULL,PURGE_AUDIT,
DELETE_MODE,S,PREDICATE,DQL_PREDICATE,S,dm_audittrail
where event_name like 'dcm%' and r_gen_source=0")
```

PURGE_CONTENT

Purpose

Sets a content file off line and deletes the file from its storage area.

Syntax

```
EXECUTE purge_content FOR 'content_object_id'

dmAPIGet("apply,c,content_obj_id,PURGE_CONTENT")
```

Arguments

PURGE_CONTENT has no arguments.

Return Value

PURGE_CONTENT returns a collection with one query result object. The object has one Boolean attribute whose value indicates the success (TRUE) or failure (FALSE) of the operation.

Permissions

You must have Sysadmin or Superuser user privileges to use this method.

General Notes

PURGE_CONTENT marks a content file as off-line and deletes the file from the storage area. Do not use this method unless you have previously moved the file to an archival or back up storage area. PURGE_CONTENT does not back up the file. The method only deletes the file from the storage area and sets the is_offline attribute of the file's content object to TRUE.

To use PURGE_CONTENT, you must be using one server for both data and content requests. If the configuration is set up for content servers, you must issue a Connect method that bypasses the content server to use PURGE_CONTENT in the session.

Examples

These examples delete the content file associated with the content object represented by 060000018745231c and set the is_offline attribute in the associated content object to TRUE:

```
EXECUTE purge_content FOR '060000018745231c'

dmAPIGet("apply,c,060000018745231c,PURGE_CONTENT")
```

PUSH_CONTENT_ATTRS

Purpose

Updates content metatdata in a content addressable systems.

Syntax

```
EXECUTE push_content_attrs FOR object_id
WITH format=format_name [,page=number]
[,page_modifier=value]
```

```
dmAPIGet("apply,session,object_id,PUSH_CONTENT_ATTRS,
FORMAT,S,format_name[,PAGE,I,number]
[,PAGE_MODIFIER,S,value]")
```

object_id is the ID of a document or other SysObject.

Arguments

Table 3–43. PUSH_CONTENT_ATTRS Arguments

Argument	Datatype	Value	Description
FORMAT	S	*name*	The file format of the content file. Use the name of the format object that defines the format.
PAGE	I	*number*	The page number of the content file in the document. If the content file is a rendition, this is the page number of the primary content with which the rendition is associated. The default value is 0.
PAGE_MODIFIER	S	*value*	This identifies a rendition. It is the page modifier defined for the rendition when the rendition was added to the document. The value is stored in the page_modifier attribute in the content object.

Return Value

PUSH_CONTENT_ATTRS returns a collection with one query result object. The object has one Boolean attribute that contains TRUE if the method was successful and FALSE if unsuccessful.

Permissions

To execute this method you must be a superuser and you must have at least Write permission on the object identified in *object_id*.

General Notes

PUSH_CONTENT_ATTRS updates the metadata fields in a content-addressed storage system for a particular content file. First, the method reads the values in the following content-related attributes in the file's associated content object:

- content_attr_name
- content_attr_value
- content_attr_data_type
- content_attr_num_value
- content_attr_date_value

content_attr_name identifies the metadata fields to be set in the storage system. The content_attr_data_type attribute identifies the data type of the value users must provide for each metadata field in the corresponding index position in content_attr_name. The actual value is defined in one of the remaining attributes, depending on the field's datatype. For example, if the field name is "title" and its datatype is character string, the content_attr_value attribute contains the actual title value. (The remaining attributes, content_attr_num_value and content_attr_date_value are set to the default NULLINT and NULLDATE values.)

After the method reads the content object attributes, it invokes the content-addressed plugin library to update the storage system metadata fields.

Note: To be updated, a field must be defined in both the content object's content_attr_name attribute and in the storage object's a_content_attr_name attribute. If a field is named in the content object's content_attr_name attribute but not defined in the storage object's a_content_attr_name object, it is not set in the storage area. Similarly, if a field is named in the storage object's a_content_attr_name but not in the content object's content_attr_name attribute, it is not set in the storage area.

If PUSH_CONTENT_ATTRS completes successfully, it generates a new content address for the content. The new address is appended to the i_contents attribute of the subcontent object that records content addresses for the content file.

For complete information about how to save a document to content-addressed storage, refer to Creating SysObjects, page 108, in *Content Server Fundamentals*.

Related Administration Methods

SET_CONTENT_ATTRS, page 283

There is also a corrsponding DMCL API method, Setcontentattrs.

Examples

```
EXECUTE push_content_attrs FOR 090000026158a4fc
WITH format=txt,page=1
```

```
dmAPIGet("apply,s0,090000026158a4fc,PUSH_CONTENT_ATTRS,
FORMAT,S,txt,PAGE,I,1")
```

REGISTER_ASSET

Purpose

Queues a request to the Media Server to generate a thumbnail, proxies, and metadata for a rich media content file.

Syntax

```
EXECUTE register_asset FOR object_id
WITH [page=page_number][,priority=priority_level]
```

```
dmAPIGet("apply,session,object_id,REGISTER_ASSET
[,PAGE,I,page_number][,PRIORITY,I,priority_level]")
```

object_id is the object ID of the document that contains the content file.

Arguments

Table 3–44. REGISTER_ASSET Arguments

Argument	Datatype	Value	Description
PAGE	I	*page_number*	Page number of the content file. If unspecified, the default is 0.
PRIORITY	I	*priority_level*	Identifies the priority of the request. A value of zero indicates no priority. Priority rises as the value rises. The Media Server processes higher priority requests before lower priority requests. If unspecified, the default is 5.

Return Value

REGISTER_ASSET returns a collection with one query result object. The object has one Boolean attribute whose value indicates success (TRUE) or failure (FALSE) of the operation.

Permissions

You must have at least Version permission on the object or Sysadmin privileges to use this method.

General Notes

REGISTER_ASSET is called by Content Server whenever an object with rich media content is saved or checked in to the Docbase. The method generates an event, dm_register_event, that is queued to the Media Server. When the Media Server processes the event, the server creates a thumbnail, proxies, and metadata for the content.

The dmi_queue_item that represents the event is queued to the dm_mediaserver user, who represents the Media Server. (dm_mediaserver is created when Documentum Media Services is installed.) REGISTER_ASSET sets the following attributes in the queue item:

Table 3–45. Queue Item Attribute Values Set by REGISTER_ASSET

Attribute	Set to
event	dm_register_event
item_id	object ID of the SysObject that triggered the request
instruction_page	page number of the content in the SysObject
date_sent	date the request was made
name	dm_mediaserver
sent_by	session user
priority	priority level identified in the REGISTER_ASSET call

Related Administration Methods

SET_CONTENT_ATTRS, page 283
TRANSCODE_CONTENT, page 300

Example

```
EXECUTE register_asset FOR 090002410063ec21
WITH page=0,priority=8
```

```
dmAPIGet("apply,s0,090002410063ec21,REGISTER_ASSET,
PAGE,I,0,PRIORITY,I,8")
```

REORGANIZE_TABLE

Purpose

Reorganizes a database table for query performance.

Syntax

```
EXECUTE reorganize_table WITH table_name='name',
[,index='index_name']

dmAPIGet("apply,session,NULL,REORGANIZE_TABLE,
TABLE_NAME,S,name[,INDEX,S,index_name]
```

Arguments

Table 3–46. REORGANZE_TABLE Arguments

Argument	Datatype	Value	Description
TABLE_NAME	S	*name*	Name of the RDBMS table to be reorganized.
INDEX	S	*index_name*	Name of an index on the table identified in TABLE_NAME. This argument is required for an Oracle database table. It is optional for a SQL Server or DB2 database table. It is not used for Sybase database table. With the exception of Sybase, which doesn't use this argument, the argument is used differently on each database. Refer to the General Notes for details.

Return Value

REORGANIZE_TABLE returns a collection with one query result object. The object has one attribute, named result, that contains T if the method was successful or F if the method was unsuccessful.

Permissions

You must be a superuser to execute this method.

Caution Do not use this method unless directed to do so by Technical Support.

General Notes

This method is used by the UpdateStatistics administration tool, in conjunction with UDPATE_STATISTICS, when the tool is run on a DB2 database. However, the method can be used on any supported database.

The INDEX Argument

On Oracle, you must include the INDEX argument. The method rebuilds the specified index. The index must be an index on the table identified in TABLE_NAME.

For SQL Server, the argument is optional. If you include it, you must specify an index on the table identified in TABLE_NAME. The method rebuilds that index. If you do not include the argument, the method rebuilds all indexes on the specified database table.

For DB2, the argument is optional. If you include it, you must specify an index on the table identified in TABLE_NAME. The method rebuilds the database table based on the ordering defined in the index.

Caution Using the INDEX argument on a DB2 database table is a powerful feature. The choice of index to use as a basis for re-ordering the table can greatly affect query performance against that table— for better or worse, depending on the index choice. It is recommended that if you use this argument, specify the index on the r_object_id attribute.

Note: Indexes created by Content Server are named using the following format: D_*index_obj_id*, where *index_obj_id* is the object ID of the dmi_index object for the index.

The INDEX argument is not used when the method is run against a Sybase table.

Related Administration Methods

UPDATE_STATISTICS, page 309

Examples

These two examples reorganize the dm_sysobject_s table. (Note that these two examples do not work on Oracle because the required INDEX argument is not included.)

```
EXECUTE reorganize_table
WITH table_name='dm_sysobject_s'

dmAPIGet("apply,S0,NULL,REORGANIZE_TABLE,
TABLE_NAME,S,dm_sysobject_s")
```

This example rebuilds an index associated with the dm_sysobject_s table.

```
EXECUTE reorganize_table
WITH table_name='dm_sysobject_s,
index='D_1f0C9fda80000108'

dmAPIGet("apply,S0,NULL,REORGANIZE_TABLE,
TABLE_NAME,S,dm_sysobject_s,
INDEX,S,D_1f0C9fda80000108")
```

REPLICATE

Purpose

Copies content files in distributed storage areas.

Syntax

```
EXECUTE replicate
WITH query='value',store='value'
[,type='value']
```

```
dmAPIGet("apply,session,NULL,REPLICATE,
 QUERY,S,value,STORE,S,name[,TYPE,S,type_name]")
```

Arguments

Table 3–47. REPLICATE Arguments

Argument	Datatype	Value	Description
QUERY	S	*dql_predicate expression*	This argument is required. The predicate expression is used to build a DQL query that selects the objects whose content you want to copy. The expression can be any expression that would be a valid WHERE clause qualification (refer to The WHERE Clause, page 130).
STORE	S	*name*	This argument is required. It identifies where to put the new content copy or copies. The name must be the name of a component of a distributed storage area.
TYPE	S	*type_name*	This argument is optional. It identifies the type of objects whose content you are replicating. *type_name* must be a direct or indirect subtype of dm_sysobject. The default is dm_sysobject.

Return Value

REPLICATE returns a collection with one query result object. The object has one Boolean attribute whose value indicates the success (TRUE) or failure (FALSE) of the operation.

Permissions

You must have Sysadmin or Superuser privileges to use REPLICATE.

General Notes

REPLICATE copies content files from one component of a distributed storage area to another. Typically, replication of content files is performed by the ContentReplication or surrogate get. Use REPLICATE as a manual backup for these tools. (Refer to Content Replication , page 403, in the *Content Server Administrator's Guide* for information about the ContentReplication and to Using the Surrogate Get Feature, page 42, in the *Distributed Configuration Guide* for information about surrogate get.)

REPLICATE checks the contents stored in the storage area at a specified site and copies back to the local storage area any contents that meet the conditions defined in the function. (Refer to the *Distributed Configuration Guide* for more information.)

To use REPLICATE, you must be using one server for both data and content requests. If the configuration is set up for content servers, you must issue a Connect method that bypasses the content server to use REPLICATE in the session.

Related Administration Methods

DELETE_REPLICA , page 176
IMPORT_REPLICA, page 216

Examples

These examples replicate all content for documents of the subtype proposals owned by jenniferk:

```
EXECUTE replicate WITH query='owner_name=jennyk',
store='distcomp_1',type='proposals'
```

```
dmAPIGet("apply,c,NULL,REPLICATE,
QUERY,S,owner_name='jennyk',
STORE,S,distcomp_1,TYPE,S,proposals")
```

The next example replicates the content of all objects whose content is authored by Jane:

```
dmAPIGet("apply,s0,NULL,REPLICATE,
QUERY,S,any authors in ('Jane'),
STORE,S,diststorage3")
```

RESET_FTINDEX

Purpose

Reinitializes an index.

Syntax

```
EXECUTE RESET_FTINDEX [[FOR] 'fulltext_index_obj_id']
[name='fulltext_index_name']
```

With EXECUTE, do not include the FOR clause if you include the NAME argument.

```
dmAPIGet("apply,session,fulltext_index_obj_id,RESET_FTINDEX
[,NAME,S,fulltext_index_name]
```

With Apply, specify the fulltext index object ID as NULL if you include the NAME argument.

Arguments

Table 3–48. RESET_FTINDEX Arguments

Argument	Datatype	Value	Description
NAME	S	*name*	Identifies the index to reinitialize. Use the name associated with the index's fulltext index object.

Return Value

RESET_FTINDEX returns a collection with one query result object. The object has one Boolean attribute whose value indicates the success (TRUE) or failure (FALSE) of the operation.

Permissions

You must have Sysadmin or Superuser privileges to use this method.

General Notes

RESET_FTINDEX reinitializes an index. It removes the existing index and marks all the indexable content in the storage area for indexing. This method is useful if you need to rebuild an index (for example, after changing the indexed attributes of a type that already has indexed objects of that type).

Before executing this method, set the FullTextMgr job to inactive. Additionally, this method can't be executed against an index if another fulltext operation, such as an update or mark all, is already executing against the index.

After you execute RESET_FTINDEX, execute UPDATE_FTINDEX to recreate the index.

Related Administration Methods

UPDATE_FTINDEX, page 303

Examples

The following example resets the testing index:

```
EXECUTE reset_ftindex
WITH NAME = 'testing'
```

The following example reinitializes the storage1_index full-text index:

```
dmAPIGet("apply,c,NULL,RESET_FTINDEX,
NAME,S,storage1_index")
```

RESTORE_CONTENT

Purpose

Restores an off-line content file to its original storage area.

Syntax

```
EXECUTE restore_content FOR 'content_object_id'
WITH file = 'path_name' [,other file = 'other_path_name']
```

```
dmAPIGet("apply,session,content_object_id,RESTORE_CONTENT,
FILE,S,file_path[,OTHER_FILE,S,other_path_name]")
```

content_obj_id is the object ID of the content object associated with the specified file.

Arguments

Table 3–49. RESTORE_CONTENT Arguments

Argument	Datatype	Value	Description
FILE	S	*file_path*	Specifies the current location of the content file. Use a full path specification.
OTHER_FILE	S	*other_path_name*	Specifies the current location of the resource fork for the content file. Use a full path specification. Include this argument only if the file is a Macintosh file.

Return Value

RESTORE_CONTENT returns a collection with one query result object. The object has one Boolean attribute whose value indicates the success (TRUE) or failure (FALSE) of the operation.

Permissions

You must have Sysadmin or Superuser user privileges to use this method.

General Notes

RESTORE_CONTENT restores an off-line content file to its original storage area. This method only operates on one file at a time. To restore multiple files, use the API Restore method.

The method places the file in the storage area indicated by the storage_id attribute of the file's content object and sets the content object's is_offline attribute to FALSE.

To use RESTORE_CONTENT, you must be using one server for both data and content requests. If the configuration is set up for content servers, you must issue a Connect method that bypasses the content server to use RESTORE_CONTENT in the session.

Examples

The following examples restore the file named Myproposal using an EXECUTE statement:

```
EXECUTE restore_content
WITH file='c:\archive1\Myproposal'
```

or, on UNIX:

```
EXECUTE restore_content
WITH file='u02/archive1/Myproposal'
```

The next two examples perform the same operation using Apply methods:

```
dmAPIGet("apply,c,NULL,RESTORE_CONTENT,
FILE,S,c:\archive1\Myproposal")
```

or, on UNIX:

```
dmAPIGet("apply,c,NULL,RESTORE_CONTENT,
FILE,S,u02/archive1/Myproposal")
```

RMSTYLE_FTINDEX

Purpose

Removes a user-defined topic set or user-defined style files from an index.

Syntax

To remove a topic set:

```
EXECUTE rmstyle_ftindex [[FOR] 'fulltext_index_id']
WITH [name='index_name',]verity_style='topic_trees'
[,install_loc='verity_location_name'][,locale='verity_locale']
```

- With Execute, do not include the FOR clause if you include the NAME argument.

  ```
  dmAPIGet("apply,session,fulltext_index_id,RMSTYLE_FTINDEX,
  [NAME,S,index_name,]VERITY_STYLE,S,topic_trees[,INSTALL_LOC,S,verity_
  location_name][,LOCALE,S,verity_locale]")
  ```

- With Apply, specify the fulltext index ID as NULL if you include the NAME argument.

To remove a style file:

```
EXECUTE rmstyle_ftindex WITH verity_style='style_type'
[,install_loc='verity_location_name'][,locale='verity_locale']
```

```
dmAPIGet("apply,session,NULL,RMSTYLE_FTINDEX,
VERITY_STYLE,S,style_type[,INSTALL_LOC,S,verity_location_
name][,LOCALE,S,verity_locale]")
```

Arguments

Table 3–50. RMSTYLE_FTINDEX Arguments

Argument	Datatype	Value	Description
NAME	S	*name*	Identifies the index whose topic set you want to remove. Specify the name of the index's fulltext index object. Optionally, you can omit the NAME argument and specify the object ID of the fulltext index object. Use this argument (or the fulltext index object ID) only when you are removing an index's topic set.

Argument	Datatype	Value	Description
INSTALL_LOC	S	*verity_location_ name*	Name of the location object that identifies the location of the Verity installation. If included, this must be one of the values in the fulltext_install_loc attribute of the docbase config object. If not included, the default is the value of the verity_location attribute in the server config object.
LOCALE	S	*verity_locale*	Name of a Verity locale. If specified, the style file or topic tree is removed from the specfiied locale subdirectory.

Return Value

RMSTYLE_FTINDEX returns a collection with one query result object. The object has one Boolean attribute whose value indicates the success (TRUE) or failure (FALSE) of the operation.

Permissions

You must have Sysadmin or Superuser privileges to use this method.

Removing Thesaurus and Stop Files

Because the Full-Text Engine requires the presence of a thesaurus file and a stop file, the system does not allow you to remove the thesaurus or stop files if there are no backup thesaurus. Customizing Indexes, page 267, in the *Content Server Administrator's Guide* provides detailed information about how thesaurus and stop files are handled.)

Related Administration Methods

GETSTYLE_FTINDEX, page 208
SETSTYLE_FTINDEX, page 292

Examples

The following EXECUTE examples remove a topic set from the Spanish locale subdirectory. The first statement identifies the index by name and the second by the object ID of the index's fulltext index object.

```
EXECUTE rmstyle_ftindex
WITH name ='storage1_indexSpanish',
verity_style='topic_trees',locale='spanishx'
```

```
EXECUTE rmstyle_ftindex FOR '3b00000113236421'
WITH verity_style='topic_trees',LOCALE,S,spanishx
```

The next statement removes a user-defined lex file:

```
EXECUTE rmstyle_ftindex
WITH verity_style ='lex'
```

The following Apply examples remove the topic tree set for the storage1_index full-text index. The first example identifies the index by name and the second by the object ID of the index's fulltext index object.

```
dmAPIGet("apply,c,NULL,RMSTYLE_FTINDEX,
NAME,S,storage1_index,
VERITY_STYLE,S,topic_trees")
```

```
dmAPIGet("apply,c,3b00000113236421,RMSTYLE_FTINDEX,
VERITY_STYLE,S,topic_trees")
```

The following example removes the stop style file for the index:

```
dmAPIGet("apply,c,NULL,RMSTYLE_FTINDEX,
VERITY_STYLE,S,stp")
```

ROLES_FOR_USER

Purpose

Retrieves the roles assigned to the user in a particular client domain.

Syntax

```
EXECUTE roles_for_user [[FOR] 'dm_user_id']
WITH [USER_NAME=value][,DOMAIN=domain_name]
```

With Execute, do not include the FOR clause if you include the NAME argument.

```
dmAPIGet("apply,session,dm_user_id,ROLES_FOR_USER,
[USER_NAME,S,value][,DOMAIN,S,domain_name]")
```

With Apply, specify the user ID as NULL if you include the NAME argument.

Arguments

Table 3–51. ROLES_FOR_USER Arguments

Argument	Datatype	Value	Description
USER_NAME	S	*user_name*	User whose roles you are retrieving. Identify the user by the user's user name.
DOMAIN	S	*domain_name*	Domain of a client application. If a domain is included, the method returns the user's roles within the specified domain. If a domain is not included, the method returns all roles for the user, regardless of domain.

Return Value

This returns a collection of one query result object that has three attributes: user_name, domain, and roles. user_name contains the name of the user being queried. domain lists the domains searched for user roles. roles is a repeating attribute that contains the roles for the user.

Permissions

Any one can use this method.

General Notes

This method is used by client applications to determine the roles assigned to users who use the applications. For example, when a user starts a session with Desktop Client, DTC executes this method to query the domain group associated with Desktop Client, to determine what roles the user has in Desktop Client.

(For more information about groups, roles, and domains, refer to Chapter 4, Security Services, of *Content Server Fundamentals*.)

Related Administration Methods

None

Examples

```
EXECUTE roles_for_user
WITH user_name=MaryJean,domain="HR_app"
```

```
dmAPIGet("apply,c,NULL,ROLES_FOR_USER,
USER_NAME,S,MaryJean,DOMAIN,S,HR_app")
```

SET_APIDEADLOCK

Purpose

Sets a deadlock trigger on a particular API method.

Syntax

```
EXECUTE set_apideadlock
WITH API=api_name,VALUE=T|F{,API=api_name,VALUE=T|F}
```

```
dmAPIGet("apply,session,NULL,SET_APIDEADLOCK,
API,S,api_name,VALUE,B,T|F{,API,S,api_name,VALUE,B,T|F}")
```

Arguments

Table 3–52. SET_APIDEADLOCK Arguments

Argument	Datatype	Value	Description
API	S	*api_name*	Name of the API method or operation on which to set the deadlock trigger.Table 3–53, page 281 lists the methods and operations on which you can set a trigger.
VALUE	BOOLEAN	T (TRUE) or F (FALSE)	TRUE sets the trigger. FALSE removes the trigger.

Return Value

SET_APIDEADLOCK returns a collection with one query result object. The object has one Boolean attribute whose value indicates the success (TRUE) or failure (FALSE) of the operation.

Permissions

Anyone can use this administration method.

General Notes

Use SET_APIDEADLOCK to test deadlock retry code in client applications that execute in explicit transactions. Operations that occur in explicit transactions are not protected by Content Server's internal deadlock retry functionality. Consequently, applications that execute in an explicit transaction may include their own deadlock retry functionality.

To test the deadlock retry functionality, use SET_APIDEADLOCK to place a trigger on one or more methods or operations executed in the application. When the application is tested, Content

Server simulates a deadlock when one of the methods or operations executes, allowing you to test the deadlock retry code in the application. The simulated deadlock sets the _isdeadlocked computed attribute and issues a rollback to the database.

The deadlock trigger is removed automatically from the method or operation which triggered the simulated deadlock.

Table 3–53, page 281, lists the operations and methods on which you can place a deadlock trigger.

Table 3–53. Valid Operation Names for SET_APIDEADLOCK

Method or Operation	Representing
Acquire	Acquire method
bp_transition	The completion of the bp_transition method
Branch	Branch method
Checkin	Checkin method
Complete	The completion of a work item
Demote	Demote method
Dequeue	Dequeue method
Destroy	Destroy method
exec	Any RPC EXEC call (on behalf of the Query, Readquery, Execquery, or Cachequery methods)
Next	Next method
Promote	Promote method
Queue	Queue method
Resume	Resume method
Revert	Revert method
Save	Save on any object Use this operation to put a deadlock trigger on a Saveasnew method for a SysObject.
save_content	A content save operation during a Save, Checkin, Saveasnew, or Branch operation
save_parts	A containment save operation during a Save, Checkin, Saveasnew, or Branch operation
Suspend	Suspend method

Related Administration Methods

None

Examples

This example sets a deadlock trigger on the Checkin method:

```
EXECUTE set_apideadlock
WITH api=checkin,value=T
```

This example sets a deadlock trigger on the Promote and Revert methods:

```
dmAPIGet("apply,S0,NULL,SET_APIDEADLOCK,
API,S,promote,VALUE,B,T,API,S,revert,VALUE,B,T")
```

This example removes the deadlock trigger from the Checkin method:

```
EXECUTE set_apideadlock
WITH api=checkin,value=F
```

SET_CONTENT_ATTRS

Purpose

Sets the content-related attributes of a content object.

Syntax

```
EXECUTE set_content_attrs FOR object_id
WITH format='format_name',[page=page_number,]
[page_modifier='value',]
parameters='name="value"{,name="value"}'[,truncate=T|F]

dmAPIGet("apply,session,object_id,SET_CONTENT_ATTRS,
FORMAT,S,format_name,[PAGE,I,page_number,]
[PAGE_MODIFIER,S,value,]
PARAMETERS,S,'name="value"{,name="value"}'[,TRUNCATE,B,T|F]")
```

object_id is the object ID of the document that contains the content file.

Arguments

Table 3–54. SET_CONTENT_ATTRS Arguments

Argument	Datatype	Value	Description
FORMAT	S	*format_name*	Name of the content format. This is the name of the format object.
PAGE	I	*page_number*	Page number of the content in the document's set of content files. If unspecified, the default is 0.
PAGE_ MODIFIER	S	*value*	Identifies the rendition. Refer to the General Notes for a detailed description of the purpose of the page modifier.
PARAMETERS	S	*name = value* pairs	Comma separated list of attribute names and values. *value* is one of: "*character_string*" FLOAT(*number*) DATE(*date_value*) Refer to the General Notes for examples.
TRUNCATE	B	T (TRUE) or F (FALSE)	Whether to remove existing values in the content attributes before setting the new values. The default value is F.

Return Value

SET_CONTENT_ATTRS returns a collection with one query result object. The object has one Boolean attribute whose value indicates the success (TRUE) or failure (FALSE) of the operation.

Permissions

You must have at least Write permission on the object or Sysadmin privileges to use this method.

General Notes

SET_CONTENT_ATTRS 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

The Media Server uses this method to store the metadata it generates for a content file in the file's content object.

Using the PARAMETERS Argument

The metadata is specified in SET_CONTENT_ATTRS as a comma-separated list of name and value pairs in the PARAMETERS argument. The format is:

```
'name=value{,name=value}'
```

Because the PARAMETERS argument 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:

```
EXECUTE set_content_attrs FOR 0900038000ab4d13
WITH format='jpeg',page=0,
PARAMETERS='name="photo_closeup"'
```

If the metadata value is a numeric value, use the FLOAT function to specify the value:

```
EXECUTE set_content_attrs FOR 0900038000ab4d13
WITH format='jpeg',page=0,
PARAMETERS='width=FLOAT(12.5)'
```

If the metadata value is a date, use the DATE function to specify the value:

```
EXECUTE set_content_attrs FOR 0900038000ab4d13
WITH format='jpeg',page=0,
PARAMETERS='take_down_date=DATE(09/30/2002)'
```

The method sets the content object attributes with values specified in the PARAMETERS argument beginning at the zero index position. The values are set in the order in which they are included in the PARAMETERS argument.

The content_attr_name and content_attr_data_type values are set to the name of the property and its datatype. If the property's datatype is string, the value is stored in content_attr_value and the remaining two attributes (content_attr_date_value and content_attr_num_value) are set to the default NULL values (NULLDATE and NULLINT). If the property's datatype is numeric, the value is stored in content_attr_num_value and the remaining two attributes (content_attr_value and content_attr_date_value) are set to the default NULL values (NULLSTRING and NULLDATE). If the property's datatype is date, the value is stored in content_attr_date_value and the remaining two attributes (content_attr_num_value and content_attr_value) are set to the default NULL values (NULLINT and NULLSTRING).

For example, suppose an application executes the following statement:

```
EXECUTE set_content_attrs FOR 0900038000ab4d13
WITH format='jpeg',page=0,
PARAMETERS='name="photo_closeup",width=FLOAT(12.5),take_down_date=DATE(09/30/2002)'
```

Table 3–55, page 285, shows the resulting values in the attributes in the content object.

Table 3–55. Example Settings for Content Metadata Attributes in Content Objects

Attribute	Index Position		
	[0]	[1]	[2]
content_attr_name	name	width	take_down_date
content_attr_data_ type	2	4	5
content_attr_value	photo_closeup	NULLSTRING	NULLSTRING
content_attr_num_ value	NULLINT	12.5	NULLINT
content_attr_date_ value	NULLDATE	NULLDATE	09/30/2002

The TRUNCATE argument controls how existing values in the content attributes are handled. If TRUNCATE is set to T (TRUE), existing values in the attributes are removed before the attributes are set to the new names and values. If TRUNCATE is F (FALSE), existing names and values are not removed. If the PARAMETERS argument includes a name that already present in the attributes, the name's value is overwritten. Names specified in the PARAMETERS argument that are not currently present in the attributes are appended to the attributes. The default for TRUNCATE is F.

Using PAGE_MODIFIER

Use the PAGE_MODIFIER argument to define an identifier that distinguishes a 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 SET_CONTENT_ATTRS 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.

Related Administration Methods

PUSH_CONTENT_ATTRS, page 261

Examples

```
EXECUTE set_content_attrs FOR 090002134529cb2e
WITH format='jpeg',page=0,
parameters='name="garage_band",sales=FLOAT(100.2),
release_date=DATE(01/02/2002)'
```

```
dmAPIGet("apply,s0,090002134529cb2e,
SET_CONTENT_ATTRS,FORMAT,S,jpeg,PAGE,I,0
PARAMETERS,S,'name="garage_band",sales=FLOAT(100.2),
release_date=DATE(01/02/2002)'")
```

SET_OPTIONS

Purpose

Turns tracing options off or on.

Syntax

```
EXECUTE set_options
WITH option='option_name',"value"=true|false

dmAPIGet("apply,session,NULL,SET_OPTIONS,
 OPTION,S,option_name,VALUE,B,T|F")
```

Arguments

Table 3–56. SET_OPTIONS Arguments

Argument	Datatype	Value	Description
OPTION	S	*option_name*	Identifies the tracing option you want to turn on or off. Refer to Table 3–57, page 288
VALUE	B	T (TRUE) or F (FALSE)	TRUE turns tracing on. FALSE turns tracing off.

Return Value

SET_OPTIONS returns a collection with on result object. The object has one Boolean attribute whose value indicates the success (TRUE) or failure (FALSE) of the operation.

Permissions

You must have Sysadmin or Superuser privileges to use this method.

General Notes

The tracing results for the options representing server operations are printed to the server log file. The trace_method_server option traces method server operations. The tracing results generated by setting trace_method_server are printed to the method server log file.

Table 3–57, page 288, lists the values for the OPTION argument:

Table 3–57. Trace Options for SET_OPTIONS

Value for the OPTION Argument	Meaning
ca_store_trace	Enables tracing for operations in a content-addressed storage system. Trace messages are placed in the server log file. Tracing content-addressed storage operations is only recommended when needed to troubleshoot the system.
clean	Removes the files from the server common area.
debug	Traces session shutdown, change check, launch and fork information
docbroker_trace	Traces DocBroker information
i18n_trace	Traces client session locale and codepage. An entry is logged identifying the session locale and client code page whenever a session is started. An entry is also logged if the locale or codepage is changed during the session.
last_sql_trace	Traces the SQL translation of the last DQL statement issued before access violation and exception errors. If an error occurs, the last_sql_trace option causes the server to log the last SQL statement that was issued prior to the error. This tracing option is enabled by default. It is strongly recommended that you do not turn off this option. It provides valuable information to Technical Support if it ever necessary to contact them.
lock_trace	Traces Windows locking information
net_ip_addr	Traces the IP addresses of client and server for authentication
nettrace	Turns on RPC tracing. Traces Netwise calls, connection ID, client host address, and client host name
sqltrace	Traces SQL commands sent to the underlying RDBMS for subsequent sessions
trace_authentication	Traces detailed authentication information
trace_complete_launch	Traces Unix process launch information
trace_method_server	Traces the operations of the method server.
trace_workflow_agent	Traces operations of the workflow agent. Messages are recorded in the server log file.

Related Administration Methods

LOG_OFF, page 232

LOG_ON, page 233
MODIFY_TRACE, page 247

Examples

This example turns on SQL tracing:

```
EXECUTE set_options
WITH option='sqltrace',"value"=true
```

The following example turns on logon authentication tracing:

```
dmapiGet("apply,c,NULL,SET_OPTIONS,
OPTION,S,trace_authentication,VALUE,B,T")
```

This example turns off tracing for logon authentication:

```
dmapiGet("apply,c,NULL,SET_OPTIONS,
OPTION,S,trace_authentication,VALUE,B,F")
```

This example enables tracing for a content-addressed storage system:

```
dmAPIGet("apply,c,NULL,SET_OPTIONS,OPTION,S,ca_store_trace,VALUE,B,T")
```

SET_STORAGE_STATE

Purpose

Takes a storage area offline, places an off-line storage area back online, or makes a storage area read-only.

Syntax

```
EXECUTE set_storage_state [[FOR] 'storage_area_obj_id']
[WITH store=storage_area_name{,argument=value}]
```

With EXECUTE, if you include the STORE argument, do not include the FOR clause.

```
dmAPIGet("apply,session,storage_area_obj_id,SET_STORAGE_STATE
[,STORE,S,storage_area_name][,argument,datatype,value}")
```

With Apply, specify the storage area object ID as NULL if you include the STORE argument.

Arguments

Table 3–58. SET_STORAGE_STATE Arguments

Argument	Datatype	Value	Description
OFFLINE	B	T (TRUE) or F (FALSE)	If set to TRUE, the storage area is taken off-line.
READONLY	B	T (TRUE) or F (FALSE)	If set to TRUE, the storage area is read-only.
STORE	S	*storage_area _name*	Identifies the storage area whose state you are changing. Use the name of the area's storage area object.

Return Value

SET_STORAGE_STATE returns a collection with one result object. The object has one Boolean attribute whose value indicates the success (TRUE) or failure (FALSE) of the operation.

Permissions

You must have Sysadmin or Superuser privileges to use this method.

General Notes

A storage area has three possible states:

- Online

Users can read from and write to an online storage area.

- Offline

  Users can neither read from nor write to an offline storage area.

- Readonly

  Users can only read from a readonly storage area. Users cannot write to a readonly storeage area.

To set a storage area's state to offline, issue SET_STORAGE_STATE with the OFFLINE argument set to T (TRUE); for example:

```
EXECUTE set_storage_state WITH store=filestore_33,OFFLINE=true
```

To set a storage area's state to readonly, issue SET_STORAGE_STATE with the READONLY argument set to T (TRUE); for example:

```
EXECUTE set_storage_state WITH store=filestore_33,READONLY=true
```

To change either an offline or readonly storage area back to the online state (users can read and write the storage area), issue the SET_STORAGE_STATE method without specifying a state argument; for example:

```
EXECUTE set_storage_state WITH store=filestore_33
```

Setting either OFFLINE or READONLY to FALSE directly has no effect.

To use SET_STORAGE_STATE, you must be using one server for both data and content requests. If the configuration is set up for content servers, you must issue a Connect method that bypasses the content server to use SET_STORAGE_STATE in the session.

Related Administration Methods

None

Examples

The following example moves the storage area called manfred offline:

```
EXECUTE set_storage_state
WITH store = 'manfred', offline = T
```

The next example sets the storage1_engr file store storage area offline:

```
dmAPIGet("apply,c,NULL,SET_STORAGE_STATE,OFFLINE,B,T,
STORE,S,storage1_engr")
```

SETSTYLE_FTINDEX

Purpose

Associates a topic set with an index or a style file with all indexes.

Syntax

To add a topic set:

```
EXECUTE setstyle_ftindex [[FOR] 'fulltext_index_id'
WITH [name='index_name',]verity_style='topic_trees',
server_path='server_path'[,install_loc='verity_location_name']
[,locale='verity_locale']
```

With Execute, do not include the FOR clause if you include the NAME argument.

```
dmAPIGet("apply,session,fulltext_index_id,SETSTYLE_FTINDEX,
[NAME,S,index_name,]VERITY_STYLE,S,topic_trees,
SERVER_PATH,S,server_path[,INSTALL_LOC,S,verity_location_
name][,LOCALE,S,verity_locale]")
```

With Apply, specify the fulltext index ID as NULL if you include the NAME argument.

To add a style file:

```
EXECUTE setstyle_ftindex WITH verity_style='style_type'
,SERVER_PATH='server_path'[,install_loc='verity_location_name']
[,locale='verity_locale']
```

```
dmAPIGet("apply,session,NULL,SETSTYLE_FTINDEX,
VERITY_STYLE,S,style_type,SERVER_PATH,S,server_path
[,INSTALL_LOC,S,verity_location_name][,LOCALE,S,verity_locale]")
```

Arguments

Table 3–59. SETSTYLE_FTINDEX Arguments

Argument	Datatype	Value	Description
NAME	S	*name*	Identifies the index with which to associate the specified topic set. Use the name associated with the index's fulltext index object. Optionally, you can omit the NAME argument and specify the object ID of the fulltext index object. Use this argument (or the fulltext index object ID) only when you are removing an index's topic set.

Argument	Datatype	Value	Description
VERITY _STYLE	S	topic_trees or *style_type*	Identifies the topic set or style file that you want to add to the system. To associate a topic set with a specified index, use the keyword topic_trees. To associate a style file with all indexes, specify the *style_type* using the style file's file extension. Valid file extensions are: **lex**, for a lex file **stp**, for a stop file **syd**, for a thesaurus file **uni**, for a style.uni file **zon**, for a zone file
SERVER _PATH	S	*server_path*	Specifies the location of the topic set or style file to add.
INSTALL_LOC	S	*verity_location_ name*	Name of the location object that identifies the location of the Verity installation. If included, this must be one of the values in the fulltext_install_loc attribute of the docbase config object. If not included, the default is the value of the verity_location attribute in the server config object.
LOCALE	S	*verity_locale*	Name of a Verity locale. If specified, the style file or topic tree is stored in the specfiied locale subdirectory. You must include this argument if VERITY_STYLE is set to syd.

Return Value

SETSTYLE_FTINDEX returns a collection with one query result object. The object has one Boolean attribute whose value indicates the success (TRUE) or failure (FALSE) of the operation.

Permissions

You must have Sysadmin or Superuser privileges to use this method.

General Notes

When you add a topic set, it is available to all indexes. A new style file is only applied to indexes created after the style file is added to the system. Existing indexes are not affected by a new style file unless you recreate them.

When you add a topic set, the topic set is not copied to a Verity directory. Instead, the server stores a reference to the directory specified in the SERVER_PATH argument. When you make changes to the topic set, the changes are visible to all subsequent sessions.

When you add a style file to the system, the method copies the file to %DOCUMENTUM%\fulltext\verity271\dctm_custom unless you specified a locale.
If you specified a locale in the SETSTYLE_FTINDEX arguments, the file is stored in subdirectory named for the locale. For example, if you added a stop file to the frenchx locale, the file is stored in %DOCUMENTUM%\fulltext\verity271\dctm_custom\frenchx.

If you add a thesaurus file, you must identify a locale in the SETSTYLE_FTINDEX arguments.

The method changes the name of the customized style file also. For all files except the thesaurus file, the format of the new name is dm_custom.*style_extension*, where *style_extension* is one of lex, stp, uni, or zon. For example, if you added a custom stop file, the file's new name would be dm_custom.stp.

Because the Full-Text Engine always uses a thesaurus file named vdk20.syd, when you add a customized thesaurus file, the method always names the new file vdk20.syd. The previous vdk20.syd (thesaurus) file is renamed to vdk20.syd.tdk if it was the default file supplied with the installation. If the previous thesaurus was an earlier customized thesaurus, it is renamed to vdk20.syd.custom.

Related Administration Methods

GETSTYLE_FTINDEX, page 208
RMSTYLE_FTINDEX, page 275

Examples

The following examples add a topic set to the storage1_index full-text index, which has an Italian locale. The first two examples show the use the EXECUTE statement on a Windows platform and a UNIX platform.

```
EXECUTE setstyle_ftindex
WITH name='storage1_index',
verity_style='topic_trees','
server_path='c:\adminops\newtopics',locale='italianx'
```

```
EXECUTE setstyle_ftindex
WITH name='storage1_index',
verity_style='topic_trees','
server_path='u01/adminops/newtopics',locale='italianx'
```

The next two examples show the same operation using an Apply method on Windows and UNIX.

```
dmAPIGet("apply,c,NULL,SETSTYLE_FTINDEX,
NAME,S,storage1_index,VERITY_STYLE,S,topic_trees,
SERVER_PATH,S,c:\adminops\newtopics")
```

```
dmAPIGet("apply,c,NULL,SETSTYLE_FTINDEX,
NAME,S,storage1_index,VERITY_STYLE,S,topic_trees,
SERVER_PATH,S,u01/adminops/newtopics")
```

The final examples add a new lex file to the indexes in the Docbase.

```
dmAPIGet("apply,c,NULL,SETSTYLE_FTINDEX,
VERITY_STYLE,S,lex,
SERVER_PATH,S,c:\adminops\newlex.lex")
```

```
dmAPIGet("apply,c,NULL,SETSTYLE_FTINDEX,
VERITY_STYLE,S,lex,
SERVER_PATH,S,u01/adminops/newlex.lex")
```

SHOW_SESSIONS

Purpose

Returns information about all the currently active Docbase sessions.

Syntax

```
EXECUTE show_sessions
dmAPIGet("apply,session,NULL,SHOW_SESSIONS")
```

Arguments

SHOW_SESSIONS has no arguments.

Return Value

SHOW_SESSIONS returns a collection. Each query result object in the collection represents one currently active Docbase session. The attributes of the objects contain information about the sessions. The attributes returned by SHOW_SESSIONS are the same as those for LIST_SESSIONS when LIST_SESSIONS is run to return a full set of attributes. For a description of the attributes, refer to Table 3–30, page 224.

Permissions

Anyone can use this method.

General Notes

SHOW_SESSIONS does not return information about historical (timed out) sessions. It only returns information about currently active sessions. The information for each session is identical to the information returned by LIST_SESSIONS for active sessions if that method is run with the BRIEF_INFO argument set to FALSE.

Note: You can use SHOW_SESSIONS to obtain the process ID if you need to shutdown or kill a session or server.

Related Administration Methods

LIST_SESSIONS, page 224

Examples

Refer to the syntax description for examples.

SYNC_REPLICA_RECORDS

Purpose

Synchronizes the dmi_replica_record objects with the current definition of a distributed storage area. (This method is not available through Documentum Administrator.)

Syntax

```
EXECUTE sync_replica_records [[FOR]'storage_obj_id']
[WITH argument=value {,argument=value}]
```

With Execute, do not include the FOR clause if you include the STORE argument.

```
dmAPIGet("apply,session,storage_obj_id,SYNC_REPLICA_RECORDS
{,argument,datatype,value}")
```

With Apply, specify the storage object ID as NULL if you include the STORE argument.

Arguments

Table 3–60. SYNC_REPLICA_RECORDS Arguments

Argument	Datatype	Value	Description
STORE	S	*storage_area_ name*	Identifies the distributed storage area for which you want to synchronize the records. Use the name associated with the area's dm_distributedstore object. Optionally, omit the STORE argument and specify the object ID of the dm_distributedstore object.
COMMIT_ INTERVAL	I	*integer*	Specifies the number of replica record objects to process before each commit to the Docbase. The default is 10,000.
START_ID	S	*object_id*	Identifies a replica record object that serves as the starting point for the operation. Replica record objects that have an object ID greater than the specified object ID are processed. Use this in conjunction with END_ID.

Argument	Datatype	Value	Description
END_ID	S	*object_id*	Identifies a replica record object that is the stopping point for the synchronization process. Record replica objects that have an object ID greater than this are not processed. Use this in conjunction with START_ID.
CONTINUE_ON_ ERROR	B	T (TRUE) or F (FALSE)	Indicates whether the method continues in the event of an error. The default is FALSE.
REMOVE_ NULL_ ENTRIES	B	T (TRUE) or F (FALSE)	Indicates whether to remove null component entries. The default is T.
UPDATE_ EPOCH	B	T (TRUE) or F (FALSE)	Indicates whether to bring the epoch value up to date even if the record does not need to be upgraded. The default is T.
TRACE_ LEVEL	I	*integer*	Sets the tracing level of the method. Valid trace levels are: 0, meaning no tracing 1, meaning informa- tional messages only 2, meaniing trace all updated objects 5, meaning trace all objects checked for update The default is 0.

Return Value

SYNC_REPLICA_RECORDS returns a collection with one query result object. The object has one Boolean attribute, named result, whose value indicates the success (TRUE) or failure (FALSE) of the operation.

If CONTINUE_ON_ERROR is set to TRUE, the method returns FALSE only when the method encounters an error from which it cannot recover.

Permissions

You must have Sysadmin or Superuser privileges to run this method.

General Notes

Use SYNC_REPLICA_RECORDS after you upgrade from DocPage Server 3.x to Documentum 4.*x*. The method updates the dmi_replica_record objects to synchronize them with the Documentum 4.*x* architecture by removing unneeded null component entries. As a side effect, the method also removes entries for components that are no longer part of the distributed storage area

Using SYNC_REPLICA_RECORDS is optional. Running the method may improve performance, but Content Server and the distributed features function correctly if you do not run it.

In a Documentum 4.*x* installation, you can run this method after you delete a distributed storage area component. However, it isn't required. Content Server will dynamically remove deleted component entries as the documents are referenced.

Sizing the Commit Interval

The method processes objects in chunks. That is, it processes a defined number of objects and then commits the work to the Docbase before processing more objects. The default number of objects in a chunk is 10,000. Use the COMMIT_INTERVAL argument to change the size of an operational chunk.

Batching the Operation

The START_ID and END_ID arguments allow you to process a specified number of objects in each execution of the operation. If you do not include these arguments, the method processes all the replica record objects associated with the specified storage area.

Related Administration Methods

None

Examples

These examples synchronize the replica records for the diststore1 distributed storage area:

```
EXECUTE sync_replica_records
WITH store='diststore1'
```

```
dmAPIGet("apply,c,NULL,SYNC_REPLICA_RECORDS,
STORE,S,diststore1")
```

This example specifies a commit interval of 5,000 objects:

```
EXECUTE sync_replica_records
WITH store='diststore1',commit_interval=5000
```

This example sets the tracing level to 5, to trace all objects checked for updating:

```
dmAPIGet("apply,c,NULL,SYNC_REPLICA_RECORDS,
STORE,S,diststore1,TRACE_LEVEL,I,5")
```

TRANSCODE_CONTENT

Purpose

Queues a transformation request for a content file to the Media Server.

Syntax

```
EXECUTE transcode_content FOR object_id
WITH [page=page_number,] [priority=priority_level,]
message='message',source_format='format_name',
target_format='format_name'
```

```
dmAPIGet("apply,session,object_id,TRANSCODE_CONTENT,
[PAGE,I,page_number,][PRIORITY,I,priority_level,]
MESSAGE,S,'message',TARGET_FORMAT,S,format_name,
SOURCE_FORMAT,S,format_name")
```

object_id is the object ID of the document that contains the content file.

Arguments

Table 3–61. TRANSCODE_CONTENT Arguments

Argument	Datatype	Value	Description
PAGE	I	*page_number*	Page number of the content file. If unspecified, the default is 0.
PRIORITY	I	*priority_level*	Identifies the priority of the request. A value of zero indicates no priority. Priority rises as the value rises. The Media Server processes higher priority requests before lower priority requests. If unspecified, the default is 5.
MESSAGE	S	-profile_id *object_id*	Identifies the profile that contains the definition of the transformation to be performed on the content. The profile is identified by its object ID. Refer to the General Notes for a more detailed description.

Argument	Datatype	Value	Description
TARGET_FORMAT	S	*format_name*	Identifies the format to which to transform the content. Use the name of the format as it is defined in the format's format object.
SOURCE_FORMAT	S	*format_name*	Identifies the content file's starting format. Use the name of the format as it is defined in the format's format object.

Return Value

TRANSCODE_CONTENT returns a collection with one query result object. The object has one Boolean attribute whose value indicates the success (TRUE) or failure (FALSE) of the operation.

Permissions

You must have at least Write permission on the object or Sysadmin privileges to use this method.

General Notes

TRANSCODE_CONTENT is issued by WebPublisher™ to request a transformation operation on a content file by the Media Server. The method generates an event, dm_transcode_content, that is queued to the Media Server. When the Media Server processes the event, the server performs the transformation on the content as defined in the specified profile.

The dm_queue_item that represents the event is queued to the dm_mediaserver user, who represents the Media Server. (dm_mediaserver is created when Documentum Media Services is installed.) TRANSCODE_CONTENT sets the following attributes in the queue item:

Table 3–62. Queue Item Attributes Set by TRANSCODE_CONTENT

Attribute	Set to
event	dm_transcode_content
item_id	object ID of the SysObject that triggered the request
instruction_page	page number of the content in the SysObject
content_type	starting format of the SysObject
message	value specified in the MESSAGE argument
item_type	format to which to transform the content
date_sent	date the request was made
name	dm_mediaserver

Attribute	Set to
sent_by	session user
priority	priority level identified in the REGISTER_ASSET call

The MESSAGE argument specifies the transformation operation to perform by specifing a profile. A profile is an XML document, stored in the Docbase, that defines transformation operations. The value for a MESSAGE argument has the following format:

```
'-profile_id object_id'
```

object_id is the object ID of the profile document.

For example, suppose that 090000132400c2e198 is the object ID of a document that you want to transform and that 09000013240053ae1b is the object ID of a profile containing the definition of the desired transformation. The following DQL EXECUTE statement generates a request to the Media Server to perform the transformation:

```
EXECUTE transcode_content FOR 090000132400c2e198
WITH page=0,message='-profile_id 09000013240053ae1b',
source_format='jpeg',target_format='gif'
```

Related Administration Methods

REGISTER_ASSET, page 264

Example

```
EXECUTE transcode_content FOR 090000017456ae1c
WITH page=0,priority=5,
message='-profile_id 090000018125ec2b',
source_format='jpeg',
target_format='jpeg_lres'
```

```
dmAPIGet("apply,s0,090000017456ae1c,TRANSCODE_CONTENT,
PAGE,I,0,PRIORITY,I,5,
MESSAGE,S,'-profile_id 090000018125ec2b',
SOURCE_FORMAT,S,jpeg,
TARGET_FORMAT,S,jpeg_lres")
```

UPDATE_FTINDEX

Purpose

Updates a full-text index.

Syntax

```
EXECUTE update_ftindex [[FOR] 'fulltext_index_obj_id']
[WITH argument = value {,argument = value}]
```

With Execute, do not include the FOR clause if you include the NAME argument.

```
dmAPIGet("apply,session,fulltext_index_obj_id,UPDATE_FTINDEX
{,argument,datatype,value}")
```

With Apply, specify the fulltext index object ID as NULL if you include the NAME argument.

Arguments

Table 3–63. UPDATE_FTINDEX

Argument	Datatype	Value	Description
NAME	S	*index_name*	Identifies the index to update. Use the name associated with the index's fulltext index object. Optionally, omit the NAME argument and specify the object ID of the fulltext index object.
PHASE	I	-	Controls whether the first phase, second phase, or both phases of the update operation are executed. Setting this to 1 means only the first phase of updating is executed. Setting this to 2 means only the second phase of updating is executed. If this is not included, both phases are executed.

Argument	Datatype	Value	Description
BATCH _SIZE	I	*value*	Specifies how many contents to include in each batch during the update operation. This argument applies only to the first phase of operation. Do not include this argument if PHASE is set to 2. The default is 1,000. Do not set this argument above 20,000.

Argument	Datatype	Value	Description
MAX_DOCS	I	*integer*	Defines an approximate number of documents to be indexed in this update. This argument applies only to the first phase of operation. Do not include this argument if PHASE is set to 2. MAX_DOCS and DATE_CUTOFF are mutually exclusive. If you include MAX_DOCS, you cannot include DATE_CUTOFF.
DATE _CUTOFF	S	*string*	Defines a date as the upper time boundary for choosing documents to index. Documents modified after the date specified are not included in the update. Valid date formats and values for this argument are: *mm/dd/yy*[*yy*] *hh*::*mm*::*ss* [AM|PM] *mm/dd/yy*[*yy*] *mon dd yy*[*yy*] *month dd yy*[*yy*] If you specify a year prior to 2000, you must use a four-digit specification. You cannot specify a date later than the current time. This argument applies only to the first phase of operation. Do not include this argument if PHASE is set to 2. DATE_CUTOFF and MAX_DOCS are mutually exclusive. If you include DATE_CUTOFF, you cannot include MAX_DOCS.

Return Value

UPDATE_FTINDEX returns a collection with one result object that has two attributes: result and status. The result attribute is a Boolean attribute that indicates whether the method completed successfully. The status attribute contains an integer value indicating the status of the update. Table 3–64, page 306, lists the possible combinations of values for result and status.

Table 3–64. Possible Combinations of Result and Status Attribute Values

Result	Status	Meaning
F	0	The method failed due to some external factor, such as a machine crash.
T	0	The method completed successfully but some objects were not included in the batch files or not indexed.
T	1	The method completed successfully and all objects were included in the batch files (for phase 1) and if phase 2 was executed, all were indexed.
T	2	Another fulltext operation is in progress.

Permissions

You must have Sysadmin or Superuser privileges to use this method.

General Notes

UPDATE_FTINDEX updates a full-text index. The method incorporates new content and updates indexed attribute values, if needed, for currently indexed objects. The method also marks for removal all entries associated with documents that have been deleted since the last update. You can execute UPDATE_FTINDEX on searchable indexes or on standby indexes when it is run manually. (When it is invoked by the Full Text Manager job, it can execute only on searchable indexes.)

The UPDATE_FTINDEX method does not execute if the r_update_inprogress attribute for the index is set to TRUE or if the CLEAN_FTINDEX method is currently executing.

Note: To prevent the Full Text Manager job from attempting to run an update while you are manually updating, you may want to set the job to inactive while you are executing UPDATE_FTINDEX manually. Reset the job to active after you complete the manual update.

When UPDATE_FTINDEX is started, the server logs the operation's process ID in the server log file.

The method executes in two phases. The first phase creates the batch files for indexing. The second phase goes through each batch file and indexes the content files in each batch. You can execute each phase in seperate executions of the method by including the PHASE argument on the method's command line.If needed, you can run the first phase multiple times before running the second phase. However, to update an index, both phases must be completed.

Log and Trace Files

UPDATE_FTINDEX automatically generates a status log file during the both phases of operation. The files contain fatal, error, warning, and status messages generated by Content Server. If you have executed a MODIFY_TRACE to set the tracing level to verity or all, mkvd debug messages are also included. The current files are called status. After the operation completes, the files are renamed to status.last. They are stored in the fulltext working directory, dm_ftwork_dir.

The status log files have a time stamp and old ones are removed when the LogPurge administration job is run.

Setting the trace level to verity or all using MODIFY_TRACE also generates a log file named fttrace_*docbase*.log. This file is stored in %DOCUMENTUM%\dba\log\fulltext ($DOCUMENTUM/dba/log/fulltext). Each time you execute UPDATE_FTINDEX, the method appends to this file. Consequently, this file can grow large. Be sure to manage it properly when tracing is turned on. If you stop and restart Content Server, when the server is started, the old log file is saved with a time stamp and a new file is started. Old log files are removed by the LogPurge administration tool.

The DATE_CUTOFF and MAX_DOCS Parameters

Use the DATE_CUTOFF or MAX_DOCS parameter if you have a large number of documents and want to index them in batches. For example, suppose you have 1 million documents, loaded into the Docbase over the course of 4 days, to index. You could run UPDATE_FTINDEX with DATE_CUTOFF set so that each iteration of the method indexes the documents loaded during one day. Or, you could run UPDATE_FTINDEX with MAX_DOCS set to 10,000. In that case, each time you execute UPDATE_FTINDEX, the operation picks up where it left off in the previous invocation.

If you batch load large numbers of documents into the Docbase, using DATE_CUTOFF is recommended when you run UPDATE_FTINDEX to add those documents to the index.

Using either parameter also provides a way to test a small batch of documents before indexing a larger set.

Related Administration Methods

MARK_ALL, page 237
MARK_FOR_RETRY, page 240

Examples

The following example updates the storage1_index in verbose mode, with a batch size of 50:

```
EXECUTE update_ftindex WITH name='storage1_index',
batch_size=50
```

The following example identifies the index using the object ID of the associated fulltext index object:

```
dmAPIGet("apply,c,0f0000018574231b,UPDATE_FTINDEX,
BATCH_SIZE,I,50")
```

The next two examples execute only phase 1 of the update:

```
EXECUTE update_ftindex WITH name='storage1_index',
batch_size=50,phase=1

dmAPIGet("apply,c,0f0000018574231b,UPDATE_FTINDEX,
BATCH_SIZE,I,50,PHASE,I,1")
```

UPDATE_STATISTICS

Purpose

Updates statistics for RDBMS tables in the Docbase.

Syntax

```
EXECUTE update_statistics WITH table_name='name'
[,count=integer] [,extra_data=value]

dmAPIGet("apply,session,NULL,UPDATE_STATISTICS,
TABLE_NAME,S,name[,COUNT,I,integer] [,EXTRA_DATA,S,value]
```

Arguments

Table 3–65. UPDATE_STATISTICS Arguments

Argument	Datatype	Value	Description
TABLE_NAME	S	*name*	Name of the database table whose statistics you want to update.
COUNT	I	*integer*	This argument is available only if the database is Oracle, SQL Server, or Sybase. It is not available on DB2. The argument is used differently for each database. Refer to the General Notes for details.
EXTRA_DATA	S	*value*	The argument is used differently for each database. Refer to the General Notes for details.

Return Value

UPDATE_STATISTICS returns a collection with one query result object. The object has one attribute, called result, that contains T if the method was successful or F if the method was unsuccessful.

Permissions

You must be a Superuser to execute this method.

Caution Do not use this method unless directed to do so by Technical Support.

General Notes

UPDATE_STATISTICS is used by the UpdateStatistics administration tool when the tool is run on a DB2 database. However, the method can be used on any supported database.

The COUNT Argument

This is an optional argument. It is used differently in Oracle, SQL Server, and Sybase. It is not used if the database is a DB2 database.

For Oracle, it defines the number of buckets to use when calculating the histogram statistics. The valid range for COUNT on an Oracle database is 1 to 254. The default is 75. Setting COUNT to 0 deletes all histogram statistics for the table.

For SQL Server, this argument defines what percentage of table rows to use when calculating the statistics. For example, if you set this to 50, the method uses one half (50%) of the table rows to calculate the statistics. The default is 100, meaning all rows are used to calculate the statistics.

For Sybase, this argument defines how many steps to use when calculating the statistics. The default is 20.

The EXTRA_DATA Argument

The EXTRA_DATA argument is an optional argument. Its use differs depending on the database.

For Oracle and Sybase, this identifies a specific set of table columns to be analyzed for statistics. The columns must exist in the table. The columns are specified as a comma-separated list of column names. Enclose the entire list in single quotes. For example:

```
dmAPIGet("apply,c,UPDATE_STATISTICS,TABLE_NAME,S,dm_sysobject_s,
EXTRA_DATA,'keywords,authors'
```

If you do not include EXTRA_DATA, all columns in the table are used.

Including this argument for Sybase databases can be expensive because the RDBMS must scan the table and perform a sort operation on the table.

For SQL Server and DB2, the argument identifies an index for which you want to generate statistics. The index must be an index on the table identified in TABLE_NAME. If you do not include the argument, the database table identified in TABLE_NAME and all of its indexes are analyzed.

Related Administration Methods

REORGANIZE_TABLE, page 266

Examples

This example updates statistics for the dm_sysobject_s table:

```
EXECUTE update_statistics
```

```
WITH table_name='dm_sysobject_s'

dmAPIGet("apply,S0,NULL,UPDATE_STATISTICS,
TABLE_NAME,S,dm_sysobject_s")
```

This example updates the statistics for the dm_user_s table, specifying that only the user_name, user_os_name and user_address columns be used for the analysis:

```
EXECUTE update_statistics
WITH table_name='dm_user_s',
extra_data='user_name,user_os_name,user_address'

dmAPIGet("apply,S0,NULL,UPDATE_STATISTICS,
TABLE_NAME,S,dm_user_s,
EXTRA_DATA,S,'user_name,user_os_name,user_address'")
```

WEBCACHE_PUBLISH

Purpose

Invokes the dm_webcache_publish method to publish documents to a Web site. (This administration method cannot be run using the EXECUTE statement.)

Syntax

```
dmAPIGet("apply,session,webc_config_obj_id,WEBCACHE_PUBLISH
[,ARGUMENTS,S,argument_list]")
```

Arguments

Table 3–66. WEBCACHE_PUBLISH Arguments

Argument	Datatype	Value	Description
ARGUMENTS	S	*argument _list*	Identifies the arguments that you want to pass to the dm_webcache_publish method. The valid arguments are: • -source_object_id *object_id*Identifies the object or source folder to be refreshed. If this identifies an object, the object must reside under the source folder identified in the webc config object specified in the command line. If this identifies a folder, the folder must be the source folder or reside under the source folder. If you specify a folder, all objects in and underneath the folder are refreshed. If you don't include this argument, the entire source data set is refreshed.This option is ignored if -full_refresh is set to TRUE. • -full_refresh TRUE|FALSE-When this is set to TRUE, the content and attribute data at the target WebCache are deleted and republished.You must have Superuser privileges to set this option to TRUE. The default is FALSE. • -force_refresh TRUE|FALSEWhen this is set to TRUE, documents are refreshed even if their modification dates do not indicate a need for a refresh.The default for this option is FALSE.

Argument	Datatype	Value	Description
ARGUMENTS	S	*argument _list*	• -method_trace_level *trace_level*Turns on tracing for the method at the indicated level. Valid trace levels correspond to the levels available for the Trace method. • -recreate_property_ schemaPerforms a full refresh and destroys and recreates the propdb tables. You must have Superuser privileges to set this option to TRUE. The default is FALSE. • -resync_state_tableDeletes state information in the probdb_*tablename*_m table and recreates the information based on the current configuration object. The default for this flag is FALSE. • -store_logCreates a log file object in the Docbase for each publish operation. By default, this flag is TRUE for all full refresh and incrementals publishing operations and FALSE when the method is issued for a single item.

Return Value

The WEBCACHE_PUBLISH administration method returns a collection identifier for a collection with one attribute. The attribute, method_return_val, contains 0 if the operation was successful.

Permissions

You must have Sysadmin or Superuser privileges to use this administration method.

Generating a Log File

When the operations generate a log file, the document is stored in the Docbase in /System/Sysadmin/Reports/Webcache. By default, the method does not generate a log file if you are publishing only a single item. To force it to generate a log file for single-item operations you can include the -store_log argument, set to TRUE

Example

```
dmAPIGet("apply,s0,080015b38001a43c,WEBCACHE_PUBLISH,
ARGUMENTS,S,-source_object 090015b38007bf19")
```

Appendix A

Using DQL Hints

This appendix describes how DQL hints are implemented and how to use them in your queries.

General Guideline for All

Database hints affect how a query is executed. When deciding whether to use a hint, it is recommended that you execute the query with and without the hint. Compare the generated SQL queries and compare the response time and the resource use to determine whether the hint is helpful.

SQL_DEF_RESULT_SET N

The SQL_DEF_RESULT_SET N is most useful on a SQL Server database. On a SQL Server database, including SQL_DEF_RESULT_SET in a query causes the RDBMS to use default result sets to return the query results rather than a cursor. Using default result sets is the way that Microsoft recommends accessing a SQL Server database.

On other databases, it only controls how many rows are returned, and is identical to the RETURN_TOP hint.

On SQL Server, using default result sets allows the RDBMS server to perform fewer logical read operations. Table A–1, page A–318 shows some test examples:

Table A–1. Comparison of Logical Reads with and without Default Result Sets

Query	Number of Returned Rows	Number of Logical Reads without DRS	Number of Logical Reads with DRS
SELECT object_name FROM dm_document WHERE r_object_id= '0900014c8000210b'	1	36	5
SELECT object_name FROM dm_document WHERE r_creator_name='user2'	100	1600	700
SELECT object_name FROM dm_document WHERE r_creator_name='user1'	2000	22500	500

However, the reduction in I/O may be offset by higher CPU costs. Additionally, because using default result sets requires Content Server to buffer the results of a query in memory, higher memory use is a result. (When using default result sets, two queries cannot be open simultaneously. If a query is issued, before another can be issued from the same session, all results from the first must be processed. To work around this limitation, Content Server internally buffers the results of queries that are executed using default result sets.)

To determine whether using SQL_DEF_RESULT_SETS is useful for a query, run the query with and without the hint and trace the results using the SQL Server Profiler.

Setting N to 0 causes all results to be returned. Setting N to an integer value limits the number of rows returned to that value.

Using SQL_DEF_RESULT_SETS is recommended if:

- You are running on a SQL Server database.
- The query returns limited result sets.
- The query is a commonly executed query.
- Tests show that using the hint greatly improves query performance.

FORCE_ORDER

The FORCE_ORDER hint controls the order in which the tables referenced in the query's FROM clause are joined. The tables may be RDBMS tables or object type tables.

Oracle and SQL Server implement this hint at the statement level. For example, suppose you issue the following DQL:

```
SELECT object_name
FROM DM_SYSOBJECT ENABLE(FORCE_ORDER)
```

The generated SQL for Oracle is:

```
select /*+ ORDERED */ object_name from dm_sysobject_s;
```

The generated SQL for SQL Server is:

```
select object_name from dm_sysobject_s
OPTIONS (FORCE_ORDER)
```

Sybase implements the hint at the session level. If you include the hint, Content Server uses the ct_options function to set the CS_OPT_FORCEPLAN variable to true and unsets it after the query executes.

Using FORCE_ORDER may not ensure that you obtain a particular join order. When you examine the SQL query generated by a DQL query, you may find there are more tables in the FROM clause than are present in the DQL query. This is because Content Server often uses views to generate the SQL for many DQL queries and may also add ACL tables to the queries for security checks.

If you are considering using this hint, run the query without the hint and obtain the generated SQL statement and the execution plan. If the join order in the generated query appears incorrect and you believe that joining the tables in the order they appear in the DQL query will result in better performance, run the query with FORCE_ORDER. Compare the results to the query execution results without the hint.

If you use this hint, it is recommended that you retest the query with and without the hint occasionally to ensure that the hint is still useful. Database changes can make the plan chosen by the hint incorrect.

RETURN_TOP N

The RETURN_TOP N hint limits the number of rows returned by a query.

Database-specific Implementations

The following sections describe how RETURN_TOP is handled for individual databases.

SQL Server

On SQL Server, using this hint results in fewer database rows being touched by a query. The internal behavior of query on SQL Server is:

1. The query executes on the database (touching whatever tables are required) and generates a list of keys or lookup IDs inside the tempdb.
2. Each time a row is fetched from the cursor, the lookup ID accesses the actual table to return the full result set with all columns.

When you include the RETURN_TOP hint, the second step is executed only as many times as the hint directs. For example, if the hint is RETURN_TOP 20, then only 20 rows are returned and the table is accessed only 20 times.

Including RETURN_TOP adds the SQL Server TOP hint to the generated SQL statement. For example, suppose you issue the following DQL statement:

```
SELECT user_name FROM dm_user ENABLE (RETURN_TOP 3)
```

The generated SQL for SQL Server is:

```
select TOP 3 user_name from dm_user_s
```

DB2

On a DB2 database, including RETURN_TOP adds the FETCH FIRST N ROWS ONLY hint to the generated SQL statement. For example, the following DQL query

```
SELECT user_name FROM dm_user ENABLE (RETURN_TOP 3)
```

is translated to the following SQL statement

```
select user_name from dm_user_s
FETCH FIRST 3 ROWS ONLY
```

It is recommended that you use RETURN_TOP in conjunction with OPTIMIZE_TOP. The previous example becomes:

```
SELECT user_name FROM dm_user
ENABLE (RETURN_TOP 3, OPTIMIZE_TOP 3)
```

The statement generates the following SQL:

```
select user_name from dm_user_s
FETCH FIRST 3 ROWS ONLY OPTIMIZE FOR 3
```

Oracle and Sybase

If you include RETURN_TOP in a query running against Oracle or Sybase, the returned results are not limited at the database level, but by Content Server itself. Consequently, using RETURN_TOP on Oracle or Sybase results in no database performance gains, but may reduce network traffic.

Affects of a SEARCH Clause

If the DQL query includes a SEARCH clause, to limit results to documents that are indexed, the implementation of RETURN_TOP depends on whether the searched documents are public or not.

If all the searched documents are public, the returned results are limited at the Verity level. If the searched documents are not all public, then the limits are imposed when Content Server performs the security checking on the returned results.

Recommended Use

Use RETURN_TOP when:

- Only a particular number of rows is needed from the database.
- The application accepts ad hoc queries from users and you want to limit the potential damage an unbounded query might cause.

Using RETURN_TOP if the query sorts the results or includes the DISTINCT keyword reduces the efficiency of the hint.

OPTIMIZE_TOP N

The OPTIMIZE_TOP N hint directs the database server to return the first N rows returned by a query quickly. The remaining rows are returned at the normal speed.

On SQL Server and DB2, you can include an integer value (as N) to define how many rows you want returned quickly. On Oracle, the number is ignored. The OPTIMIZE_TOP hint is not available on Sybase.

For example, suppose you issue the following DQL query:

```
SELECT r_object_id FROM dm_sysobject
ENABLE (OPTIMIZE_TOP 4)
```

On SQL Server, the generated SQL statement is:

```
select r_object_id from dm_sysobject_s
OPTION (FAST 4)
```

On DB2, the generated SQL statement is:

```
select r_object_id from dm_sysobject_s
OPTIMIZE FOR 4 ROWS
```

On Oracle, the generated SQL statement is:

```
select /*+ OPTIMIZE_TOP */ r_object_id
from dm_sysobject_s
```

OPTIMIZE_TOP is recommended for use:

- When you want to return all the results but want the first few rows more quickly
- With the RETURN_TOP hint, to optimize the return of specified rows
- When the execution plan chosen for query is a bad plan and there is no obvious solution

Using OPTIMIZE_TOP if the query sorts the results or includes the DISTINCT keyword reduces the efficiency of the hint.

FETCH_ALL_RESULTS N

The FETCH_ALL_RESULTS N hint fetches all the results from the database immediately and closes the cursor. The hint does not affect the execution plan, but may free up database resources more quickly.

To fetch all the results, set N to 0.

On SQL Server, it is recommended that you use SQL_DEF_RESULT_SETS instead of the FETCH_ALL_RESULTS hint. SQL_DEF_RESULTS_SETS provides the same benefits and is the recommended way to access SQL Server databases.

Use FETCH_ALL_RESULTS if you want to reduce the resources used by the database server by quickly closing cursors. On SQL Server, try FETCH_ALL_RESULTS if using SQL_DEF_RESULT_SETS did not improve query performance.

IN and EXISTS

IN and EXISTS are mutually exclusive hints that you can use in a WHERE clause, in a repeating attribute predicate that contains a subquery. The syntax is:

```
WHERE ANY [IN|EXISTS] attribute_name (subquery)
```

For example:

```
SELECT "r_object_id" FROM "dm_document"
WHERE ANY IN "authors" IN
(SELECT "user_name" FROM "dm_user")
```

If you do not include either IN or EXISTS explicitly, when Content Server translates the query, it includes one or the other automatically. Which hint Content Server chooses to include is determined by the indexes present in the Docbase, the attributes referenced in the query, and other factors.

The queries generated by each option are different and perform differently when executed. For example, here is the generated SQL statement for the query in the previous example:

```
select all dm_sysobject.r_object_id
from dm_sysobject_sp dm_sysobject
where (dm_sysobject.r_object_id in
  (select r_object_id from dm_sysobject_r
   where authors in
    (select all dm_user.user_name
     from dm_user_sp dm_user)))
and (dm_sysobject.i_has_folder=1 and
      dm_sysobject.i_is_deleted=0)
```

If the DQL query used the EXISTS hint, the generated SQL statement would look like this:

```
select all dm_sysobject.r_object_id
from dm_sysobject_sp dm_sysobject
where (exists (select r_object_id
  from dm_sysobject_r
  where dm_sysobject.r_object_id = r_object_id
     and authors in
   (select all dm_user.user_name
     from dm_user_sp dm_user )))
and (dm_sysobject.i_has_folder=1 and
      dm_sysobject.i_is_deleted=0)
```

If you feel that a query that references a repeating attribute predicate that uses a subquery is performing badly, run the query and examine the generated SQL statement to determine which hint Content Server is adding to the generated SQL and note the performance time. Then, rewrite the DQL query to include the hint that the server isn’t choosing. For example, if the server is choosing to add the EXISTS hint to the generated SQL statement, rewrite the DQL query to include the IN hint. Then, rerun the query to determine if the performance improves.

Including Multiple Hints Limiting Rows Returned

If you include more than one hint that limits the rows returned by a query, the number of rows returned is the least number of rows defined in an included hint. For example, suppose you issue the following DQL statement:

```
SELECT object_name FROM dm_document ENABLE
(FETCH_ALL_RESULTS 10, RETURN_TOP 5)
```

Content Server returns 5 rows for the query because the RETURN_TOP hint is the most constraining hint in the list.

On SQL Server, if the list includes SQL_DEF_RESULT_SET, the query is always executed using default result sets regardless of where the hint falls in the list of hints. For example, suppose you issue the following statement:

```
SELECT object_name FROM dm_document ENABLE
(SQL_DEF_RESULT_SET 10, RETURN_TOP 5)
```

The query executes using default result sets but returns only 5 rows.

Passthrough Hints

Passthrough hints are hints that are passed to the RDBMS server. They are not handled by Content Server.

SQL Server and Sybase have two kinds of hints: those that apply to individual tables and those that apply globally, to the entire statement. To accommodate this, you can include passthrough hints in either a SELECT statement's source list or at the end of the statement. The hints you include in the source list must be table-specific hints. The hints you include at the end of the statement must be global hints. For example, the following statement includes passthrough hints for Sybase at the table level and the statement level:

```
SELECT "r_object_id" FROM "dm_document"
WITH (SYBASE('NOHOLDLOCK'))
WHERE "object_name"='test' ENABLE (FORCE_PLAN)
```

For DB2 and Oracle, include passthrough hints only at the end of the SELECT statement.

Syntax

To include a passthrough hint, you must identify the database for which the hint is valid. To identify the target database, keywords precede the hints. The valid keywords are: ORACLE, SQL_SERVER, SYBASE, and DB2. For example, the following statement includes passthrough hints for SQL Server:

```
SELECT "r_object_id" FROM "dm_document"
WHERE "object_name" ='test'
ENABLE SQL_SERVER('ROBUST PLAN','FAST 4',
 'ROBUST PLAN')
```

For portability, you can include passthrough hints for multiple databases in one statement. The entire list of hints must be enclosed in parentheses. The syntax is:

```
(database_hint_list {,database_hint_list})
```

where *database_hint_list* is:

```
db_keyword('hint'{,'hint})
```

db_keyword is one of the valid keywords identifying a database. *hint* is any hint valid on the specified database.

For example:

```
SELECT object_name FROM dm_document doc dm_user u
WHERE doc.r_creator_name = u.user_name
ENABLE (ORACLE('RULE','PARALLEL'),
SYBASE('AT ISOLATION READ UNCOMMITTED'),
SQL_SERVER('LOOP JOIN','FAST 1')
```

Error Handling and Debugging

It is possible to execute a DQL statement that the server considers correct DQL but is incorrect at the RDBMS level if you incorrectly specify a passthrough hint. If this occurs, any errors returned are returned by the RDBMS server, not Content Server. Not all databases return errors on database hints. For example, Oracle does not return an error if a database hint is incorrect—it simply ignores the hint. For the other databases, the error messages may be difficult to decipher.

To debug problems with queries containing database hints, you can turn on SQL tracing. By default, Content Server runs with the last SQL trace option turned on. You can turn on full SQL tracing using a Trace method or the SET_OPTIONS administrative function. The SET_OPTIONS administration method is described in SET_OPTIONS, page 287.

Appendix B

DQL Quick Reference

This appendix contains only the formal syntax descriptions of the DQL statements and the DQL reserved words. For a full description of each statement, refer to Chapter 2, DQL Statements.

The DQL Statements

This section contains the formal syntax for each of the DQL statements.

Abort

```
ABORT [TRAN[SACTION]]
```

Alter Group

```
ALTER GROUP group_name ADD members

ALTER GROUP group_name DROP members

ALTER GROUP group_name SET ADDRESS email_address
```

Alter Type

```
ALTER TYPE type_name
[FOR POLICY policy_id STATE state_name]
type_modifier_list [PUBLISH]

ALTER TYPE type_name
[FOR POLICY policy_id STATE state_name]
MODIFY (attribute_modifier_clause)[PUBLISH]

ALTER TYPE type_name
ADD attribute_def {,attribute_def}[PUBLISH]

ALTER TYPE type_name
DROP attribute_name {,attribute_name}[PUBLISH]

ALTER TYPE type_name ADD_FTINDEX [ON] attribute_name
{,attribute_name}
```

```
ALTER TYPE type_name DROP_FTINDEX [ON] attribute_name
 {,attribute_name}
```

Begin Tran

```
BEGIN TRAN[SACTION]
```

Change...Object

```
CHANGE current_type OBJECT[S] TO new_type[update_list]
[IN ASSEMBLY document_id [VERSION version_label] [DESCEND]]
[SEARCH fulltext search condition]
[WHERE qualification]
```

Commit

```
COMMIT [TRAN[SACTION]]
```

Create Group

```
CREATE GROUP group_name [WITH] [ADDRESS mail_address]
[MEMBERS member_list]
```

Create...Object

```
CREATE type_name OBJECT update_list
[,SETFILE filepath WITH CONTENT_FORMAT=format_name]
{,SETFILE filepath WITH PAGE_NO=page_number}
```

Create Type

```
CREATE TYPE type_name [(attribute_def {,attribute_def})]
[WITH] SUPERTYPE parent_type
[type_modifier_list] [PUBLISH]
```

Delete

```
DELETE FROM table_name WHERE qualification
```

Delete...Object

```
DELETE type_name [(ALL)][correlation_variable] OBJECT[S]
[IN ASSEMBLY object_id [VERSION version_label]
[NODE component_id][DESCEND]]
[SEARCH fulltext search condition]
[WHERE qualification]
```

Drop Group

```
DROP GROUP group_name
```

Drop Type

```
DROP TYPE type_name
```

Grant

```
GRANT privilege {,privilege} TO users
```

Insert

```
INSERT INTO table_name [(column_name {,column_name})]
VALUES (value {,value}) | dql_subselect
```

Register

```
REGISTER TABLE [owner_name.]table_name (column_def {,column_def})
[[WITH] KEY {column_list)]
[SYNONYM [FOR] 'table_identification']
```

Revoke

```
REVOKE privilege {,privilege} FROM users
```

Select

```
SELECT [FOR base_permit_level][ALL|DISTINCT] value [AS
name] {,value [AS name]}
FROM [PUBLIC] source_list
```

```
[IN DOCUMENT clause | IN ASSEMBLY clause]
[SEARCH [FIRST|LAST]fulltext_search_condition
[IN FTINDEX index_name{,index_name}]
[WITH|WITHOUT FT_OPTIMIZER]]
[WHERE qualification]
[GROUP BY value_list]
[HAVING qualification]
[UNION dql_subselect]
[ORDER BY value_list]
[ENABLE hint_list]
```

where the IN DOCUMENT *clause* is:

```
IN DOCUMENT object_id [VERSION version_label]
[DESCEND][USING ASSEMBLIES]
[WITH binding condition]
[NODESORT BY attribute {,attribute} [ASC|DESC]]
```

and the IN ASSEMBLY *clause* is:

```
IN ASSEMBLY object_id [VERSION version_label]
[NODE component_id][DESCEND]
```

Unregister

```
UNREGISTER [TABLE] [owner_name.]table_name
```

Update

```
UPDATE table_name SET column_assignments
WHERE qualification
```

Update...Object

```
UPDATE type_name [(ALL)][correlation_variable]OBJECT[S] update_list
[,SETFILE filepath WITH CONTENT_FORMAT=format_name]
{,SETFILE filepath WITH PAGE_NO=page_number}
[IN ASSEMBLY document_id [VERSION version_label]
[NODE component_id][DESCEND]]
[SEARCH fulltext search condition]
[WHERE qualification]
```

DQL Reserved Words

If you using DQL reserved words as object or attribute names, enclose name in double quotes when using in a DQL query.

Table B–1. DQL Reserved Words

ABORT	BOOL	CONTENT_ID	DOCBASIC
ACL	BOOLEAN	COUNT	DOCUMENT
ADD	BROWSE	CREATE	DOUBLE
ADD_FTINDEX	BUSINESS	CURRENT	DROP
ADDRESS	BY	DATE	DROP_FTINDEX
ALL	CABINET	DATEADD	ELSE
ALLOW	CACHING	DATEDIFF	ELSEIF
ALTER	CHANGE	DATEFLOOR	ENABLE
AND	CHARACTER	DATETOSTRING	ENFORCE
ANY	CHARACTERS	DAY	ESCAPE
APPEND	CHAR	DEFAULT	ESTIMATE
APPLICATION	CHECK	DELETE	EXEC
AS	COMMENT	DELETED	EXECUTE
ASC	COMMIT	DEPENDENCY	EXISTS
ASSEMBLIES	COMPLETE	DEPTH	FALSE
ASSEMBLY	COMPONENTS	DESC	FIRST
ASSISTANCE	COMPOSITE	DESCEND	FLOAT
ATTR	COMPUTED	DISABLE	FOLDER
AVG	CONTAIN_ID	DISPLAY	FTINDEX
BEGIN	CONTAINS	DISTINCT	FUNCTION
BETWEEN	CONTENT_FORMAT	DM_SESSION_DD _LOCALE	GRANT
GROUP	LINK	OF	REPORT
FOR	LOWER	ON	REVOKE
FOREIGN	MAPPING	ONLY	SCORE
FROM	MAX	OR	SEARCH
FT_OPTIMIZER	MCONTENTID	ORDER	SELECT
HAVING	MEMBERS	OWNER	SEPARATOR
HITS	MFILE_URL	PAGE_NO	SERVER
ID	MHITS	PARENT	SET
IF	MIN	PATH	SETFILE
IN	MODIFY	PERMIT	SMALLINT
INSERT	MONTH	POLICY	SOME
INT	MOVE	POSITION	STATE
INTEGER	MSCORE	PRIMARY	STORAGE

INTERNAL	NODE	PRIVATE	STRING
INTO	NODESORT	PRIVILEGES	SUBSTR
IS	NONE	PUBLIC	SUBSTRING
ISCURRENT	NOT	QRY	SUM
ISPUBLIC	NOTE	RDBMS	SUMMARY
ISREPLICA	NOW	READ	SUPERTYPE
KEY	NULL	REFERENCES	SUPERUSER
LANGUAGE	NULLDATE	REGISTER	SYNONYM
LAST	NULLINT	RELATE	SYSADMIN
LATEST	NULLSTRING	REMOVE	SYSOBJ_ID
LIST	OBJECT	REPEATING	SYSTEM
LIKE	OBJECTS	REPLACEIF	TABLE
TAG	TRUE	USER	WHERE
TEXT	TRUNCATE	USING	WITH
TIME	TYPE	VALUE	WITHIN
TO	UNION	VALUES	WITHOUT
TODAY	UNIQUE	VERSION	WORLD
TOMORROW	UNLINK	VERITY	WRITE
TOPIC	UNREGISTER	VIOLATION	YEAR
TRAN	UPDATE	WEEK	YESTERDAY
TRANSACTION	UPPER		

Appendix C

Document Query Language Examples

This appendix contains examples of Document Query Language (DQL) SELECT statements. The examples begin with basic SELECT statements that retrieve information about objects using the standard SELECT clauses, such as the WHERE clause and GROUP BY clause. Then the examples show SELECT statements that make use of the DQL extensions to the statement, such as the ability to use folder and virtual document containment information or registered tables to retrieve object information.

This appendix does not attempt to describe the syntax of the DQL SELECT statement in detail. For a complete description of SELECT, refer to Select, page 109.

Basic Examples

This section presents basic SELECT statements. These examples demonstrate the use of standard clauses such as the WHERE, GROUP BY, and HAVING clauses, as well as how to use aggregrate functions in a query. Some of these examples also demonstrate how to query repeating attributes.

The Simplest Format

The basic SELECT statement has the format:

```
SELECT target_list FROM type_name
```

The *target_list* identifies the value or values that you want to retrieve and *type_name* identifies the types or registered tables from which you want to retrieve the requested information.

The following example returns the user names of all the users in the Docbase:

```
SELECT "user_name" FROM "dm_user"
```

This next example returns the names of the users and their user privileges within a Docbase:

```
SELECT "user_name","user_privileges" FROM "dm_user"
```

Using the WHERE Clause

The WHERE clause restricts the information retrieved by placing a qualification on the query. Only the information that meets the criteria specified in the qualification is returned. The qualification can be simple or complex. It can contain any of the valid predicates or logical operators, such as AND, OR, or NOT.

The following example returns the user names of all users that belong to a specified group:

```
SELECT "user_name" FROM "dm_user"
WHERE "user_group_name"='engr'
```

The next example retrieves all the types that do not have SysObject as their direct supertype and have more than 15 attributes:

```
SELECT "name" FROM "dm_type"
WHERE "super_name" != 'dm_sysobject'
AND "attr_count" > 15
```

Searching Repeating Attributes in a WHERE Clause

To search a repeating attribute for a specific value, you use the ANY predicate in the WHERE clause qualification. When you use this predicate, if any of the values in the specified repeating attribute meets the search criteria, the server returns the object.

The following example searches the authors attribute to return any document that has Jim as one of its authors:

```
SELECT "object_name" FROM "dm_document"
WHERE ANY "authors" = 'jim'
```

This next example searches the authors attribute and returns any document that has either Jim or Kendall as an author:

```
SELECT "object_name" FROM "dm_document"
WHERE ANY "authors" IN ('jim', 'kendall')
```

The following example searches the keywords attribute and returns any document that has a key word that begins with hap:

```
SELECT "object_name" FROM "dm_document"
WHERE ANY "keywords" LIKE 'hap%'
```

Using Aggregate Functions

DQL recognizes several aggregate functions. Aggregate functions are functions that operate on a set and return one value. For example, the count function is an aggregate function. It counts some number of objects and returns the total. (For a full description of all the aggregate functions that you can use in DQL queries, refer to Aggregate Functions, page 21.)

This example counts the number of documents owned by the user named john:

```
SELECT COUNT(*) FROM "dm_document"
```

```
WHERE "owner_name"='john'
```

The following example returns the dates of the oldest and newest documents in the Docbase:

```
SELECT MIN(DISTINCT "r_creation_date"),MAX(DISTINCT "r_creation_date")
FROM "dm_document"
```

This final example returns the average number of attributes in Documentum types:

```
SELECT AVG(DISTINCT "attr_count") FROM "dm_type"
```

Using the GROUP BY Clause

The GROUP BY clause provides a way to sort the returned objects on some attribute and return an aggregate value for each sorted subgroup. The basic format of the SELECT statement when you want to use the GROUP BY clause is:

```
SELECT attribute_name,aggregate_function FROM type_name
GROUP BY attribute_name
```

The attribute named in the target list must be the same as the attribute named in the GROUP BY clause.

The following example retrieves the names of all document owners and for each, provides a count of the number of documents that person owns:

```
SELECT "owner_name",count(*) FROM "dm_document"
GROUP BY "owner_name"
```

Using the HAVING Clause

The HAVING clause is used in conjunction with the GROUP BY clause. It restricts which groups are returned.

The following example returns the owner names and a count of their documents for those owners who have more than 10 documents:

```
SELECT "owner_name",count(*) FROM "dm_document"
GROUP BY "owner_name"
HAVING COUNT(*) > 10
```

The ORDER BY Clause

The ORDER BY clause lets you sort the values returned by the query. The clause has the format:

```
ORDER BY num [ASC|DESC] {,num [ASC|DESC] }
```

or

```
ORDER BY attribute [ASC|DESC] {,attribute [ASC|DESC]}
```

The *num* identifies one of the selected values by its position in the selected values list. The *attribute* identifies one of the selected values by its name. ASC (ascending) and

DESC (descending) define the order of the sort and must be specified for each element specified in the ORDER BY clause.

The following example returns the owner's name, the object's title, and its subject for all SysObjects. The results are sorted by the owner's name and, within each owner's name group, by title:

```
SELECT "owner_name","title","subject" FROM "dm_sysobject"
ORDER BY "owner_name", "title"
```

This next example returns the owner names and a count of their documents for those owners who have more than 10 documents. The results are ordered by the count, in descending order.

```
SELECT "owner_name",count(*) FROM "dm_document"
GROUP BY "owner_name" HAVING COUNT(*) < 10
ORDER BY 2 DESC
```

Using the Asterisk (*) in Queries

DQL lets you use the asterisk in the target list if the FROM clause specifies a type name or a registered table (it cannot specify both).

If you specify a type, for each object returned by the query, the server returns the object's ID and all of its single-valued attributes whose names do not have the i_, r_, or a_ prefix. In addition, if the object is a SysObject or a SysObject subtype, the server also returns its object type (r_object_type), creation date (r_creation_date), modification date (r_modify_date), and content type (a_content_type). Specifying an asterisk does not return any of an object's repeating attributes.

If you specify a registered table, the query returns all the columns in the registered table.

The following example retrieves the attributes described above for all documents owned by the current user:

```
SELECT * FROM dm_document
WHERE "owner_name"=USER
```

This next example retrieves the attributes described above plus the repeating attribute authors for all documents owned by the current user:

```
SELECT *, "authors" FROM dm_document
WHERE "owner_name"=USER
```

Searching Cabinets And Folders

In Documentum, objects of all SysObject subtypes except cabinets are stored in cabinets, and sometimes, in folders within those cabinets. It may, at times, be advantageous to restrict the search for an object or objects to a particular cabinet or folder. Or, you may want to determine what a particular cabinet or folder contains. To make these operations possible, DQL provides the CABINET and FOLDER predicates for use with the WHERE clause. The CABINET predicate lets you specify a particular cabinet to search. The FOLDER predicate lets you specify a particular folder or cabinet to search. (You can use the FOLDER predicate for cabinets as well as folders because cabinets are subtypes of folders.)

Use the cabinet or folder's folder path or its object ID to identify it in the query. A folder path has the format:

/*cabinet_name*{/*folder_name*}

To use the object ID, you must use the ID function and the object ID. The following examples illustrate the use of both folder paths and object IDs with the CABINET and FOLDER predicates.

This example returns all the folders contained in Research folder, which is identified by its object ID:

```
SELECT "object_name" FROM "dm_folder"
WHERE FOLDER (ID('0c00048400001599'))
```

The next example returns all the objects in the Marketing cabinet sorted by their type and, within each type, ordered alphabetically. The Marketing cabinet is identified by its folder path.

```
SELECT "object_name","r_object_type" FROM "dm_sysobject"
WHERE CABINET ('/Marketing')
ORDER BY "r_object_type","object_name"
```

This final example returns all the objects in the Correspondence folder in the Marketing cabinet. Again, the objects are sorted by their type and ordered alphabetically within each type group. A folder path is used to identify the correct folder.

```
SELECT "object_name","r_object_type" FROM "dm_sysobject"
WHERE FOLDER ('/Marketing/Correspondence')
ORDER BY "r_object_type","object_name"
```

The CABINET and FOLDER predicates have an optional keyword, DESCEND, that directs the server to return not only those objects directly contained within a cabinet or folder but to also return the contents of any folders contained in that folder or cabinet, and so forth.

The following example returns the name, object ID, and type of all objects in the Clippings folder, including the contents of any folders that are contained by the Clippings folder. The Clippings folder resides in the Marketing cabinet.

```
SELECT "object_name","r_object_id","r_object_type"
FROM "dm_sysobject"
WHERE FOLDER ('/Marketing/Clippings', DESCEND)
```

Querying Registered Tables

A registered table is a table from your underlying RDBMS that has been registered with the Docbase. This registration allows you to specify the table in a DQL query, either by itself or in conjunction with a type. The following examples illustrate each possibility. (For a full discussion of the rules concerning the use of registered tables in SELECT statements, refer to Chapter 2, DQL Statements.)

The first example returns all the name of all products and their owners from the registered table named ownership:

```
SELECT "product","manager" FROM "ownership"
```

This second example joins the registered table called ownership to the user-defined type called product_info to return the names of all the objects in Collateral folder, along with the names of the managers who own the products described by the objects:

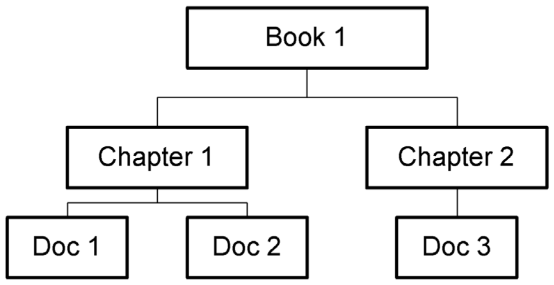
```
SELECT p."object_name", o."manager"
FROM "product_info" p, "ownership" o
WHERE p."product" = o."product" AND
FOLDER ('/Marketing/Collateral', DESCEND)
```

Querying Virtual Documents

DQL provides a clause that facilitates querying virtual documents. The clause is the IN DOCUMENT clause. The IN DOCUMENT clause lets you search a particular virtual document.

The examples in this section are based on the virtual document shown in Figure C–1, page C–336:

Figure C–1. Virtual Document Model



Determining the Components

The following example returns only the directly contained components of Book1:

```
SELECT "r_object_id" FROM "dm_sysobject"
IN DOCUMENT ID('Book1_object_id')
```

This query returns Book1, Chapter 1, and Chapter 2. Book1 is included because a virtual document is always considered to be a component of itself.

To return all the components of a virtual document, including those that are contained in its components, you use the keyword DESCEND. For example:

```
SELECT "r_object_id" FROM "dm_sysobject"
IN DOCUMENT ID('Book1_object_id') DESCEND
```

The above example returns Book1, Chapter 1, Doc 1, Doc2, Chapter 2, and Doc 3, in that order.

Note: The components of a virtual document are returned in a pre-determined order that you cannot override with an ORDER BY clause.

You can limit the search to a particular object type by specifying the type in the FROM clause. For example, the following statement returns only the documents that make up Book1 (Doc 1, Doc 2, and Doc 3):

```
SELECT "r_object_id" FROM "dm_document"
IN DOCUMENT ID('Book1_object_id')
```

Index