OpenText™ Documentum™ Content Management

Server DQL Reference Guide

Build a content or workgroup management application that uses Documentum Query Language.

EDCCS250400-DRD-EN-01

OpenText™ Documentum™ Content Management Server DQL Reference Guide
EDCCS250400-DRD-EN-01
Rev.: 2025-Oct-17

This documentation has been created for OpenText™ Documentum™ Content Management CE 25.4.
It is also valid for subsequent software releases unless OpenText has made newer documentation available with the product, on an OpenText website, or by any other means.

Open Text Corporation

275 Frank Tompa Drive, Waterloo, Ontario, Canada, N2L 0A1

Tel: +1-519-888-7111
Toll Free Canada/USA: 1-800-499-6544 International: +800-4996-5440
Fax: +1-519-888-0677
Support: https://support.opentext.com
For more information, visit https://www.opentext.com

Table of Contents

Chapter 1

Documentum Query Language language elements

This is the reference guide for OpenText™ Documentum™ Content Management's Documentum Query Language (DQL), supported by OpenText Documentum Content Management (CM) Server. It is a companion to *OpenText Documentum Content Management - Server Fundamentals Guide (EDCCS250400-GGD)*, *OpenText Documentum Content Management - Server System Object Reference Guide (EDCCS250400-ORD)*, *OpenText Documentum Content Management - Administrator User Guide (EDCAC250400-UGD)*, and *OpenText Documentum Content Management - Server and Server Extensions Installation Guide (EDCSY250400-IGD)*. This guide is written for application developers and system administrators and any others who want to build a content or workgroup 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.

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

- "Literals" on page 8, which describes the literal formats for the OpenText Documentum Content Management (CM) datatypes
- "Special keywords" on page 14, which describes the special keywords that you can use in DQL queries
- "Functions" on page 15, which describes the functions that you can use in DQL queries
- "Predicates" on page 29, which describes the predicates that you can use in expressions in queries
- "Concat operator" on page 39, which describes the concat operator supported by DQL
- "Logical operators" on page 39, which describes the logical operators supported by DQL
- "DQL reserved words" on page 41, which cross-references you to a list of the words reserved in DQL

1.1 Literals

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

- "Integer literals" on page 8
- "Floating point literals" on page 8
- "Character string literals" on page 9
- "ID literals" on page 10
- "Date literals" on page 10

1.1.1 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.

1.1.2 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 a property 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. "Valid ranges for floating point literals" on page 9 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 OpenText Documentum CM Administration jobs.

Table 1-1: Valid ranges for floating point literals

RDBMS	Range	Significant digits
Oracle	1.0 x 10-129 to 9.99 x 10-129	15
Microsoft SQL Server	1.7e-308 to 1.7e+308	15
PostgreSQL	1E-307 to 1E+308	15

1.1.3 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.'
```

The maximum length of a character string literal is determined by the maximum allowed by the underlying RDBMS, but in no case will the maximum length exceed 7000 bytes.

If a property is defined as a string datatype, the maximum length of the character string literal you can provide in the property is defined by the property's defined length. If you attempt to provide a longer value in the property using interactive API (IAPI), OpenText™ Documentum™ Content Management Foundation Java API will throw an exception. To change the behavior and allow OpenText Documentum Content Management (CM) Foundation Java API to truncate the character string value to fit the property, set the Foundation Java API preference called `dfc.compatibility.truncate_long_values` in the `dfc.properties` file.

Note: You can provide a longer value in the property using interactive DQL (IDQL) to avoid any exception from Foundation Java API.

1.1.4 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.

1.1.5 Date literals

A date literal specifies a date using the following syntax:

```
DATE('date_value[utc]' [,'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.

If utc is included, Documentum CM Server assumes that the specified *date_value* is UTC time. The specification of utc is not case sensitive.

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).

By default, any date value earlier to 01/01/1753 (January 1, 1753) is considered invalid. Setting such invalid value to a date attribute results in an error. Setting the environment variable `DM_ALLOW_INVALID_TIME` to 1 results in date values earlier to 1st January 1753 to be stored in the database as an invalid date (which is internally considered as nulldate).

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.

1.1.5.1 Default formats

"Default character string formats for dates" on page 11 lists the character string formats for *date_value* that are accepted by default:

Table 1-2: Default character string formats for dates

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')
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 "Default character string formats for dates" on page 11, 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.

1.1.5.1.1 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 platform provides a default short date format. The Windows platform also lets you define your own default short date format. The *OpenText Documentum Content Management - Server Administration and Configuration Guide (EDCCS250400-AGD)* contains more information about defining short date formats. For Linux 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.

1.1.5.1.2 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 Foundation Java API.

1.1.5.1.3 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')
```

1.1.5.2 Date literal keywords

Four keywords represent dates. They are:

- TODAY, which returns the current date in UTC. 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 in UTC. 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 in UTC. 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
  ```

1.1.5.3 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: The default short date format is the format specified in the regional settings accessed through the Control Panel dialog box.
- Linux: The format is assumed to be *mm/dd/yy hh:mi:ss*.

Note: The session config property 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. If you retrieve a property has a Date datatype, you can include a pattern argument that defines the format in which the date is returned.

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

1.1.5.4 Date storage and handling

How dates are stored in the repository and handled during transmission between Documentum CM Server and a client application is not affected by the input or output format chosen for use.

By default, when communicating with a OpenText Documentum CM 6.0 client, a Documentum CM Server connecting to a new repository assumes that all date values in a query are expressed as server local time and stores all dates in the repository normalized to UTC time based on that assumption. This is the recommended format for date storage. Documentum CM Server and Foundation Java API will make the necessary time zone adjustments when dates are sent and received. This behavior can be changed so that Documentum CM Server stores the dates in server local time, rather than in UTC time. However, this is not recommended.

When communicating with a client version earlier than OpenText Documentum CM 6.0, Documentum CM Server assumes that the date values in queries are server local time.

The *OpenText Documentum Content Management - Server Administration and Configuration Guide (EDCCS250400-AGD)* contains complete information about date storage and handling.

1.2 Special keywords

DQL includes some special keywords for use in queries. These keywords have special meanings for Documentum CM Server when used in a DQL query. They cannot be used in Foundation Java 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 property 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 repository. 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.

1.3 Functions

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

- "Scalar functions" on page 16

  Scalar functions operate on one value and return one value.

- "Aggregate functions" on page 19

  Aggregate functions operate on a set of values and return one value.

- "Date functions" on page 21

  Date functions operate on date values.

- "CHECK_ACL function" on page 26

  The CHECK_ACL function enforces security for queries to registered tables.

- "ID function" on page 28

  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.

- "MFILE_URL function" on page 28

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

1.3.1 Scalar functions

The scalar functions are:

- ASCII
- BITAND
- BITCLR
- BITSET
- UPPER
- LOWER
- SUBSTR

1.3.1.1 ASCII

The ASCII function takes one argument and returns the ASCII code value of the first character of the argument. The argument can be a character string literal, a property that has a character string datatype, or the SUBSTR function. The underlying database provides the ASCII function, so there may be slight variations in behavior, depending on the database.

The following example returns the number of documents whose owner name begins with a lower case letter b (ASCII value=98):

```
select count(*) from dm_document where ascii(owner_name)=98
```

1.3.1.2 BITAND, BITCLR, BITSET

The three functions BITAND, BITCLR, and BITSET can appear in the WHERE clause of the DELETE, SELECT, and UPDATE DQL statements, and as a SELECT list item in the SELECT DQL statement. These are bitwise functions that take two integer expressions as arguments and return an integer. They are typically used to test bit values stored in integer properties.

To understand how the return values of the three functions are computed, imagine each input integer as its binary representation (for example 5 is 101 binary and 9 is 1001 binary). Examine the corresponding bits, then AND, clear, or OR each pair (use leading zeros if necessary), and use that value for the corresponding bit of the integer return value.

The BITAND function ANDs the bit values of the two input arguments and returns that result. For example, BITAND(5,9) returns 1, since 101 binary and 1001 binary ANDed together bitwise is 0001 binary.

The BITCLR function sets the bit in the result to zero (clears it), if the corresponding bit in the second parameter is one, otherwise it uses the value of that corresponding bit in the first parameter. For example, BITCLR(5,9) returns 4, since 101 binary cleared bitwise by 1001 binary is 0100 (the bit in the lowest order position of 9 clears the bit in the lowest order position of 5). As another example, BITCLR(9,5) returns 8

(1001 binary cleared bitwise by 101 binary is 1000 binary). Notice that the order of the arguments affects the return result.

The BITSET function ORs the bit values of the two input arguments and returns the ORed value. For example, BITSET(5,9) returns 13, since 101 binary ORed with 1001 binary is 1101 binary.

For example, the following statement returns the user names of all users that have the lowest order bit of the user_privileges property set to one. These users have the privilege of creating types. This query will not return all the users who can create types, since sysadmins and superusers can also create types.

```
SELECT "user_name" FROM "dm_user"
WHERE BITAND("user_privileges",1) = 1
```

The following statement returns the user names of all users that have the bit set of the user_privileges property that represents 16 in binary. These should be all the superusers.

```
SELECT "user_name" FROM "dm_user"
WHERE BITAND("user_privileges",16) = 16
```

1.3.1.3 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 a property 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%'
```

1.3.1.4 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 a property 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'
```

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

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 do not specify a length, the default behavior of the function differs depending on the RDMBS you are using. For Oracle, the default is the entire string value or the full width of the column if a property 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 a property called order_no. The values in this property 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 property 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 statement. 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 property 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 property, the function returns one value for each value in the property.

You cannot use the SUBSTR function in assignment statements.

Additionally, you cannot specify SUBSTR in a LIKE predicate. Refer to “Pattern matching with LIKE” on page 34, for more information about using LIKE.

1.3.2 Aggregate functions

The five aggregate functions are:

- COUNT
- MIN
- MAX
- AVG
- SUM

1.3.2.1 COUNT

The COUNT function counts values. The syntax is:

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

The name argument must identify a property or column. You can count all values for the property 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'
```

1.3.2.2 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 a property 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 a property 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
```

1.3.2.3 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 a property 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 a property 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
```

1.3.2.4 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 a property 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 a property 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
```

1.3.2.5 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 a property 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 a property 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'
```

1.3.3 Date functions

For information about DATE(), refer to "Date literals" on page 10.

Note: When you include the date functions in the selected values list of a SELECT statement, you must use the AS clause to assign a name to the column representing the returned values. For example:

```
SELECT DATEFLOOR(month,"r_creation_date") AS Created...
SELCT DATETOSTRING("r_creation_date",'dd/mm/yyyy hh:mi:ss') AS date_val...
```

1.3.3.1 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 properties 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 repository is using Oracle, the return value is a floating point number.

If the repository is using SQL Server, 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 SQL Server implementation 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'))
```

If the repository is using PostgreSQL, the return value is an integer for all units. If you specify week in the function, the return value will round up if the difference is one half or greater of the specified unit or round down if the difference is less than one half of the specified unit.

1.3.3.2 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)
```

Note: DATEADD function works only with SELECT queries and does not work with UPDATE queries.

1.3.3.3 DATEFLOOR

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

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

The *date* argument is the name of a date property. The function rounds the value of the property 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 property has a value of March 23, 1996 at 9:30:15 am. Using the DATEFLOOR function on this property 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`

1.3.3.4 DATETOSTRING

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

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

The date argument is the name of a date property. The format argument defines how you want the character string formatted. The supported formats are:

Starting with year:

- yyyy_mm_dd hh_mi_ss
- yyyy_mm_dd
- yyyy_mm
- yyyy
- yyyy_mm hh_mi_ss
- yyyy hh_mi_ss
- yy_mm_dd hh_mi_ss
- yy_mm_dd
- yy_mm
- yy_mm hh_mi_ss
- yy hh_mi_ss

Starting with month:

- mm_dd_yyyy hh_mi_ss
- mm_dd_yy hh_mi_ss
- mm_dd_yyyy
- mm_dd_yy
- mm_yyyy
- mm_yy
- mm_yyyy hh_mi_ss
- mm_yy hh_mi_ss
- month dd yy
- month dd yyyy
- month yy
- month yy hh_mi_ss
- month yyyy hh_mi_ss
- month yyyy
- mon dd yy
- mon dd yyyy
- mon_yy
- mon_yy hh_mi_ss
- mon_yyyy
- mon_yyyy hh_mi_ss

Starting with date:

- dd_mm_yy hh_mi_ss
- dd_mon_yy
- dd_mm_yyyy hh_mi_ss
- dd_mon_yyyy
- dd_mon_yyyy hh_mi_ss
- dd_mm_yy
- dd_mm_yyyy

Suppose that a document's r_creation_date property 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`

1.3.3.5 DATEFLOOR_LOCAL

The DATEFLOOR_LOCAL function rounds a date down to the beginning of an indicated granularity (day, month, or year) in the server local time. The syntax is:

```
DATEFLOOR_LOCAL(date_part,date)
```

The *date* argument is the name of a date property. It indicates a date literal, name of a date attribute, or a date keyword. The *date_part* argument indicates the level of granularity. The valid *date_part* values are year, month, and day.

For example, suppose that the r_creation_date property of a document has a value of October 26, 2014 at 9:30:15 AM and the time zone of the Documentum CM Server machine is GMT+5:30. Using the DATEFLOOR_LOCAL function on this property returns these values:

Specifying	Returns
`DATEFLOOR_LOCAL(year,"r_creation_date")`	`1/1/2014 12:00:00 AM (January 1, 2014 00:00:00)`
`DATEFLOOR_LOCAL(month,"r_creation_date")`	`10/1/2014 12:00:00 AM (October 1, 2014 00:00:00)`
`DATEFLOOR_LOCAL(day,"r_creation_date")`	`10/26/2014 12:00:00 AM (October 26, 2014 00:00:00)`

1.3.3.6 DATETOSTRING_LOCAL

The DATETOSTRING_LOCAL function formats a date value to a character string in the server local time. This function always expresses the result in the server local time. The syntax is:

```
DATETOSTRING_LOCAL(date,'format')
```

The *date* argument is the name of a date property. The format can be any of the valid character string input formats in which you want to see the date. If the format is not supported, result is displayed in the default format along with a warning message.

For example, suppose that the r_creation_date property of a document has the value of October 26, 2014 at 9:30:15 AM. Here are some examples of the values returned by the DATETOSTRING_LOCAL function using this date and a variety of formats:

Specifying	Returns
`DATETOSTRING_LOCAL("r_creation_date",'dd-mon-yy')`	`26-Oct-14`
`DATETOSTRING_LOCAL("r_creation_date",'mm/dd/yyyy hh:mi:ss')`	`10/26/2014 09:30:15`

Specifying	Returns
`DATETOSTRING_LOCAL("r_creation_date",'month, yy')`	`October, 14`

1.3.4 CHECK_ACL function

The CHECK_ACL function provides similar security enforcement for queries against registered tables as DQL does for queries against sysobjects. When a non-superuser issues a DQL statement against dm_sysobject or its subtypes, an ACL security-checking clause is appended at the end of the SQL statement issued against the database. The CHECK_ACL function is used similarly to enforce security with registered tables. Since there is no security-checking clause added to DQL statements run against registered tables, the CHECK_ACL function must be ANDed with the other qualifications in the statement. The registered table must contain a column that has the ACL domain and the ACL name for each table row to use for the security check. The syntax of the function is:

```
CHECK_ACL('acl_domain_column','acl_name_column'
,'user_name'[,'active_dynamic_groups_list']])
```

The parameters of the function:

- *acl_domain_column*—the name of the column that contains the ACL domains in the table. This column can be qualified with a table name or its alias. If not qualified, then it must be unique among the registered tables in the FROM clause.
- *acl_name_column*—the name of the column that contains the ACL names in the table. This column can be qualified with a table name or its alias. If not qualified, then it must be unique among the registered tables in the FROM clause.
- *user_name*—an optional parameter that specifies the user against whose security is checked. If not provided, the session user is used.
- *active_dynamic_groups_list*—an optional parameter that specifies the list of active dynamic groups which the specified user is a member of.

1.3.4.1 Examples

For example, if we have a registered table, registered_table_1, like the following:

Table 1-3: registered_table_1

loan_r_object_id	loan_acl_domain	loan_acl_name
0b00000000000001	domainA	car_loan_acl
0b00000000000002	domainA	car_loan_acl
0b00000000000003	domainB	home_loan_acl
0b00000000000004	domainB	home_loan_acl

We can issue the following DQL statement to enforce security for the user, user_1:

```
SELECT <any_attributes_in_the_table>
    FROM registered_table_1
    WHERE loan_r_object_id = '<some_id>' AND
       CHECK_ACL('loan_acl_domain',  'loan_acl_name', 'user_1')
```

The ACL domain and ACL name columns are not qualified because there is only one registered table in the FROM clause. If you have two registered tables to join and use an ACL, then you must provide the table name if the column names are not unique.

The following example assume that registered_table_2 also has columns for loan_acl_domain and loan_acl_name. In this example, you are not required to qualify the ACL domains and names if they are unique among the two registered tables.

```
SELECT <any_attributes_in_either_table>
    FROM registered_table_1, registered_table_2
    WHERE (<predicate_for_table_1_or_2>) AND
       CHECK_ACL('registered_table_1.loan_acl_domain',
                  'registered_table_1.loan_acl_name', '<user_name>')
     AND
         CHECK_ACL('registered_table_2.loan_acl_domain',
                   'registered_table_2.loan_acl_name', '<user_name>')
```

If aliases are used in the FROM clause, then you can use the alias as shown in the following example:

```
SELECT <any_attributes_in_either_table>
    FROM registered_table_1 r1, registered_table_2 r2
    WHERE (<predicate_for_table_1_or_2>) AND
       CHECK_ACL('r1.loan_acl_domain',
                  'r1.loan_acl_name', '<user_name>')
     AND
         CHECK_ACL('r2.loan_acl_domain',
                    'r2.loan_acl_name', '<user_name>')
```

If registered_table_1 has another set of ACL domain and name (for example, customer_acl_domain, customer_acl_name), you can issue the following DQL statement to enforce security by the two ACLs:

```
SELECT <any_attributes_in_either_table>
    FROM registered_table_1
    WHERE (<predicate_for_table_1>) AND
       CHECK_ACL('loan_acl_domain',
                  'loan_acl_name', '<user_name>')
     AND
         CHECK_ACL('customer_acl_domain',
                     'customer_acl_name', '<user_name>')
```

In the preceding example, the two ACL predicates are applied to every attribute in the same row. In the most likely use case, where there is more than one set of ACL domain and name in a registered table, each set of ACL domain and name is relevant to a subset of attributes in the same row. That is, each set of ACL domain and name should be used to enforce security for a subset of, but not all of, the attributes in a row. From the preceding example, if registered_table_1 has other attributes like loan_amount from a loan type and customer_account from a customer type, the following SELECT statement only returns rows that satisfy both ACL predicates:

```
SELECT customer_account, loan_amount
    FROM registered_table_1
    WHERE (<predicate_for_table_1>) AND
       CHECK_ACL('loan_acl_domain',
                 'loan_acl_name', '<user_name>')
     AND
         CHECK_ACL('customer_acl_domain',
                   'customer_acl_name', '<user_name>')
```

For this example, customer_account is returned when the ACL domain and name (customer_acl_domain, customer_acl_name) allow it, regardless of whether the other ACL domain and name (loan_acl_domain, loan_acl_name) allow it or not. Similarly, loan_amount is returned when the ACL domain and name (loan_acl_domain, loan_acl_name) allow it, regardless of the values in customer_acl_domain and customer_acl_name.

1.3.5 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 repository.

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%'
```

1.3.6 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, Documentum CM Server takes the format value from the returned object's a_content_type property.

The page_no argument restricts the returned URLs to those pointing to content files associated with given page number. If you do not 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, Documentum CM 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. Documentum CM Server sets the page_modifier property 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 do not include page_modifier, Documentum CM Server returns the URLs that satisfy the other arguments regardless of whether their content files have a page modifier or not.

1.3.6.1 Examples

The following statement returns the URLs to the jpeg_th renditions of the first content pages of 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'
```

1.4 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 property:

```
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 property value that equals (matches) the specified word (in this case, the word approved).

DQL recognizes these predicates:

- Arithmetic operators (refer to “Arithmetic operators” on page 30)
- Comparison operators (refer to “Comparison operators” on page 30)

- Column and property predicates (refer to "Column and property predicates" on page 31)
- SysObject predicates (refer to "SysObject predicates" on page 37)

1.4.1 Arithmetic operators

Arithmetic operators perform arithmetic operations on numerical expressions. "Arithmetic operators" on page 30 lists the arithmetic operators that you can use as predicates.

Table 1-4: Arithmetic operators

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

1.4.2 Comparison operators

Comparison operators compare one expression to another. "Valid comparison operators for DQL statements" on page 30 lists the comparison operators that can be used as predicates.

Table 1-5: 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

1.4.3 Column and property predicates

Column and property predicates let you compare values in a registered table's columns or an object's properties 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 property. The other group is used when the search criteria specifies a repeating property. Before you use the predicates related to NULL values, please read “NULLs, default values, and DQL” on page 359.

1.4.3.1 Predicates for columns and single-valued properties

The predicates in this group allow you to compare a value in a table column or single-valued property to defined criteria. “Predicates for columns and single-valued properties” on page 31 lists the predicates in this group.

Table 1-6: Predicates for columns and single-valued properties

Predicate	Description
IS [NOT] NULL	Determines whether the property is assigned a value. This predicate is useful only for registered table columns.
IS [NOT] NULLDATE	Determines whether the property is assigned a null date.
IS [NOT] NULLSTRING	Determines whether the property is assigned a null string.
IS [NOT] NULLID	Determines whether the property is assigned a null ID value. Introduced in release version 6.6.
IS [NOT] NULLINT	Determines whether the property is assigned a null integer value.
[NOT] LIKE *pattern* [ESCAPE *character*]	Determines whether the property is like a particular pattern. Refer to “ESCAPE character” on page 36 for information about the ESCAPE clause option.
[NOT] IN *value_list*	Determines whether the property is in one of a particular list of values.
[NOT] EXISTS (*subquery*)	Determines whether the property matches a value found by the subquery.
comparison_op SOME (*subquery*) *comparison_op* ANY (*subquery*)	Determines whether the comparison between the property value and the results of the subquery evaluate to TRUE in any case.
comparison_op ALL (*subquery*)	Determines whether the comparison between the property 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
```

1.4.3.2 Predicates for repeating properties

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

```
[NOT] [ANY] predicate
```

The ANY keyword must be used whenever an expression references a repeating property predicate unless the query also includes the DQL hint, ROW_BASED. If ROW_BASED is included in the query, it is not necessary to include the ANY keyword with repeating property predicates.

"Predicates for repeating properties" on page 32 lists the predicates in this group.

Table 1-7: Predicates for repeating properties

Predicate	Description
attr_name [NOT] LIKE *pattern*[ESCAPE *character*]	Evaluates to TRUE if any value of the repeating property is [not] like a particular pattern. Refer to "ESCAPE character" on page 36 for information about the optional ESCAPE clause.
attr_name IN (*value_list*)	Evaluates to TRUE if any value of the property 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 property 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 on page 373, for more information about these optional hints.
attr_name IS [NOT] NULL	Evaluates to TRUE if any value of the property is [not] null.

Predicate	Description
attr_name IS [NOT] NULLDATE	Evaluates to TRUE if any value of the property is [not] a nulldate.
attr_name IS [NOT] NULLSTRING	Evaluates to TRUE if any value of the property is [not] a null string.
attr_name IS [NOT] NULLID	Evaluates to TRUE if any value of the specified repeating property is [not] a null ID value. Introduced in release version 6.6.
attr_name IS [NOT] NULLINT	Evaluates to TRUE if any value of the specified repeating property 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 property.

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 property'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 property values list and with the keyword working as a value in any position in the list of values in the keywords property.

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 property. You want the name of this object only if the keyword working is in position 3 of the keywords property.

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 properties, respectively, and occupy corresponding positions in the properties.

```
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 properties, refer to "Repeating properties in queries" on page 364.

1.4.3.3 Pattern matching with LIKE

The [NOT] LIKE predicate for both single-valued and repeating properties lets you identify a pattern to match the property 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 property 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, OpenText Documentum CM provides two pattern-matching characters that serve as wildcards. The two characters are:

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

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

1.4.3.3.1 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.

To search for any object that contains a particular string in any position, use a predicate that contains a percent sign both before and after the literal text. For example, the following predicate returns all objects whose subject contains the string 'work':

```
subject LIKE '%work%'
```

By default, this predicate returns all objects whose subject contains “work” in any position, including where the string appears as part of a larger string. For example, it returns any object whose subject contains “networking” or “workers”.

However, for full-text queries only, a behavior change was implemented beginning in release 6.0 to make them more useful and perform better. The fulltext query:

```
object_name like '%mall%'
```

is interpreted as searching for the whole-word mall instead of a word-fragment mall. The following query will behave differently for Documentum CM Server 6.0 and pre-6.0:

```
select r_object_id from dm_document where object_name like '%mall%' enable(ftdql)
```

For pre-6.0, the query produces the following hits:

```
small.txt
the mall document.txt
```

For 6.0 and later, the query produces hits only for:

```
the mall document.txt
```

This change was done to produce more meaningful hits, since word-fragments often produce a mass of unwanted hits and cause performance issues for the full-text engine. Additionally, studies of customer queries showed that users often use this form to look for whole words.

1.4.3.3.2 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.

1.4.3.3.3 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%'
```

1.4.3.3.4 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 wildcard 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 wildcard 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.

1.4.4 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

1.4.4.1 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")
```

1.4.4.2 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 repository. You cannot search a remote folder (a folder that does not reside in the current repository). Valid values are:

- An ID function
- The ID function (described in “ID function” on page 28) 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. The specified folder or folders may have no more than 25,000 nested folders if you include the DESCEND keyword.

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

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.

1.4.4.3 CABINET predicate

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

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

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

- An ID function

  The ID function (described in “ID function” on page 28) must specify a cabinet ID.

- A folder path

  The folder path must identify a cabinet. The 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 repository. 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 repository. It does not return the object's ID in the remote repository.

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

1.5 Concat operator

DQL supports the + (concat) operator for Oracle and SQL Server databases. The concat operator is supported only in SELECT and RETURN_RANGE clauses. Example DQL queries:

- For SELECT: `SELECT r_object_id + object_name FROM dm_server_config`
- For RETURN_RANGE: `SELECT r_object_id, object_name, title FROM dm_folder enable (return_range 1 10 'r_object_id + object_name DESC')`

When you are using the SELECT clause containing the concat operator with RETURN_RANGE, you must create an alias for the columns being concatenated. For example: `select r_object_id+object_name as tempcolumn from dm_document ENABLE (RETURN_RANGE 1 10 'r_object_id asc')`.

1.6 Logical operators

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

- AND
- OR
- NOT

1.6.1 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'
```

1.6.2 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 property 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 properties that do not contain the value Cheeses. The server returns TRUE if the subject is not Cheeses and FALSE if the subject is Cheeses.

1.6.3 Order of precedence

You can join any number of expressions together with logical operators. Documentum CM 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.

1.7 DQL reserved words

"DQL reserved words" on page 426, lists the DQL reserved words. If you use any of these as object or property names, you must enclose the name in double quotes whenever it is used in a DQL statement.

Chapter 2

DQL statements

This section 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 Property Names: It is recommended that you put double quotes around all object type names and property names referenced in applications. Doing that will ensure that the names will not conflict with any DQL reserved words or any words reserved by your underlying RDBMS. OpenText Documentum CM object type names and property names will not generate conflicts, but using the quotes in applications will make sure that no conflicts are generated by user-defined type or property names. To encourage this best practice, the DQL examples in our guides use the double quotes.

Comments within DQL Statements: DQL does not accept inline comments. An inline comment is a comment embedded within a DQL statement.

2.1 Abort

Cancels an explicit transaction.

2.1.1 Syntax

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

2.1.2 Description

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 repository.

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.

2.1.3 Related statements

"Begin Tran" on page 69
"Commit" on page 72

2.2 Alter Group

Modifies a user group.

2.2.1 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
```

2.2.2 Arguments

Table 2-1: ALTER GROUP argument descriptions

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. In addition, if the name contains German umlauts, enclose the name in single quotes. For example, `ALTER GROUP 'köln' ADD 'dmadmin';` **Note:** In API, names containing German umlauts need not be enclosed in single quotes.

Variable	Description
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” on page 46). When you are identifying an individual user, the name must be the value of the user's user_name property.
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.

2.2.3 Permissions

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

2.2.4 Description

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.

2.2.4.1 SET PRIVATE clause

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

2.2.5 Related statements

“Create Group” on page 73
“Drop Group” on page 101

2.2.6 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'
```

2.3 Alter Aspect

Adds, drops, or changes fulltext-indexed properties for an aspect.

2.3.1 Syntax

```
ALTER ASPECT aspect_name ADD
[(property_def {,property_def})]
[[NO]OPTIMIZEFETCH]
```

```
ALTER ASPECT aspect_name FULLTEXT SUPPORT ADD ALL
ALTER ASPECT aspect_name FULLTEXT SUPPORT DROP ALL
ALTER ASPECT aspect_name FULLTEXT SUPPORT ADD property_list
ALTER ASPECT aspect_name FULLTEXT SUPPORT DROP property_list
```

2.3.2 Arguments

Table 2-2: ALTER ASPECT argument descriptions

Variable	Description
aspect_name	Identifies the aspect for which the properties are defined. Use the object_name of a dmc_aspect_type object.
[NO] OPTIMIZEFETCH	Indicates whether to duplicate the names and values of the properties in the object's property bag when the aspect is attached to an object that has a property bag. OPTIMIZEFETCH directs Documentum CM Server to store the values in the object's property bag. NO OPTIMIZEFETCH directs Documentum CM Server not to duplicate the properties into the property bag. The default is OPTIMIZEFETCH for the release version earlier than 6.7, and is NO OPTIMIZEFETCH for the release version 6.7 and later.

Variable	Description
property_def	Defines a property for the aspect. Each property definition has the following format: `property_name domain[REPEATING] [(property_modifier_list)]` *property_name* names the property and *domain* defines its datatype. The property 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. **Note:** Reserved words should not be used as property names in aspects. REPEATING defines the property as a repeating property.
	property_modifier_list defines data dictionary information for the property. Valid property modifiers are: *update_modifier value_assistance_specification* *mapping_table_specification* *default_specification* *constraint_specification* Separate multiple modifiers with commas. You cannot include a property modifier if the statement is inside an explicit transaction. Refer to "Property modifiers" on page 87, for descriptions of the property modifiers. You can specify a maximum of 30 properties in an ALTER ASPECT statement.
property_list	Identifies the properties to be indexed or dropped from indexing. The properties must be properties defined for the aspect identified in *aspect_name*.

2.3.3 Permissions

You must have Superuser privileges to use this statement.

2.3.4 Description

Use the ALTER ASPECT statement to add properties to aspects or to drop or modify properties already attached to an aspect. The statement is also used to define or modify the fulltext indexing of aspect properties.

2.3.4.1 Adding properties

You can add a maximum of 30 properties to an aspect in a single operation. If you wish to add more than 30 to a particular aspect, you must issue multiple ALTER ASPECT statements.

You cannot add properties to an aspect whose name includes a dot. For example, if the name of the aspect is “com.mycompany.policy”, then you cannot define properties for that aspect.

If the ALTER ASPECT statement is inside an explicit transaction, you cannot include property modifiers (data dictionary information) in the definition.

If you specify OPTIMIZEFETCH when you first add properties to an aspect, you must specify it also if you add more properties later. Similarly, if you specify NO OPTIMIZEFETCH, when you first add properties to an aspect, you must specify NO OPTIMIZEFETCH also if you add more properties later.

When you add a property to an aspect that is already attached to one or more objects, the value for that property in those objects is the null value (nulldate, nullstring, and so forth).

2.3.4.2 Dropping properties

Dropping properties of an aspect also removes the property from objects to which the aspect is attached.

2.3.4.3 [NO] OPTIMIZEFETCH

You can choose to optimize performance for fetching or querying aspect properties by including the OPTIMIZEFETCH keyword in the ALTER ASPECT statement. That keyword directs Documentum CM Server to duplicate the properties and their values into the property bag of any object to which the aspect is attached, if the object has a property bag.

When aspects were first introduced, it was assumed that better performance in fetching qualifiable aspect properties would result by using OPTIMIZEFETCH when adding properties to an aspect. OPTIMIZEFETCH was set to be the default for the ALTER ASPECT DQL command. However, as we accumulated more experience, it became clear that NO OPTIMIZEFETCH gives better performance. From the 6.7

release, the default is changed from OPTIMIZEFETCH to NO OPTIMIZEFETCH for the ALTER ASPECT DQL command. We recommend that you use the NO OPTIMIZEFETCH option explicitly, in releases earlier than 6.7, when using this command.

The *OpenText Documentum Content Management - Server Fundamentals Guide (EDCCS250400-GGD)* contains more information about property bags, non-qualifiable properties, and how they are stored in the object type tables.

2.3.4.4 Full-text indexing of aspect properties

Properties associated with aspects are not fulltext-indexed by default. If you wish to index them, you must issue an ALTER ASPECT statement to identify the aspects you want indexed. “Syntax variations for full-text-indexing of aspect properties” on page 49, describes the syntax options for specifying full-text indexing and what each defines.

Table 2-3: Syntax variations for full-text-indexing of aspect properties

Syntax	Description
FULLTEXT SUPPORT ADD ALL	Defines all properties of the aspect for indexing
FULLTEXT SUPPORT ADD *property_list*	Defines for indexing only those aspect properties listed in *property_list*
FULLTEXT SUPPORT DROP ALL	Stops indexing of all properties of the aspect
FULLTEXT SUPPORT DROP *property_list*	Stops indexing of those aspect properties listed in *property_list*

When you add or drop indexing for aspect properties, only new objects are affected. The index is not updated to add or drop aspect property values for aspects attached to existing objects.

2.3.5 Related statements

None

2.3.6 Examples

```
ALTER ASPECT grant_validation
ADD(grant_amount integer, disbursement_date date,
approved_by string(32))NO OPTIMIZEFETCH
```

2.4 Alter Type

Modifies an object type.

2.4.1 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 (property_modifier_clause)[PUBLISH]
```

```
ALTER TYPE type_name
ADD property_def {,property_def}[PUBLISH]
```

```
ALTER TYPE type_name
DROP property_name {,property_name}[PUBLISH]
```

```
ALTER TYPE type_name ALLOW ASPECTS
```

```
ALTER TYPE type_name
ADD|SET|REMOVE DEFAULT ASPECTS aspect_list
```

```
ALTER TYPE type_name ENABLE PARTITION
```

```
ALTER TYPE type_name SHAREABLE [PUBLISH]
```

```
ALTER TYPE type_name FULLTEXT SUPPORT [
 NONE |
 LITE ADD ALL
 LITE ADD property_list |
 BASE ADD ALL |
 BASE ADD property_list
]
```

2.4.2 Arguments

Table 2-4: ALTER TYPE argument descriptions

Variable	Description
type_name	Identifies the type to alter. Use the name defined for the type when it was created. If you are altering a type to allow aspects, *type_name* must be the top-most type in the type's hierarchy. Refer to "Allowing aspects" on page 57, for more information. If you are altering the type to add default aspects, the type may not be lightweight object type and the type must have the r_aspect_name property. If you are altering the type to enable data partitioning, *type_name* must be the top-most type in the type's hierarchy. Refer to "Allowing partitioning" on page 59, for more information. If you are altering the type to modify fulltext support, using the FULLTEXT SUPPORT keywords, *type_name* must be the name of a lightweight type.
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.
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" on page 92 for information about the syntax and use of each specification.

Variable	Description
property_modifier_clause	Defines the change you want to make to the property. The syntax for this clause is: *property_name domain* [SPACEOPTIMIZE] or *property_name (property_modifier_list)* *domain* is any valid DQL datatype. *property_modifier_list* lists one or more modifiers that set or alter data dictionary information for the property. Valid property modifiers are: *update_modifier* *value_assistance_specification* *mapping_table_specification* *default_specification* *constraint_specification* *property_drop_clause* Separate multiple modifiers with commas. Refer to “Property modifiers” on page 87 for information about the syntax and use of each property modifier.
property_def	Defines the properties you want to add to the type. *property_def* has the following syntax: `property_name domain[REPEATING] [[NOT]QUALIFIABLE | SPACEOPTIMIZE] (property_modifier_list)` *property_name* is the name of the property and *domain* is any valid DQL datatype. The keyword REPEATING defines the property as a repeating property. By default, a property is qualifiable. Include NOT QUALIFIABLE if you wish to define the property as non-qualifiable. The length of a NOT QUALIFIABLE property with a string datatype must be less than the value in the max_nqa_string key in the server.ini file if that is set. Refer to “QUALIFIABLE and NOT QUALIFIABLE properties” on page 55, for more information about QUALIFIABLE. SPACEOPTIMIZE allows you to assign true NULLs as a property value. *property_modifier_list* lists one or more modifiers that define data dictionary information for the property. Refer to the preceding description of *property_modifier_clause* for a list of valid property modifiers. Separate multiple modifiers with commas.
property_name	For the DROP option, *property_name* identifies the property that you want to remove from the specified type.

Variable	Description
aspect_list	Comma-separated list of aspects. Use the value of the object_name property of their dmc_aspect_type objects. You are not required to quote the aspect list; however, if you do, use single quotes. "Adding and setting default aspects" on page 58, describes how default aspects are added to an object type. "Removing default aspects" on page 59, describes how default aspects are removed from an object type.

2.4.3 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 property of the repository configuration (docbase config object).

"Alter Type operations" on page 53, lists the ALTER TYPE operations and describes who can perform them and on which types.

Table 2-5: Alter Type operations

Alteration	Who can do it	To which types
Set default ACL	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 the exception that value assistance and constraints may not be added to system-defined object types.
Add read/write properties	Type owner or Superuser	User-defined only.
Add read-only properties	Superuser	User-defined only.
Drop read/write properties	Type owner or Superuser	User-defined only.
Lengthen character string properties	Type owner or Superuser	User-defined only.
Allow aspects	Superuser	Only user-defined object types that have no supertype (that is, user-defined object types that are the top-most type in their hierarchy).
Add, set, or remove default aspects	Superuser	Any object type that is not a lightweight object type and that has the r_aspect_name property.

Alteration	Who can do it	To which types
Enable type to be partitionable	Type owner or Superuser	Only user-defined object types that have no supertype (that is, user-defined object types that are the top-most type in their hierarchy).
Make type shareable	Type owner or Superuser	User-defined only.
Allow properties to store true NULL values.	Type owner or Superuser	User-defined only. (However, see Appendix B, Database footprint reduction of dmr_content objects on page 389.)

The *OpenText Documentum Content Management - Server System Object Reference Guide (EDCCS250400-ORD)* contains information about the definition of a read and write or read-only property.

2.4.4 Description

This section contains information about using the statement.

2.4.4.1 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 "Alter Type operations" on page 53 for a summary of valid operations, the types on which they can be performed, and who can perform them.

You cannot issue an ALTER TYPE statement in an explicit transaction.

In release 6.0, a dynamic type feature was added so that non-intrinsic types will be updated immediately. Intrinsic type names start with dm_, dmi_, and dmr_ but do not include these types: dm_message_archive, dm_message_container, dm_acs_config, dm_validation_descriptor, dm_cont_transfer_config, and aspect attribute types. In previous releases, after you execute an ALTER TYPE statement, the changes were not reflected to users until the global type cache was updated. The updating was automatic, but could take several minutes. Users currently in a session with the repository will not see the type change if they performed operations on any instance of the type in their session, prior to the change. When a user performs an operation on an object, the system caches the object type definition for that object.

2.4.4.2 QUALIFIABLE and NOT QUALIFIABLE properties

A qualifiable property is a standard property. It is stored in a column in the appropriate table in the database in which the repository resides. Queries can reference qualifiable properties in selected value lists and in expressions in qualifications. Properties are qualifiable by default.

A property defined as NOT QUALIFIABLE is stored in a property bag. A non-qualifiable property can be referenced in selected value lists, but may not be referenced in expressions or qualifications in a query.

If a property is a string datatype and it is defined as NOT QUALIFIABLE, its length must be less than the value in the max_nqa_string key in the server.ini file if that key is set.

The *OpenText Documentum Content Management - Server Fundamentals Guide (EDCCS250400-GGD)* contains more information about property bags, non-qualifiable properties, and how they are stored in the object type tables.

2.4.4.3 SPACEOPTIMIZE properties

In releases prior to 6.6, Documentum CM Server did not allow you to assign true (actual) NULLs as a property value, as described in the section, “NULLs, default values, and DQL” on page 359 . Use SPACEOPTIMIZE in the *property_modifier_clause* to change the property, or in the *property_def* clause to add a property that allows true NULL values. If you want to modify a string property to allow actual NULL values, you must specify the *domain* as string(0). If you wish to also lengthen the string value, you must do it in another *property_modifier* clause. The property values stored before the property was modified will not change. The change only affects newly stored values.

For Oracle installations, SPACEOPTIMIZE can modify all properties. For SQL Server and PostgreSQL, only the character and string properties and the ID properties can be set to SPACEOPTIMIZE.

For example, if you created a property named “attribute1” in “my_type1” with domain of STRING(32), you could alter that property to use true NULLs with the following statement:

```
ALTER TYPE "my_type1" MODIFY (attribute1 STRING(0) SPACEOPTIMIZE)
```

Note: The recommendation is that ID attributes should only have SPACEOPTIMIZE applied when it is expected that most of the time the value will be NULLID (that is, 0000000000000000). This is because when a non-null ID value is stored, VARCHAR(16) causes 17 bytes of data to be allocated in SQL Server versus 16 for CHAR(16). This is why we do not put SPACEOPTIMIZE on r_object_id since every object has a non-null value.

2.4.4.4 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 property).

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

2.4.4.5 PUBLISH

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

If you do not include PUBLISH, the information is published the next time the Data Dictionary Publish job runs.

It is not necessary to include PUBLISH if you are dropping a property from the type definition. The data dictionary information for the property is automatically removed in such cases.

2.4.4.6 Modifying a type definition

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

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

2.4.4.6.1 Adding a new property

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

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

property_def defines the new property and has the following syntax:

```
property_name domain [REPEATING]
[[NOT]QUALIFIABLE](property_modifier_list)
```

The property name must following the naming rules outlined in the *OpenText Documentum Content Management - Server System Object Reference Guide (EDCCS250400-ORD)*.

domain is any valid DQL datatype. The keyword REPEATING defines the property as a repeating property. The keyword QUALIFIABLE identifies the property as a qualifiable property. If you include NOT QUALIFIABLE, the property is a non-qualifiable property. QUALIFIABLE is the default. For information about this property characteristic, refer to “QUALIFIABLE and NOT QUALIFIABLE properties” on page 55.

Valid property modifiers are described in “Property modifiers” on page 87.

Do not add more than 30 properties in one ALTER TYPE statement.

2.4.4.6.2 Lengthening a character string property

The character string property must be defined for the object type. It cannot be an inherited property. The only valid change that you can make is to lengthen the property (or to allow true NULL values). 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 property 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))
```

2.4.4.6.3 Dropping properties from the type

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

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

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

2.4.4.7 Allowing aspects

Altering an object type to allow instances of the type to accept aspects performs the following actions on the object type:

- Adds the repeating property, r_aspect_name, to the type definition if the type does not have the property already
- Adds the i_property_bag property to the type definition if the type does not have the property already

2.4.4.8 Adding and setting default aspects

A default aspect is an aspect that is associated with an object type and, by default, associated with each instance of the type that is created after the aspect is added to the type. Default aspects defined for an object type are also default aspects for subtypes of that type.

ALTER TYPE supports two key words to add default aspects to an object type definition: ADD and SET. The ADD keyword appends the specified aspect or aspects to any others already associated with the object type. The SET keyword replaces the object type's existing defined default aspects with the aspects specified in the statement. The SET keyword replaces only the default aspects explicitly defined for the type. It does not replace any inherited default aspects.

For example, suppose you have an object type named “my_custom_type” with two default aspects, one that is inherited, named full_validation, and one that is defined for the type: quick_validation. You want to add an aspect named correct_figures. The following statement adds the correct_figures aspect to the object type definition while retaining the other two default aspects:

```
ALTER TYPE my_custom_type ADD DEFAULT ASPECTS correct_figures
```

After the statement completes, my_custom_type has three default aspects: full_validation, which is inherited, and quick_validation and correct_figures, which are defined explicitly for my_custom_type.

Now, suppose that your application developers have created a new version of the correct_figures aspect that also performs a quick validation. This new version is called correct_figures_v2. To remove the current correct_figures and quick_validation aspects and replace them with the new, combined aspect, use the following statement:

```
ALTER TYPE my_custom_type SET DEFAULT ASPECTS correct_figures_v2
```

my_custom_type now has the following default aspects: full_validation and correct_figures_v2. The SET keyword replaced both aspects defined for the type with the one aspect specified in the statement, but left the inherited aspect alone.

When you add an aspect to an object type, only the new instances of the type created after the addition are affected. The aspect is not attached to existing instances of the type.

2.4.4.9 Removing default aspects

Use the REMOVE keyword to remove default aspects from an object type definition. You can remove any default aspect defined specifically for the object type. You cannot remove a default aspect inherited by the object type.

The statement removes the aspects specified in *aspect_list*. For example, the following statement removes the aspect named conditional_validation from my_custom_type:

```
ALTER TYPE my_custom_type REMOVE DEFAULT ASPECT conditional_validation
```

An error is returned if the specified aspect is not associated with the specified object type definition.

When you remove a default aspect from an object type, existing instances of the type are not affected. The aspect remains attached to those instances of the type. You must remove the aspect from the existing type instances explicitly after removing the default aspect from the type definition.

2.4.4.10 Allowing partitioning

Altering an object type to be partitionable adds an additional attribute, i_partition, to the type and its subtypes. Since an object creates database entries in the tables for its type and all its supertypes, the whole type hierarchy must be either partitionable or non-partitionable. Predefined types are already partition enabled (or not), so only user-defined supertypes can be modified to be partitionable. For example, the following statement makes my_custom_type a partitionable type:

```
ALTER TYPE my_custom_type ENABLE PARTITION
```

An error is returned if the type is already partitionable, or the type is not a user-defined type that has no supertype.

If you alter a type to be partitionable, you must use the Administrative method PARTITION_OPERATION to take advantage of data partitioning.

2.4.4.11 Making a type shareable

Altering a type to be shareable adds additional properties, i_sharing_type, i_orig_parent, and allow_propagating_changes to the type, and marks it as a shareable type. An error will occur if you attempt to alter an already shareable type or a lightweight type to be shareable.

2.4.4.12 Fulltext support for lightweight types

Use the FULLTEXT SUPPORT arguments to specify which properties are included in the fulltext index for lightweight types:

- FULLTEXT SUPPORT—Default, all lightweight attributes are indexed
- FULLTEXT SUPPORT NONE—No attributes are indexed
- FULLTEXT SUPPORT LITE ADD ALL—All lightweight attributes are indexed
- FULLTEXT SUPPORT LITE ADD *property_list*—Selected lightweight attributes are indexed only (cannot select parent attributes)
- FULLTEXT SUPPORT BASE ADD ALL—All lightweight and parent attributes are indexed
- FULLTEXT SUPPORT BASE ADD *property_list*—Selected lightweight or parent attributes are indexed

2.4.5 Type modifiers

Type modifiers set or change some properties 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.

2.4.5.1 update_modifiers

Update modifiers set or remove property 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 properties 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.

2.4.5.1.1 Setting or removing data dictionary values

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

```
SET property_name[[index]]=valueAPPEND property_name=valueINSERT
property_name[[index]]=valueREMOVE property_name[[index]]
TRUNCATE property_name[[index]]
```

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

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

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

  If the operation is adding a new value to the property, 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 properties because it automatically puts the new value at the end of the repeating property'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 property is single-valued, *value* can be NULL.

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

- 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

2.4.5.1.2 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 repository owner. The combination of the ACL name and domain uniquely identifies the ACL within the repository. The *OpenText Documentum Content Management - Server Administration and Configuration Guide (EDCCS250400-AGD)* or *OpenText Documentum Content Management - Server Fundamentals Guide (EDCCS250400-GGD)* contains more information about ACLs and their names and implementation.

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 properties for the type to NULL also.

2.4.5.1.3 Defining the default storage area

The default storage area for a type is where Documentum CM 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.

2.4.5.1.4 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
```

2.4.5.1.5 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.

2.4.5.2 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 properties 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.

2.4.5.3 constraint_specification

You can define the `Check` constraint in a type modifier list.

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
```

The FOR POLICY clause can 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* to include the FOR POLICY clause.

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.

2.4.5.4 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" on page 94.

2.4.5.5 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 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*] properties, 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.

2.4.6 Property modifiers

Property modifiers set or drop data dictionary information for a property. Data dictionary information includes constraints, value assistance, default value definitions, and mapping information. Property modifiers set properties of dm_nls_dd_info and dm_dd_info objects.

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

2.4.6.1 update_modifiers

An update modifier lets you directly set a property 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 (property_name (update_modifier))
```

Including the FOR POLICY clause makes the change to the property 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 a property, refer to “update_modifier” on page 87.

2.4.6.2 value_assistance_specification

A value assistance specification identifies one or more values for a property. Typically, value assistance is used to populate a pick list of values for the property 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 a property:

```
ALTER TYPE type_name[FOR POLICY policy_id STATE state_name]
MODIFY (property_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" on page 88.

2.4.6.3 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 a property 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 (property_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" on page 89.

2.4.6.4 default_specification

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

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

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

```
ALTER TYPE type_nameMODIFY (property_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 properties.)

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

```
DEFAULT[=]default_value
```

For a repeating property, 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 property and one repeating property 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 property. For example, you cannot specify the keyword USER for a date property. You cannot specify NULL for a repeating property.

2.4.6.5 constraint_specification

You can specify the following constraints in a property modifier list:

- Not null
- Check

You cannot add a constraint to a system-defined object type.

The syntax for adding a constraint to a property is:

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

The FOR POLICY clause can 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" on page 90.

2.4.6.6 property_drop_clause

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

- 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*] properties, 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 (property_name (DROP VALUE ASSISTANCE))
```

- To drop mapping information, use the following syntax:

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

2.4.7 Related statements

"Alter Aspect" on page 46
"Create Type" on page 79
"Drop Type" on page 102

2.4.8 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 a property 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 property 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 next example demonstrates the use of single quotes in setting the ACL:

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

This final example changes the type my_super to a partitionable type:

```
ALTER TYPE "my_super" ENABLE PARTITION
```

2.5 Begin Tran

Opens an explicit transaction.

2.5.1 Syntax

```
BEGIN TRAN[SACTION]
```

2.5.2 Permissions

Anyone can execute the BEGIN TRAN statement.

2.5.3 Description

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 property values in objects are saved until you issue a COMMIT statement to commit the changes.

2.5.4 Related statements

"Abort" on page 43
"Commit" on page 72

2.6 Change...Object

Changes the object type of one or more objects.

2.6.1 Syntax

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

2.6.2 Arguments

Table 2-6: CHANGE...OBJECT argument descriptions

Variable	Description
current_type	Identifies the object type of the objects to change. Use the type's name.
new_type	Specifies the new object type for the objects. Use the type's name. Valid new types depend on the object's current type. Refer to "Rules and constraints on changing types" on page 71 for a complete description.
update_list	Specifies one or more change operations to perform on the objects you are changing. Valid formats are: `set property_name = valueset property_name[[index]] = valueappend [n] property_name = valueinsert property_name[[index]] = valueremove property_name[[index]] truncate property_name[[index]]` If you specify more than one operation, use commas to separate the clauses. Refer to "update operations" on page 176 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 "IN ASSEMBLY clause" on page 150.
SEARCH clause	Restricts the candidates for change to those that satisfy the full-text search condition. Refer to "SEARCH clause" on page 151 for a description of its syntax and use.
WHERE clause	Defines a qualification used to restrict what objects are changed. Refer to "WHERE clause" on page 155 for a description of a valid qualification.

2.6.3 Return value

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

2.6.4 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 property), you must be have Superuser user privileges or Change Ownership privilege.

2.6.5 Description

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

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

Objects that meet the criteria in the statement are deleted from the repository 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.

2.6.5.1 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.

  The type identifier is the first two characters in an object ID. This value is inherited from the type's supertype. For example, all dm_document subtypes have 09 as their type identifier.
- The old and new types cannot be at the same level in the type hierarchy. The new type must be a supertype or subtype of the current type. That is, the new type must be in a direct hierarchical line from the current type.

  You can move objects up or down in the object hierarchy but not laterally. For example, suppose TypeA and TypeB are both subtypes of mybasetype, and TypeC is a subtype of TypeB. You can change objects of Type B to mybasetype or to TypeC. You cannot change objects of TypeA directory to TypeB because these two types are peers in the hierarchy. Nor can you change objects of TypeC to TypeA because TypeA is not a supertype or subtype of TypeC.
- You cannot change an object type that is not a subtype of dm_sysobject using DQL.

When you change an object to a type that is higher in the hierarchy, the server drops any properties 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 properties in the new type that were not in the old type. Unless you use the update_list option to set these new properties, 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

2.6.5.2 Using (ALL) option

Including (ALL) directs Documentum CM Server to consider all versions of the object for the change. If you do not include (ALL), the server only considers the CURRENT version for the change. ALL must be enclosed in parenthesis.

2.6.6 Related statements

"Create...Object" on page 76
"Delete...Object" on page 97
"Update...Object" on page 171

2.6.7 Examples

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

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

2.7 Commit

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

2.7.1 Purpose

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

2.7.2 Syntax

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

2.7.3 Permissions

Anyone can execute the COMMIT statement.

2.7.4 Description

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

You can include either TRAN or TRANSACTION.

2.7.5 Related statements

"Abort" on page 43
"Begin Tran" on page 69

2.8 Create Group

Creates a user group.

2.8.1 Syntax

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

2.8.2 Arguments

Table 2-7: CREATE GROUP argument descriptions

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 repository. You can specify the name as an identifier or as a character string literal. In addition, if the name contains German umlauts, enclose the name in single quotes. For example, `CREATE GROUP 'köln';` **Notes** • In API, names containing German umlauts need not be enclosed in single quotes. • Documentum CM 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 property.

2.8.3 Return value

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

2.8.4 Permissions

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

2.8.5 Description

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

2.8.5.1 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 property for the group. The setting is not used by Documentum CM Server. All groups, public or private, are visible in the dm_group type. This property is provided for use by user-written applications.

2.8.6 Related statements

"Alter Group" on page 44
"Drop Group" on page 101

2.8.7 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
```

2.9 Create...Object

2.9.1 Purpose

Creates an object.

2.9.2 Syntax

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

2.9.3 Arguments

Table 2-8: CREATE...OBJECT argument descriptions

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 repository with the exception of the object types that represent aspect properties. *type_name* cannot be the name of a type representing a type describing aspect properties.

update_list	Specifies one or more operations you want to perform on the new object. Valid formats are:
	`set property_name = value`
	`set property_name[[index]] = valueappend`
	`[n]property_name = valueinsert`
	`property_name[[index]] = valueremove`
	`property_name[[index]] truncate`
	`property_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" on page 78 for details.
SETFILE clause WITH PAGE_NO option	Adds additional content to the object. Refer to "WITH PAGE_NO option" on page 78 for details.

2.9.4 Return value

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

2.9.5 Permissions

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

2.9.6 Description

The CREATE...OBJECT statement creates and saves a new object. As part of the process, you can set some of the object's properties, 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.

2.9.6.1 SETFILE clauses

A SETFILE clause adds content to the new object. You must have Superuser privileges to include the clause in the statement. Using the SETFILE clause is subject to the following conditions:

- The content must be a file. It cannot be a block of data in memory.
- The content file must be located in a directory visible to Documentum CM Server.
- You cannot add a file created on a Macintosh machine.
- The content cannot be stored in content-addressed storage or 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. 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.

2.9.6.1.1 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 Documentum CM Server.

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

2.9.6.1.2 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 Documentum CM 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.

2.9.7 Related statements

"Change...Object" on page 69
"Delete...Object" on page 97
"Update...Object" on page 171

2.9.8 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 Linux 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 Linux:

```
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
```

2.10 Create Type

Creates an object type.

2.10.1 Syntax

To create a type:

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

To create a partitionable type:

```
CREATE PARTITIONABLE TYPE type_name
[(property_def {,property_def})]
[WITH] SUPERTYPE NULL
[type_modifier_list] [PUBLISH]
```

To create a shareable type:

```
CREATE SHAREABLE TYPE type_name
[(property_def {,property_def})]
[WITH] SUPERTYPE parent_type [PUBLISH]
```

To create a lightweight type:

```
CREATE LIGHTWEIGHT TYPE type_name
[(property_def {,property_def})]
SHARES shareable_type
[AUTO MATERIALIZATION |
MATERIALIZATION ON REQUEST |
DISALLOW MATERIALIZATION]
[FULLTEXT SUPPORT [
 NONE |
 LITE ADD ALL
 LITE ADD property_list |
 BASE ADD ALL |
 BASE ADD property_list
]
[PUBLISH]
```

2.10.2 Arguments

Table 2-9: CREATE TYPE argument descriptions

Variable	Description
type_name	Names the new type. Use any valid name that is unique among the other user-defined type names in the repository. 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.

Variable	Description

property_def	Defines a property for the new type. You can define up to 30 properties. Each property definition has the following format:
	`property_name domain[REPEATING]`
	`[[NOT] QUALIFIABLE | SPACEOPTIMIZE]`
	` [(property_modifier_list)]`
	property_name names the property and *domain* defines its datatype. The property name must consist of ASCII characters. The domain can be any valid DQL datatype. If the datatype is a character or string datatype, the *domain* specification must also include the length.
	The domain specification is one of the following:
	• bool[ean]
	• char[acter]
	• date
	• double
	• float
	• ID
	• integer
	• smallint (Oracle uses NUMBER(5))
	• string
	• time
	• tinyint (Oracle uses NUMBER(3))
	REPEATING defines the property as a repeating property.
	By default, a property is qualifiable. Include NOT QUALIFIABLE if you wish to define the property as non-qualifiable. The length of a NOT QUALIFIABLE property with a string datatype must be less than the value in the max_nqa_string key in the server.ini file if that is set. Refer to "QUALIFIABLE and NOT QUALIFIABLE properties" on page 84, for more information about QUALIFIABLE.
	SPACEOPTIMIZE allows you to assign true NULLs as a property value.

Variable	Description
property_def continued	*property_modifier_list* defines data dictionary information for the property. Valid property modifiers are: *update_modifier value_assistance_specification* *mapping_table_specification* *default_specification* *constraint_specification* Separate multiple modifiers with commas. You cannot include a property modifier if the statement is inside an explicit transaction. Refer to "Property modifiers" on page 87 for descriptions of the property modifiers.
parent_type	Identifies the supertype of the new type. Valid supertypes for non-shareable types are: • dm_category • dm_email_message **Note:** dm_email_message is a deprecated object type. • dm_message_archive • dm_relation • dm_state_extension • dm_state_type • dm_sysobject and its subtypes • dm_taxonomy • dm_user and its subtypes • user-defined types Valid supertypes for shareable types are: • dm_sysobject and its subtypes • user-defined types To create a type with no supertype, specify *parent_type* as NULL. This requires the Superuser user privilege.

Variable	Description
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" on page 92 for descriptions of the type modifiers.
shareable_type	A previously defined shareable type that will be the parent of the lightweight type.
property_list	A comma-separated list of properties.

2.10.3 Return value

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

2.10.4 Permissions

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

To define read-only properties for a new type (properties 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 properties, your session locale must match exactly one of the locales defined in the dd_locales property of the repository configuration (docbase config object).

2.10.5 Description

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

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

Do not define more than 30 properties in a single CREATE TYPE statement. If the type has more than 30 properties, use ALTER TYPE to add the additional properties. The total number of properties in the new type cannot exceed the supported number of columns in a table in the underlying RDBMS.

Additionally, be sure to follow the naming rules for properties described in the *OpenText Documentum Content Management - Server System Object Reference Guide (EDCCS250400-ORD)*.

2.10.5.1 QUALIFIABLE and NOT QUALIFIABLE properties

A qualifiable property is a standard property. It is stored in a column in the appropriate table in the database in the repository. You can reference qualifiable properties in selected value lists and in expressions in qualifications in queries. Properties are qualifiable by default.

A property defined as NOT QUALIFIABLE is stored in a property bag. A non-qualifiable property can be referenced in selected value lists, but not in qualifications except under certain circumstances.

If a property is a string datatype and it is defined as NOT QUALIFIABLE, its length must be less than the value in the max_nqa_string key in the server.ini file if that key is set.

The *OpenText Documentum Content Management - Server Fundamentals Guide (EDCCS250400-GGD)* contains more information about property bags, non-qualifiable properties, and how they are stored in the object type tables.

2.10.5.2 SPACEOPTIMIZE properties

In releases prior to 6.6, Documentum CM Server did not allow you to assign true (actual) NULLs as a property value, as described in the section, "NULLs, default values, and DQL" on page 359 . Use SPACEOPTIMIZE in the *property_def* clause to add a property that allows true NULL values. For Oracle installations, SPACEOPTIMIZE can modify all properties. For SQL Server, only the character and string properties and the ID properties can be set to SPACEOPTIMIZE.

For example, to create a type, my_type_a, with a property named attr1 of string(32) and using true NULLs, use the following statement:

```
CREATE TYPE "my_type_a" ( attr1 STRING(32) SPACEOPTIMIZE) SUPERTYPE "dm_document"
```

Note: The recommendation is that ID attributes should only have SPACEOPTIMIZE applied when it is expected that most of the time the value will be NULLID (that is, 0000000000000000). This is because when a non-null ID value is stored, VARCHAR(16) causes 17 bytes of data to be allocated in SQL Server versus 16 for CHAR(16). This is why we do not put SPACEOPTIMIZE on r_object_id since every object has a non-null value.

2.10.5.3 Specifying an ACL for the type

If one or more servers in the repository is using type-based ACL inheritance, it is recommended that you make sure that the object type has an ACL defined for the type. The ACL does not determine who can use the type, but serves as the default ACL for objects of the type when users create the objects without naming an ACL for the objects. After you create the object type, use ALTER TYPE to set the acl_name and acl_domain properties for the object type if needed.

Note: Default ACL inheritance is defined in the server config object, in the default_acl property. If that property is set to 2, for type-based ACL inheritance, when a user creates an instance of the type and does not explicitly assign an ACL to the object, the server will assign the ACL associated with the object type.

2.10.5.4 Localization

When you create a new type, any data dictionary information that you define for the type or its properties in the statement is published in all locales identified in the dd_locales property of the repository configuration (docbase config object).

2.10.5.5 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 do not include PUBLISH, the information is published the next time the Data Dictionary Publish job runs.

2.10.5.6 PARTITIONABLE types

A partitionable type (and its subtypes) has an additional attribute, i_partition. Since an object creates database entries in the tables for its type and all its supertypes, the whole type hierarchy must be either partitionable or non-partitionable. Predefined types are already partition enabled (or not), so only user-defined supertypes can be created as partitionable.

If you create a partitionable type, you must use the Administrative method PARTITION_OPERATION to take advantage of data partitioning.

2.10.5.7 SHAREABLE type

A shareable type is created for use with lightweight SysObjects. A shareable object acts as the parent to the lightweight SysObject. Creating a shareable type creates additional non-inherited properties, i_sharing_type, i_orig_parent, and allow_propagating_changes to the type, in addition to the properties explicitly specified, and marks it as a shareable type. The *OpenText Documentum Content Management - Server Fundamentals Guide (EDCCS250400-GGD)* contains more information about shareable and lightweight SysObjects.

2.10.5.8 LIGHTWEIGHT type

A lightweight object shares its SysObject properties with other lightweight objects, reducing storage requirements for the object. A shareable object acts as the parent to the lightweight SysObject, and contains the properties shared among the lightweight types. Creating a lightweight type creates the i_sharing_parent property, in addition to the custom properties explicitly specified, and marks it as a lightweight type. This property links the child lightweight object with its shareable parent. The *OpenText Documentum Content Management - Server Fundamentals Guide (EDCCS250400-GGD)* contains more information about shareable and lightweight SysObjects.

Note: The maximum length of light-weight type name for a repository that uses Oracle is 26.

2.10.5.9 Fulltext support for lightweight types

Use the FULLTEXT SUPPORT arguments to specify which properties are included in the fulltext index:

- FULLTEXT SUPPORT—Default, all lightweight attributes are indexed
- FULLTEXT SUPPORT NONE—No attributes are indexed
- FULLTEXT SUPPORT LITE ADD ALL—All lightweight attributes are indexed
- FULLTEXT SUPPORT LITE ADD *property_list*—Selected lightweight attributes are indexed only (cannot select parent attributes)
- FULLTEXT SUPPORT BASE ADD ALL—All lightweight and parent attributes are indexed
- FULLTEXT SUPPORT BASE ADD *property_list*—Selected lightweight or parent attributes are indexed

2.10.6 Property modifiers

Property modifiers define data dictionary information for a property. Defining property modifiers sets properties in data dictionary-related objects for the type. It does not set any dm_type or dm_type_info properties for the type.

2.10.6.1 update_modifier

An *update_modifier* specification can be:

```
SET property_name[[index]]=valueAPPEND property_name=valueINSERT
property_name[[index]]=valueREMOVE property_name[[index]]
TRUNCATE property_name[[index]]
```

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

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

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

  If the operation is adding a new value to the property, 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 properties because it automatically puts the new value at the end of the repeating property'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 property is single-valued, *value* can be NULL.

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

- 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

2.10.6.2 value_assistance_modifier

A value assistance modifier identifies one or more values for the property. Typically, value assistance is used to populate a pick list of values for the property 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 ([property_list])
```

You can include multiple ELSEIF clauses.

2.10.6.2.1 expression_string

expression_string defines a Boolean condition. If it returns TRUE, the server executes the associated *va_clause* to return values for the property. *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 routine 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 a property name.

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

2.10.6.2.2 va_clause

The *va_clause* provides the values for the property. It has two possible formats. One format provides a list of literal values. The second defines a query to return the values. Each format allows you to provide an estimate of the expected number of property values and indicate whether the values represent the complete list of allowed values for the property.

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

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

2.10.6.2.3 dependency clause

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

2.10.6.3 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 property called country in which each value represents a different country. If you display that property 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.

2.10.6.4 default_specification

A default specification provides a default value for a property. When a user creates a new instance of the type, Documentum CM Server automatically sets the property 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 properties.)

For a single-valued property, the syntax is:

```
DEFAULT[=]default_value
```

For a repeating property, 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 property and one repeating string property 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 property. For example, you cannot specify the keyword USER for a date property. You cannot specify NULL for a repeating property.

2.10.6.5 constraint_specification

A *constraint_specification* defines constraints for a property. For example, you can define a check constraint to provide data validation for the property.

You can define a `CHECK` constraint in a property modifier list:

If you define a check constraint in a property modifier list, only the single property for which the constraint is defined can be part of the constraint. If the constraint must include multiple properties, define it in the type modifier list.

For each constraint, you can define an error message to display if the constraint is violated.

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

The *OpenText Documentum Content Management - Server Fundamentals Guide (EDCCS250400-GGD)* contains the expanded information about the constraints and what they do.

2.10.6.5.1 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 property'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 a property 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 do not include a version label, the label CURRENT is assumed.

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(*property_name*)

  When the message is displayed, $value(*property_name*) is replaced with the value entered by the user in the field associated with the property. The property 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.

2.10.7 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 properties in the data dictionary objects for the type. No properties are set in the dm_type or dm_type_info objects for the type.

2.10.7.1 update_modifier

You can use the same update modifiers for types as for properties (refer to "Property modifiers" on page 87). 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.

2.10.7.2 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 properties 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.

2.10.7.3 constraint_specification

You can define a Check constraint in a type modifier list.

For each constraint, you can define an error message to display if the constraint is violated.

Constraints are defined in a type modifier list if the constraint includes multiple properties. If only a single property defines the constraint, the constraint is typically defined in the property modifier list for the property rather than the type modifier list.

The *OpenText Documentum Content Management - Server Fundamentals Guide (EDCCS250400-GGD)* contains more information about the constraints and their use.

2.10.7.3.1 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 property's value violates the constraint. Check constraints are defined at the type level when *expression_string* references two or more properties. 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 a property 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 do not include a version label, the label CURRENT is assumed.

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(*property_name*)

When the message is displayed, the server replaces $value(*property_name*) with the value entered by the user in the field associated with the property. The property 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.

2.10.7.4 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 properties 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.

2.10.7.5 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.

2.10.8 Related statements

"Alter Type" on page 50
"Drop Type" on page 102

2.10.9 Examples

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

```
CREATE TYPE "employee"
(
"emp_ssn" string(10)
(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
(
 NOT NULL,
 SET "label_text"='Department Code')
)
WITH SUPERTYPE NULL
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"
```

2.11 Delete

2.11.1 Purpose

Removes rows from a registered table.

2.11.2 Syntax

```
DELETE FROM table_name WHERE qualification
```

2.11.3 Arguments

Table 2-10: DELETE argument descriptions

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 “WHERE clause” on page 155 for a description of the valid forms of a qualification.

2.11.4 Return value

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

2.11.5 Permissions

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

- Your object-level permission for the dm_registered object in the repository 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 Documentum CM 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.

The *OpenText Documentum Content Management - Server Administration and Configuration Guide (EDCCS250400-AGD)* contains more information about security and object-level and table permissions.

2.11.6 Description

The DELETE statement deletes rows from a registered table. A registered table is a table in the underlying RDBMS that has been registered with Documentum CM Server. (Refer to "Register" on page 113 for information about registering tables.) All rows for which the WHERE clause qualification evaluates to TRUE are deleted.

2.11.7 Related statements

"Insert" on page 111
"Register" on page 113
"Select" on page 120
"Unregister" on page 167
"Update" on page 169

2.11.8 Examples

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 property of a document:

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

2.12 Delete...Object

2.12.1 Purpose

Deletes objects from the repository.

2.12.2 Syntax

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

2.12.3 Arguments

Table 2-11: DELETE...OBJECT argument descriptions

Variable	Description
type_name	Identifies the type of object to remove from the repository. 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 property references.
WITHIN PARTITION clause	Restricts the statement to objects in particular partitions. *partition_id* identifies the object partition. The property, i_partition, contains the partition value for an object.
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.

Variable	Description
SEARCH clause	Restricts the statement to objects that meet the SEARCH clause *fulltext search condition*. Refer to "SEARCH clause" on page 151 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 "WHERE clause" on page 155 for a description of the valid forms of a qualification.

2.12.4 Return value

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

2.12.5 Permissions

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

2.12.6 Description

This section contains information about using this statement.

2.12.6.1 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 property 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 property 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, WITHIN PARTITION, IN ASSEMBLY, SEARCH, and WHERE, restrict the statement's scope. The WITHIN PARTITION clause restricts the operation to particular partitions. 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.

2.12.6.2 IN ASSEMBLY clause

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

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

2.12.7 Related statements

"Change...Object" on page 69
"Create...Object" on page 76
"Update...Object" on page 171

2.12.8 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')
```

2.13 Drop Group

2.13.1 Purpose

Removes a group from the repository.

2.13.2 Syntax

```
DROP GROUP group_name
```

2.13.3 Syntax description

Table 2-12: DROP GROUP syntax description

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

2.13.4 Permissions

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

2.13.5 General notes

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

2.13.6 Related statements

"Alter Group" on page 44
"Create Group" on page 73

2.13.7 Example

This example drops the group called rons_baby_gift:

```
DROP GROUP rons_baby_gift
```

2.14 Drop Type

2.14.1 Purpose

Removes a user-defined object type from the repository.

2.14.2 Syntax

```
DROP TYPE type_name
```

2.14.3 Arguments

Table 2-13: DROP TYPE argument descriptions

Variable	Description
type_name	Identifies the object type to remove.

2.14.4 Permissions

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

2.14.5 Description

The type you identify must meet the following conditions:

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

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

- The IDfSession.publishDataDictionary method is executed for the entire repository.
- 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.

2.14.6 Related statements

“Alter Type” on page 50
“Create Type” on page 79

2.14.7 Examples

```
DROP TYPE "my_base_type"
```

2.15 Execute

2.15.1 Purpose

Executes administration methods.

2.15.2 Syntax

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

2.15.3 Arguments

Table 2-14: EXECUTE argument descriptions

Argument	Description
admin_method_name	Specifies which administration method you want to execute. "Administration methods by category for the EXECUTE statement" on page 105 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 "Administration methods" on page 181, for a description of each administration method and its arguments.

2.15.4 Return value

The EXECUTE statement returns a collection. The properties of the query result object in the collection depend on which administration method was executed. Refer to the description of each method in "Administration methods" on page 181, for details.

2.15.5 Permissions

The privileges required depend on the administration method you are executing. Refer to the description of the individual method in "Administration methods" on page 181, for information.

2.15.6 Description

The EXECUTE statement is the DQL equivalent of the IDfSession.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.

"Administration methods by category for the EXECUTE statement" on page 105, lists the administration methods that you can invoke with the EXECUTE statement and briefly describes the task that each performs.

Table 2-15: Administration methods by category for the EXECUTE statement

Category of operation	Function	Description
Process Management	"CHECK_SECURITY" on page 197	Checks a user or group's permissions level for one or more objects.
	"GET_INBOX" on page 232	Returns items in user's Inbox.
	"MARK_AS_ARCHIVED" on page 264	Sets the i_is_archived property of a dm_audittrail, dm_audittrail_acl, or dm_audittrail_group object to T.
	"PURGE_AUDIT" on page 310	Deletes audit trail entries from the repository.
	"RECOVER_AUTO_TASKS" on page 320	Recovers work items claimed by a workflow agent master session but not yet processed.
	"ROLES_FOR_USER" on page 332	Returns the roles assigned to a user in a particular client domain.
Execute procedures	"DO_METHOD" on page 207	Executes system-defined procedures such as lpq or who or user-defined procedures.
	"HTTP_POST" on page 240	Directs the execution of a method to an application server.
Content storage management	"CAN_FETCH" on page 188	Determines whether content in a distributed storage area component can be fetched by the server.
	"CHECK_RETENTION_EXPIRED" on page 193	Finds SysObjects in content-addressed storage that have an expired retention period or no retention period.
	"CLEAN_LINKS – Deprecated" on page 201 This method is deprecated. Foundation Java API 6.0 does not support linked storage areas or link record objects. Consequently, linked storage areas, link records and this supporting method are deprecated.	Provides maintenance for linked store storage areas. On Windows platforms, cleans up unneeded link record objects and resets file storage object security. On Linux platforms, cleans up unneeded linkrecord objects, directories, and links associated with linked storage areas.

Category of operation	Function	Description
	"DELETE_REPLICA" on page 204	Removes a replica from a distributed storage area.
	"DESTROY_CONTENT" on page 206	Removes a content object and its associated file from the repository. Do not use this for archiving; use PURGE_CONTENT instead.
	"GET_FILE_URL" on page 230	Returns the URL to a content file.
	"GET_PATH" on page 237	Returns the path to a particular content file in a particular distributed storage area component.
	"IMPORT_REPLICA" on page 245	Imports an external file as a replica of content already in the repository.
	"MIGRATE_CONTENT" on page 267	Moves content files from one storage area to another.
	"PURGE_CONTENT" on page 316	Deletes a content file from a storage area. Used as part of the archiving process.
	"PUSH_CONTENT_ATTRS" on page 317	Sets the content metadata in a content-addressed storage system for a document stored in that storage system.
	"REGISTER_ASSET" on page 321	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 OpenText™ Documentum™ Content Management Transformation Services - Media Media running.
	"REPLICATE" on page 327	Copies content in one component of a distributed storage area to another area.
	"RESTORE_CONTENT" on page 330	Moves a file or files from archived storage to the original storage location.

Category of operation	Function	Description
	"SET_CONTENT_ATTRS" on page 336	Sets the content_attr_name and content_attr_value properties in the content object associated with the content file.
	"SET_STORAGE_STATE" on page 343	Sets the state of a storage area to off-line, on-line, or read-only.
	"TRANSCODE_CONTENT" on page 347	Queues a request for a content transformation to Media Server. This method is only available or useful if you have OpenText Documentum Content Management (CM) Transformation Services - Media running.
Database Methods	"DB_STATS" on page 202	Provides database operation statistics for a session.
	"DROP_INDEX" on page 217	Drops an index.
	"EXEC_SQL" on page 220	Executes SQL statements.
	"EXPORT_TICKET_KEY" on page 222	Exports a repository's login ticket key.
	"FINISH_INDEX_MOVES" on page 223	Completes an interrupted object type index move operation.
	"GENERATE_PARTITION_SCHEME_SQL – Deprecated" on page 225	Creates an SQL script to partition a repository.
	"IMPORT_TICKET_KEY" on page 246	Imports a login ticket to the repository.
	"MAKE_INDEX" on page 260	Creates an object type index.
	"MOVE_INDEX" on page 297	Moves an object type index from one tablespace to another.
	"PARTITION_OPERATION" on page 299	Partitions a repository.
	"REORGANIZE_TABLE" on page 324	Reorganizes a database table for query performance.
	"RESET_TICKET_KEY" on page 329	Generates a login ticket key for the repository.

Category of operation	Function	Description
	"UPDATE_STATISTICS" on page 350	Updates the statistics for a database table.
Full-Text Methods	"ESTIMATE_SEARCH" on page 218	Returns the number of results matching a particular SEARCH condition.
	"MARK_FOR_RETRY" on page 265	Finds all queue items representing objects that failed indexing and marks them as pending indexing.
	"MODIFY_TRACE" on page 295	Sets the tracing level for full-text querying operations.
Session Management	"CHECK_CACHE_CONFIG" on page 190	Requests a consistency check on a particular cache config object.
	"GET_LAST_SQL" on page 236	Returns the last SQL statement issued.
	"GET_SESSION_DD_LOCALE" on page 239	Returns the locale in use for the current session.
	"LIST_AUTH_PLUGINS" on page 248	Lists the authentication plug-ins loaded by Documentum CM Server.
	"LIST_RESOURCES" on page 249	Provides information about the server operating system environment.
	"LIST_SESSIONS" on page 253	Provides information about current, active sessions.
	"LIST_TARGETS" on page 256	Lists the connection brokers defined as targets for the server. The information is returned in a collection with one result object whose properties list the connection brokers defined as targets for the server.
	"LOG_ON" on page 259 and "LOG_OFF" on page 258	Turn server logging of information about RPC calls on or off.
	"SET_APIDEADLOCK" on page 333	Sets a deadlock trigger on a particular API method or operation.
	"SET_OPTIONS" on page 340	Turn various tracing options on or off.

Category of operation	Function	Description
	"SHOW_SESSIONS" on page 345	Provides information about current, active sessions and a user-specified number of timed-out sessions.

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.

2.15.7 Examples

Refer to the individual descriptions of the method in "Administration methods" on page 181, for examples.

2.16 Grant

2.16.1 Purpose

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

2.16.2 Syntax

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

2.16.3 Arguments

Table 2-16: GRANT argument descriptions

Argument	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

Argument	Description
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" on page 110, for the use of a SELECT statement. The user name must be the value is found in the user_name property of the dm_user object associated with the user.

2.16.4 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 repository owner or a Superuser. Repository owners and Superusers cannot grant these privileges to themselves.

2.16.5 Description

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

2.16.6 Related statements

"Revoke" on page 118

2.16.7 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
```

2.17 Insert

2.17.1 Purpose

Inserts a row into a registered table.

2.17.2 Syntax

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

2.17.3 Arguments

Table 2-17: INSERT argument descriptions

Argument	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" on page 112 for more information.
dql_subselect	Specifies a SELECT statement. The returned values are inserted into the table.

2.17.4 Return value

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

2.17.5 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 Documentum CM 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.

The *OpenText Documentum Content Management - Server Administration and Configuration Guide (EDCCS250400-AGD)* contains more information about security and object-level and table permissions.

2.17.6 Description

Use the information in this section to execute this statement.

2.17.6.1 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.

2.17.6.2 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" on page 120 for the subselect statement's syntax.

2.17.7 Related statements

"Delete" on page 96
"Register" on page 113
"Select" on page 120
"Unregister" on page 167
"Update" on page 169

2.17.8 Examples

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')
```

2.18 Register

2.18.1 Purpose

Registers a table from the underlying RDBMS with the repository.

2.18.2 Syntax

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

2.18.3 Arguments

Table 2-18: REGISTER argument descriptions

Argument	Description
owner_name	Identifies the owner of the table. Use the owner's RDBMS user name. If the owner is a repository user, this value may be found in the user's user_db_name property. If the RDBMS is Oracle and the owner is the DBA, you can use the alias dm_dbo. If the RDBMS is SQL Server 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.
table_name	Identifies the RDBMS table to register with the repository. 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 and PostgreSQL, 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 SQL Server, if you specify a synonym, you must include the SYNONYM clause in the statement. For SQL Server, you cannot register a remote table. Refer to “The SYNONYM clause” on page 117 for details. If you do not have Superuser user privileges, you must be the owner of the table.

Argument	Description
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 tinyint smallint 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)).
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 SQL Server and you specify a synonym as the table name in the statement. Including the SYNONYM clause is optional if you are running against Oracle and PostgreSQL and specify a synonym as the table name in the statement. “The SYNONYM clause” on page 117 contains the details.

2.18.4 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 property, called new_object_ID, that contains the object ID of the dm_registered object for the table.

2.18.5 Permissions

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

2.18.6 Description

This section contains usability notes.

2.18.6.1 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 repository. 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 property of the server's server config object.

The statement also sets default table permissions. The table permissions are set to SELECT for the owner. The group and world are not given any default table permissions. You can change these by setting the properties directly. Note that table permissions for registered tables are not hierarchical. The *OpenText Documentum Content Management - Server Administration and Configuration Guide (EDCCS250400-AGD)* contains complete description of registered table permits.

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.

2.18.6.2 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 property of the table's dm_registered object.

When you run against Oracle, Documentum CM 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 property 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 SQL Server, Documentum CM Server substitutes the value you specified in the SYNONYM clause (recorded in the synonym_for property) 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"
```

Documentum CM Server substitutes the actual table name for the synonym and the following SQL is generated:

```
SELECT * FROM londonserver.londonremote.johndoe.myremotetable
```

2.18.6.2.1 Special note for Oracle platforms

If you create a database link between the database on which the OpenText Documentum CM repository is installed and another database, and then the use SYNONYM feature to obtain information from that second database, both databases must be in the same codepage.

2.18.7 Related statements

"Delete" on page 96
"Insert" on page 111
"Unregister" on page 167
"Update" on page 169

2.18.8 Examples

The following statement registers the RDBMS table named departments:

```
REGISTER TABLE "departments"
("dept_name" CHAR(30), "dept_code" INT)
KEY ("dept_code")
```

2.19 Revoke

2.19.1 Purpose

Removes one or more user privileges from one or more users.

2.19.2 Syntax

```
REVOKE privilege {,privilege} FROM users
```

2.19.3 Arguments

Table 2-19: REVOKE argument descriptions

Argument	Description
privilege	Specifies the privilege to revoke. Valid privileges are: SUPERUSER SYSADMIN CREATE TYPE CREATE CABINET CREATE GROUP CONFIG AUDIT PURGE AUDIT VIEW AUDIT

Argument	Description
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" on page 119 for the use of a SELECT statement. The user name must be the value is found in the user_name property of the dm_user object associated with the user.

2.19.4 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 repository owner or a Supersuser. Repository owners and Superusers cannot revoke these privileges from themselves.

2.19.5 Description

The statement succeeds even if a user does not have the privilege being revoked.

2.19.6 Related statements

"Grant" on page 109

2.19.7 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')
```

2.20 Select

2.20.1 Purpose

Retrieves information from the repository, the database, or both.

2.20.2 Syntax

A SELECT statement may be either a standard SELECT or an FTDQL SELECT statement. The syntax for an FTDQL SELECT statement is a subset of the standard syntax. For a brief description of an FTDQL SELECT statement, refer to "FTDQL SELECT statements" on page 126.

The syntax for a standard SELECT statement is:

```
SELECT [FOR base_permit_level][ALL|DISTINCT] value [AS name] {,value [AS name]}
FROM [PUBLIC] source_list[WITHIN PARTITION (partition_id{,partition_id})
| IN DOCUMENT clause
| IN ASSEMBLYclause]
[SEARCH [FIRST|LAST]fulltext_search_condition[IN FTINDEX index_name{,index_name}]
[WHERE qualification]
[GROUP BY value_list]
[HAVING qualification]
[UNION dql_subselect]
[ORDER BY value_list]
[ENABLE (hint_list)]
```

The syntax for an FTDQL SELECT statement is:

```
SELECT [FOR base_permit_level][ALL|DISTINCT] value [AS name] {,value [AS name]}
FROM [PUBLIC] source_list[SEARCH fulltext_search_condition[IN FTINDEX
index_name{,index_name}]
[WHERE qualification]
[ORDER BY SCORE]
[ENABLE (hint_list)]
```

2.20.3 Arguments

Table 2-20: SELECT argument descriptions

Argument	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. There are no FTDQL constraints on this value.

Argument	Description
value	Identifies the information to retrieve. Valid values are: • Property and column names • Scalar, aggregate, and date functions • MFILE_URL function • Arithmetic expressions • The keywords CONTAIN_ID, CONTENTID, DEPTH, ISCURRENT, ISPUBLIC, ISREPLICA, OBJTYPE, PARENT, SCORE, SUMMARY, SYSOBJ_ID, TEXT, 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. There are constraints on the values in the selected values list for FTDQL queries. For details, refer to the Usage Notes sections describing each kind of selected value.
AS *name*	Defines a name for the result object property 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" on page 142 for information about the default name. *name* must consist of ASCII characters You may include the AS *name* option in an FTDQL SELECT statement.

Argument	Description

source_list	Identifies the object types and RDBMS tables to search. You can specify any combination of object types, RDBMS tables, and inline views in a standard SELECT statement.
	
	Types are specified using the following syntax:
	
	`type_name [(ALL)|(DELETED)|(LITE)] [correlation_variable] [WITH passthrough_hint_list]`
	
	type_name cannot reference an object type that represents aspect properties.
	
	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'})`
	`( SELECT ... ) [correlation_variable]`
	
	Valid *db_keywords* are: ORACLE and SQL_SERVER. *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” on page 387 describes using passthrough hints fully.
	
	On Oracle, 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.
	
	Inline views are specified using the following syntax:
	
	An inline view is a subquery statement. In other words, it is a SELECT statement

Argument	Description
	enclosed in parenthesis. The results from the subquery act like a temporary table in the select list. An optional *correlation_variable* can be specified, just as with tables or types. For complete information about specifying the *source_list* clause, including LEFT OUTER JOIN, refer to “Source list” on page 143. For a description of the constraints imposed on the source list by an FTDQL query, refer to “FTDQL constraints on the source list” on page 147.
WITHIN PARTITION clause	Restricts the statement to objects in particular partitions. *partition_id* identifies the object partition. The property, i_partition, contains the partition value for an object.
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” on page 147 for the syntax and usage of the IN DOCUMENT clause. This clause may not be included in an FTDQL SELECT statement.
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. This clause may not be included in an FTDQL SELECT statement.
fulltext_search_condition	Defines criteria for searching of the full-text index. Refer to “SEARCH clause” on page 151 for a description of its syntax and use. You can include a fulltext search condition in an FTDQL SELECT statement.

Argument	Description
qualification	Restricts returned results to objects meeting the conditions in the qualification. Refer to "WHERE clause" on page 155 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. A *dql_subselect* may not be included in an FTDQL SELECT statement.
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 "GROUP BY clause" on page 159 for the specific use of this in each clause. A GROUP BY clause may not be included in an FTDQL SELECT statement.

Argument	Description
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: CONVERT_FOLDER_LIST_TO_ORFETCH_ALL_RESULTS N FORCE_ORDER FTDQL and NOFTDQL FT_CONTAIN_FRAGMENT GROUP_LIST_LIMIT N HIDE_SHARED_PARENT IN and EXISTS OPTIMIZE_ON_BASE_TABLE OPTIMIZE_TOP N RETURN_RANGE RETURN_TOP N ROW_BASED SQL_DEF_RESULT_SET N TRY_FTDQL_FIRSTUNCOMMITTED_READ DM_LEFT_OUTER_JOIN_FOR_ACL N is an integer value and level_1 and level_2 are optimization levels. For a brief description of the standard hints, refer to "DQL standard hints" on page 162. The ROW_BASED hint affects the selected values list and the WHERE clause qualification. For information about its effects, refer to the descriptions of those portions of the syntax. Information about ROW_BASED is also found in Appendix A, Using DQL hints on page 373, which contains full information about all the standard hints. ROW_BASED may not be included in an FTDQL query, nor may it be used in queries that reference a lightweight object type in the FROM clause. All other hints are acceptable in FTDQL queries or in queries against a lightweight object type.

2.20.4 Description

This section contains usability information for the Select statement.

2.20.4.1 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 standalone 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" on page 73.

2.20.4.2 FTDQL SELECT statements

An FTDQL SELECT statement is a SELECT statement whose syntax conforms to a particular set of rules that allow the query to be executed directly against the fulltext index. If the statement conforms to FTDQL syntax, the statement is run solely against the fulltext index; the repository is not queried.

This feature provides enhanced performance for the query. The general syntax accepted for an FTDQL query is shown in the formal syntax description. The syntax constraints on particular clauses enforced for an FTDQL query are listed in the Usage Notes sections describing each clause. A summary of the rules is found in "Summary of FTDQL query rules" on page 423.

An FTDQL SELECT statement must be a standalone statement. That is, SELECT statements embedded as a subselect in another DQL statement are never executed as FTDQL SELECT statements. Nor are unioned SELECT statements executed as FTDQL SELECT statements. All FTDQL SELECT statements must include one of the following:

- A SEARCH clause
- The keyword SCORE, SUMMARY, or TEXT in the list of selected values
- The DQL hint ENABLE(FTDQL)

A query that conforms to the FTDQL syntax rules is automatically executed as an FTDQL query. If you are wondering whether a particular query conforms to the rules, you can include the ENABLE(FTDQL) hint in the query. If the query conforms, the hint has no effect and the query is executed as an FTDQL query. If the query does not conform, Documentum CM Server returns an error.

If you do not want a particular query to execute as an FTDQL query even though it conforms to the FTDQL syntax, include the ENABLE(NOFTDQL) in the query.

A standard query queries both the fulltext index and the database. Such a query typically contains a SEARCH clause and a WHERE clause. If the WHERE clause is not compliant with the rules of FTDQL or the query contains an explicit ENABLE(NOFTDQL), Documentum CM Server executes the SEARCH clause against the fulltext index and the WHERE clause against the database and then returns the intersection of the results.

Note: A query that does not conform to the FTDQL syntax and does not include the ENABLE(FTDQL) hint is executed as a standard SELECT query. It does not return an error.

If an FTDQL query contains references to aspect properties, those properties must be indexed. Aspect properties are not indexed by default. You must explicitly declare them for indexing. Use Documentum Application Builder or an ALTER ASPECT statement to do so.

FTDQL metadata queries that contain LIKE predicates with pattern matching or on metadata that contains underscores in the values may return different results than non-FTDQL queries on the same metadata. This occurs because the index server processes FTDQL queries against metadata and the database server processes non-FTDQL queries against metadata.

FTDQL queries are not affected by the distinct_query_results key in the server.ini. Setting this key to T (TRUE) does not affect how an FTDQL query is processed.

2.20.4.3 Referencing non-qualifiable properties

Non-qualifiable properties (those stored in a property bag) can be referenced in SELECT statements. They may appear in the selected values list. However, they may not be referenced in an expression. Nor may they appear in a WHERE clause unless the query is an FTDQL query.

2.20.4.4 Referencing aspect properties

To reference an aspect property in a SELECT statement, you must qualify the property name with the name of the aspect for which it is defined. For example, suppose you have an aspect named grant_validation, with a property defined for it named grant_amount. To select the grant amount, reference the property as follows in the selected values list:

```
grant_validation.grant_amount
```

For example:

```
SELECT grant_validation.grant_amount FROM education_grants
....
```

If the query has multiple object types in the FROM clause, any aspect properties referenced in the selected values must be also qualified with the type name. For example, assuming that grant_validation and salary_calc are names of aspects:

```
SELECT education_grants.grant_validation.grant_amount,
dm_user.research_personnel.salary_calc
FROM education_grants, dm_user
...
```

The query result column that contains the selected aspect property value is named with the fully qualified name as specified in the selected values list.

2.20.4.5 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.

2.20.5 Selecting property values

Select object property values by specifying the property's name as a selected value. The properties you specify must belong to an object type identified in the FROM clause. The properties can be either single-valued or repeating properties. “Repeating properties” on page 130, contains guidelines for selecting repeating property values.

You cannot select any of the following properties if you are querying audit trail entries unless you have Superuser or View_audit privileges:

• acl_name	• object_name
• acl_domain	• object_type
• property_list	• owner_name
• property_list_id	• session_id
• chronicle_id	• version_label

The following statement returns the value of title, a single-valued property, for all documents in the repository:

```
SELECT "title" FROM "dm_document"
```

This next example returns the value of authors, a repeating property, from all documents that have bread as their subject:

```
SELECT "authors" FROM "dm_document"
WHERE "subject"='bread'
```

You can use an asterisk (*) to select a predefined set of system-defined properties for the object. Refer to “The asterisk (*) as a selected value” on page 140 for details.

2.20.5.1 Selecting non-qualifiable properties

Non-qualifiable properties can be referenced in the selected values list. However, you can reference them only as an individual selected value, by name. They cannot be referenced in expressions or functions, such as an aggregate function, in the selected values list.

2.20.5.2 Selecting aspect properties

You can select values of properties defined for aspects attached to objects. If the properties are repeating properties, there are some constraints on the query. For a listing of these, refer to "Repeating properties" on page 130.

2.20.5.3 Repeating properties

Including a repeating property in the selected values list is subject to the following constraints:

- You cannot select a repeating property and the name of a column from a registered table unless you also include the DQL hint, ROW_BASED.

  Note: The ROW_BASED hint may not be included in FTDQL queries or queries that reference a lightweight object type in the FROM clause.

- You cannot select a repeating property if there are multiple object types identified in the FROM clause unless you also include the DQL hint, ROW_BASED.

  Note: The ROW_BASED hint may not be included in FTDQL queries or queries that reference a lightweight object type in the FROM clause.

- If the repeating property is an aspect property or from a lightweight object or its shared parent, the following constraints also apply:
  - If the property is used in an expression, all properties referenced in the expression must be repeating properties from the same aspect or the same lightweight object type.
  - If the property is referenced in an aggregate function, you must include a GROUP BY clause that references r_object_id and only r_object_id.
  - The query must include r_object_id in the selected values list.
  - The query may not be a subquery.

You can select both repeating and single-valued properties in the same statement. If you do, the format of the returned results varies depending on how the query is written. The results can be returned with a separate result row for each value returned for a selected repeating property or the result rows can be returned in a master-detail format, with each row representing the values, including all repeating values, for one object.

2.20.5.3.1 To return results by object ID

To obtain query result objects that include all selected repeating property values for one object in one query result object, include the r_object_id property in the selected values list. Additionally, the source list in the FROM clause may contain only item.

To ensure that the results are ordered in either ascending or descending order, you can include an ORDER BY clause also. Typically, results are returned in ascending order by default.

Note: Refer to "ORDER BY clause" on page 161 for more information about using the ORDER By clause.

For example, the following statement returns one result object that contains the names of all three authors in the authors property:

```
SELECT "r_object_id","object_name","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

2.20.5.3.2 To return each repeating value in an individual result object

To obtain result objects that contain one repeating property value in each object, include the DQL hint, ROW_BASED, in the query.

Note: The ROW_BASED hint may not be included in FTDQL queries or queries that reference a lightweight object type in the FROM clause.

For example, suppose you want to obtain the object ID and authors of a document whose object_name is French Bread, and you want the results in a separate row for each author. The following query that includes the ROW_BASED hint returns a separate row for each returned value for the authors property:

```
SELECT "r_object_id","authors" FROM dm_document
WHERE "title" = 'Breads of France'
ENABLE(ROW_BASED)
```

Assuming that there are three authors, named Jean, Connie, and Corwin, the returned rows are:

r_object_id	authors
0900000334fa21c3	Jean
0900000334fa21c3	Connie
0900000334fa21c3	Corwin

Additionally, if the query has a WHERE clause qualification that references a value in the selected repeating property, then the query will return only the row that contains that value. For example, suppose you execute the following statement:

```
SELECT "r_object_id","title","authors" FROM dm_document
WHERE "authors" = 'JPierpont'
ENABLE(ROW_BASED)
```

The query returns one row:

r_object_id	title	authors
0900000334fa21c3	English Muffins	JPierpont

2.20.5.4 FTDQL requirements

The requirements for single or repeating properties in a selected values list for an FTDQL query are the same as those for a standard query.

2.20.6 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 Documentum CM 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.

2.20.6.1 FTDQL requirements

The requirements for column values in a selected values list for an FTDQL query are the same as those for a standard query.

2.20.7 Scalar, aggregate, and date functions as selected values

You can include scalar, aggregate, and date 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'
```

If an aggregate function references a repeating property from a lightweight object or its shared parent or from an aspect, the query must also have r_object_id in the selected values list and must include a GROUP BY clause that references r_object_id and only r_object_id.

An aggregate function cannot reference a non-qualifiable property.

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. The *OpenText Documentum Content Management - Server Fundamentals Guide (EDCCS250400-GGD)* contains more 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. “IN ASSEMBLY clause” on page 150 contains more information.

A date function manipulates a date in some manner. To use a date function in the selected values list, use the AS clause to provide an alias for the returned value in the result object. For example:

```
SELECT DATETOSTRING("r_modify_date",'yy-mm-dd") AS ModifyDate
FROM "dm_document"
```

It is recommended that you use the AS clause to alias a return value for any function referenced in the selected values list. “The AS clause” on page 142, has information about the AS clause.

2.20.7.1 FTDQL requirements

You can include scalar, aggregate, and date functions in the selected values list of an FTDQL query.

2.20.8 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 “MFILE_URL function” on page 28.

You can include the MFILE_URL function in the selected values list of an FTDQL query.

2.20.9 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:

```
property A + property B
column C * column D
(2 * column B) - property F
property 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'
```

The expression cannot reference a non-qualifiable property.

2.20.9.1 FTDQL requirements

Arithmetic expressions are not allowed in the selected values list in an FTDQL query.

2.20.10 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 property 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 is not 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, the returned values are not useful unless a SEARCH clause is included.

2.20.10.1 Full-Text keywords

The following keywords return information about full-text indexes.

- CONTENTID

  The CONTENTID keyword returns the object ID of the content object representing the content file that matches the select criteria. A content object links a content file to all documents that contain the file.

- 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 property 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.

- SCORE

  The SCORE keyword returns a document's relevance ranking as determined by the index server. If an ORDER BY clause is not included in the query, the returned document order is by descending score.

- SYSOBJ_ID

  The SYSOBJ_ID keyword returns the object ID of the objects returned with a fulltext search. It is the object ID stored in the fulltext index for the object.

- SUMMARY

  The SUMMARY keyword returns a summary of each document returned by the SELECT statement. The summary is up to 4096 bytes from the document, chosen by the Full-Text Engine. 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'
```

- TEXT

  The TEXT keyword returns the variant text forms to which a specified term was match, to return a particular document. The values are returned as a repeating property. For example, if the search string specifies “talk”, the index server may return documents that contain the words “talking” or “talked”. If so, then TEXT would return those forms of the word “talk”.

2.20.10.1.1 FTDQL requirements

All the keywords classified as fulltext keywords can be included in an FTDQL query.

2.20.10.2 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 an IN DOCUMENT clause with the USING ASSEMBLIES option, 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 Figure 2-1:

```
SELECT "r_object_id", DEPTH FROM "dm_document"
IN DOCUMENT ID('object_id_A') DESCEND
```

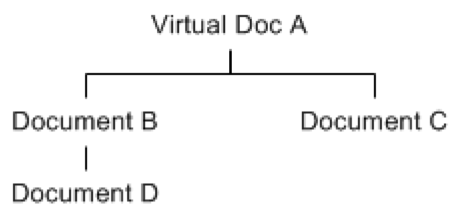

Figure 2-1: Sample virtual document

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 Figure 2-1:

  ```
  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.

2.20.10.2.1 FTDQL requirements

You may not use any of the keywords classified as virtual document keywords in an FTDQL query.

2.20.10.3 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 repository that has been placed in the repository as a copy of an object from another repository. 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.

- USER

  The USER keyword returns the current user's user name.

- 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 property that points to the thumbnail file.

  – A store property that identifies the storage area containing the thumbnail.

    If the storage area's require_ticket property 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.

  – A parameter MAX_URL_LEN can be specified in dm_docbase_config object (for r_module_name/r_module_mode) that indicates the maximum length of the URL for thumbnail_url and mfile_url keywords if the URL increases beyond the default length. If specified, you must restart the Documentum CM Server.

Note: Set the r_module_mode value for MAX_URL_LEN. The maximum allowed length is 1024.

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.

In some applications, there may be multiple thumbnails generated for a single content object. For example, thumbnails with different sizes can be associated with a single content object. In those cases, you may want to use the MFILE_URL function to retrieve the URL. Typically, the page_modifier property identifies the different thumbnails, so you would use that property to select the desired thumbnail. See "MFILE_URL function" on page 28, for more information.

However, another mechanism is available to affect the query issued to find thumbnail content when the THUMBNAIL_URL DQL keyword is used. The following server.ini settings should be used to control what content is returned by the keyword. They require knowledge of the underlying SQL query issued by the Documentum CM Server to use correctly. The server.ini settings are:

- thumbnail_url_where_clause_extra—specifies an additional predicate that is added to the where clause of the query
- thumbnail_url_order_by_clause—specifies the order by clause of the query

From the 6.7 release, the following query is used to find thumbnail content:

```
select b.parent_id,storage_id, format, data_ticket
from dmr_content_s a, dmr_content_r b
where a.r_object_id = b.r_object_id and
b.parent_id in (selected_id_list) and page = 0 and
storage_id in (list_of_thumbnail_stores)
custom_predicate
order by order_by_clause
```

In the query:

- *selected_id_list* is the list of object IDs for which we are trying to find thumbnail content
- *list_of_thumbnail_stores* is the list of storage IDs where thumbnail content can be stored
- *custom_predicate* is:
  - nothing—if the thumbnail_url_where_clause_extra server.ini parameter is not set
  - and (*string*)—where *string* is the value of the parameter

- *order_by_clause* is:
  - the string “full_format desc”, the default value, if the thumbnail_url_order_by_clause server.ini parameter is not set
  - the string specified in the parameter, if set

So, for example, to configure the query to return only objects that have the format `jpeg_th` and have the page_modifier `medium_jpeg_th`, the following line should be added to server.ini:

```
thumbnail_url_where_clause_extra =
   "exists (select x.r_object_id from dmr_content_r x where
       x.r_object_id=a.r_object_id and x.page_modifier='medium_jpeg_th')"
```

The thumbnail_url_order_by_clause is blank in server.ini for this example.

When that line is added to the server.ini, subsequent queries to select the thumbnail content would be modified to:

```
select b.parent_id,storage_id, format, data_ticket
from dmr_content_s a, dmr_content_r b
where a.r_object_id = b.r_object_id and
b.parent_id in (selected_id_list) and page = 0 and
storage_id in (list_of_thumbnail_stores)
and (exists (select x.r_object_id from dmr_content_r x where
x.r_object_id=a.r_object_id and x.page_modifier='medium_jpeg_th'))
order by full_format desc
```

To validate the settings used to modify the behavior of thumbnail_url, use the VALIDATE_THUMBNAIL_URL_SETTINGS administrative method to check the settings. If the settings are invalid, the method returns a detailed error message explaining the problem. This information can be used to fix the settings in server.ini. In DQL, run the following query to execute this method:

```
EXECUTE VALIDATE_THUMBNAIL_URL_SETTINGS
```

2.20.10.3.1 FTDQL requirements

The THUMBNAIL_URL keyword is the only keyword in the miscellaneous category that is acceptable in an FTDQL query. You may not use the IS_REPLICA or USER keywords in an FTDQL query.

2.20.11 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 lightweight object types:

- If the LITE keyword is included in the query, the asterisk returns all single-valued properties of the lightweight object.
- If the LITE keyword is not included in the query, the asterisk returns all single-valued properties of both the lightweight object and its shared parent.

For non-lightweight object types, what is returned by an asterisk depends on whether or not the ROW_BASED hint is included in the query. If the hint is not included, an asterisk returns values only for the first object type listed in the FROM clause. The values returned depend on the object type:

- For objects that are subtypes of persistent object type, the asterisk returns:
  - All read/write single-valued properties
  - The r_object_id property
- If the object type is SysObject or a SysObject subtype, in addition to the properties returned for a persistent object, the asterisk also returns:
  - r_object_type
  - r_creation_date
  - r_modify_date
  - a_content_type

If the query includes the ROW_BASED hint, the asterisk returns the values of all properties of all object types in the FROM clause. For example, the following query returns all property values from both object types for returned objects:

```
SELECT * FROM dm_user,dm_group
```

If you want to limit the returned values to only the property values for a particular type in the FROM clause, qualify the asterisk with the type. For example:

```
SELECT a.*, b.r_object_id FROM dm_document a, mydoc b
WHERE a.subject = b.title ENABLE(ROW_BASED)
```

In the above example, the asterisk returns the values only for properties from dm_document objects. You can use an asterisk in this manner only when the ROW_BASED hint is included in the query.

An asterisk does not return values of properties defined for any aspects attached to an object.

2.20.11.1 FTDQL requirements

You may not use an asterisk as a selected value in an FTDQL query.

2.20.12 The AS clause

The AS clause lets you name the properties 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 property that contains the returned a + b values.

When the statement returns the value of a property or column name, providing a name for the query result property is optional. The system assigns the property 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 SQL Server, and OpenText recommends that you also do so if you are running against Oracle and PostgreSQL.

SQL Server does 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.

Because the server does not allow duplicate property names in an object, if you assign the same name to more than one property, the system ignores all subsequent occurrences of the name and assigns those properties 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 AS clause is applicable for PostgreSQL also.

2.20.12.1 FTDQL requirements

You may use the AS clause in an FTDQL query.

2.20.13 The FROM clause

The FROM clause defines which items are searched. Both the PUBLIC keyword and the source list restrict the search.

2.20.13.1 PUBLIC keyword

The PUBLIC keyword restricts the search to objects for which the r_is_public property 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.

You may include PUBLIC in an FTDQL query.

Including PUBLIC increases the search performance.

2.20.13.2 Source list

A source list defines which object types and which RDBMS tables to search. The source list for a standard query can include:

- One or more object types, to a maximum of 20 types. In releases prior to 6.6, the maximum is 10.

  Any object type except those internal object types that represent aspect properties may be specified.
- One or more RDBMS table names
- One or more inline views

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 properties, 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” on page 387.

2.20.13.2.1 Source list JOIN operations

If more than one element is specified in the source list, separating each element by a comma causes the types, tables, or inline views to be combined in an implied JOIN operation. However, a LEFT OUTER JOIN clause can also be used. Use the following syntax:

```
FROM table | type | inline_view {,table | type | inline_view}
[LEFT [OUTER] JOIN type|table [correlation_variable] ON Boolean_condition
[LEFT [OUTER] JOIN type|table [correlation_variable] ON Boolean_condition]]
```

In the syntax above, you must supply at least one table, type, or inline view after FROM, and can include additional comma-separated tables, types, or inline views. An implied JOIN will join those comma-separated elements. Following the comma-separated list of tables, types, and inline views, you can optionally specify a LEFT OUTER JOIN clause. In that clause, you must specify one type or table and a Boolean condition. You cannot use an inline view as the right-hand element in the LEFT OUTER JOIN clause. One additional optional LEFT OUTER JOIN clause can follow the first one. The *Boolean_condition* specifies what property to use to specify the JOIN.

Unlike the standard SQL behavior, a LEFT OUTER JOIN will behave like a simple JOIN, if the *Boolean_condition* clause uses a repeating property from the right-hand element of the LEFT OUTER JOIN.

Additionally:

- If any repeating attribute appears in the ON clause, a ROW_BASED hint must be used.
- If any repeating attributes are used in a select list or ON clause, only matching rows are returned. The server adds the dm_repeating1_1.r_object_id=<type_name_1>.r_object_id predicate in the WHERE clause which translates LEFT OUTER JOIN to an INNER JOIN.
- If there are aspect attributes in the select list, no type join can be performed, including LEFT OUTER JOIN.

2.20.13.2.2 Including type names in the source list

Use the following syntax to specify a type name:

```
type_name [(ALL)|(DELETED)|(LITE)] [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 property is set to TRUE if you delete the object and it is the root version of a version tree. The keyword LITE directs the server to search only the properties in the lightweight type, otherwise the server also searches the properties in the shareable parent and the supertype(s). If you specify LITE, then you must only specify properties in the lightweight type. You must enclose ALL, DELETED, and LITE 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 property references in the query.

Any user can specify any object type in a query with two exceptions. The first exception is the type dmi_audittrail_attrs. To specify that type, you must have either Superuser or View Audit privileges. The second exception are the object types representing aspect properties. These types cannot be referenced in the FROM clause by any user.

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. The *OpenText Documentum Content Management - Server Administration and Configuration Guide (EDCCS250400-AGD)* contains full description of server security.

Including multiple object types is subject to the following constraints:

- The selected values list cannot contain repeating properties unless the ROW_BASED hint is included in the query.

  Note: The ROW_BASED hint may not be included in FTDQL queries or queries that reference a lightweight object type in the FROM clause.

- If the selected values list contains an asterisk (*), only the first type's properties will be expanded unless the ROW_BASED hint is included in the query.

  "The asterisk (*) as a selected value" on page 140, contains details about including an asterisk in the selected values list.

  Note: The ROW_BASED hint may not be included in FTDQL queries or queries that reference a lightweight object type in the FROM clause.

- The query cannot include a SEARCH clause.
- Unqualified property 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.

2.20.13.2.3 Including table names in the source list

You can include one or more RDBMS table names in the FROM clause of a standard query. 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.

2.20.13.2.4 Including inline views in the source list

You can include one or more inline view in the FROM clause of a standard query, just like a type or registered table. The syntax is:

```
( SELECT ... ) [correlation_variable]
```

The parenthesis are required, as an inline view is the result from a subquery. The *correlation_variable* is optional, and is used to refer to the view in other sections of a query statement.

2.20.13.2.5 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
```

2.20.13.2.6 FTDQL constraints on the source list

Observe the following rules if you want to execute the query as an FTDQL query:

- You can only reference one object type and that type must be dm_sysobject or a dm_sysobject subtype.
- You cannot reference a registered table in the source list.

2.20.14 The IN DOCUMENT clause

Note: You may not include an IN DOCUMENT clause in an FTDQL SELECT statement.

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 repository with the virtual document.

  If a component is a folder that doesn't reside in the current repository, the query returns the object ID of the folder's mirror object in the repository, 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 property and the statement contains an IN DOCUMENT clause, the server automatically removes any duplicates of the repeating property.

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 property {,property} [ASC|DESC]]]
```

2.20.14.1 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 property 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.

2.20.14.2 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.

2.20.14.3 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. The *OpenText Documentum Content Management - Server Fundamentals Guide (EDCCS250400-GGD)* contains more information about 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.

2.20.14.4 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, qualifiable property names, 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. The *OpenText Documentum Content Management - Server Fundamentals Guide (EDCCS250400-GGD)* contains more information about late binding.

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.

2.20.14.5 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 properties. The server includes the first version.

The syntax of the NODESORT BY option is:

```
NODESORT BY property {,property} [ASC|DESC]
```

where *property* is any qualifiable property 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 properties specified in the NODESORT BY option.

2.20.15 IN ASSEMBLY clause

Note: You may not include the IN ASSEMBLY clause in an FTDQL SELECT statement.

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.

2.20.16 SEARCH clause

The SEARCH clause identifies a subset of the full-text indexed documents. When you include a SEARCH clause, Documentum CM Server submits the query to the index server, which returns all objects that meet the SEARCH clause condition. The index server searches both content files and indexed metadata (property values) for matches with the specified condition.

The syntax of the SEARCH clause for a standard query is:

```
SEARCH [FIRST|LAST]DOCUMENT CONTAINS [NOT]search_string[IN FTINDEX index_name
{,index_name}]
```

The *search_string* syntax is described in “SEARCH DOCUMENT CONTAINS” on page 152.

2.20.16.1 FIRST and LAST keywords

The FIRST and LAST keywords determine when the search is conducted. These keywords are only valid in standard SELECT queries. You may not include them if the query is an FTDQL query.

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 repository 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.

2.20.16.2 SEARCH DOCUMENT CONTAINS

The SEARCH DOCUMENT CONTAINS syntax is the way to specify a fulltext search condition in a SEARCH clause. Including a list of words or phrases in a SEARCH DOCUMENT CONTAINS clause returns those documents whose content contains one, several or all of the specified words or phrases depending on the syntax you specify. You can also filter out the documents that contain the specified words or phrases. The documents are returned in order of importance.

The syntax is:

```
SEARCH [FIRST|LAST]DOCUMENT CONTAINS [NOT] search_string
```

where *search_string* has the following syntax:

```
[NOT]'word' {AND|OR [NOT] 'word'}
```

word may be a word, a phrase. It may also be a list of words or phrases, or both, separated by spaces and quoted appropriately. For more information about this syntax, refer to “Specifying words and phrases in SEARCH DOCUMENT CONTAINS” on page 153.

Warning

The complex syntax using logical operators is not recommended as future support may be dropped for this syntax and replaced with a different syntax. However, this syntax is currently supported.

2.20.16.2.1 Specifying words and phrases in SEARCH DOCUMENT CONTAINS

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'
```

The SEARCH DOCUMENT CONTAINS clause uses AND as the default logic operator for search items. For example, the following query returns documents that have both yeast and cake in their content or metadata:

```
SELECT * FROM "dm_documents"
SEARCH DOCUMENT CONTAINS 'yeast cake'
```

To search for any word in a list of search items, you have to explicitly use the OR operator. For example, the following query returns documents that have yeast or cake or both words in their content or metadata:

```
SELECT * FROM "dm_documents"
SEARCH DOCUMENT CONTAINS 'yeast' OR 'cake'
```

If you want to include a single quote in the search string, you must escape the quote with another single quote. For example:

```
SELECT * FROM "dm_documents"
SEARCH DOCUMENT CONTAINS 'O''Malley'
```

To search on a phrase, enclose the phrase in double quotes inside the quoted search string. For example, the following query returns all documents that include the phrase “blend butter and sugar”:

```
SELECT * FROM "dm_documents"
SEARCH DOCUMENT CONTAINS '"blend butter and sugar"'
```

Note: You can use single quotes to enclose a phrase, but if you do so, you must escape the quotes with single quotes.

You can combine words and phrases in the same search string. For example, the following query returns documents that contain the word “yeast” or the phrase “blend butter and sugar” or both:

```
SELECT * FROM "dm_documents"
SEARCH DOCUMENT CONTAINS 'yeast "blend butter and sugar"'
```

You can filter out the documents that contain the specified words or phrases by using the NOT operator. The following example searches for documents that contain “yeast” but not “butter”:

```
SELECT * FROM "dm_documents"
SEARCH DOCUMENT CONTAINS 'yeast' AND NOT 'butter'
```

2.20.16.2.2 Case sensitivity

All searches conducted by the index server are case insensitive.

For example, suppose you issue the following query:

```
SELECT owner_name,r_creation_date FROM dm_document
SEARCH DOCUMENT CONTAINS 'budget'
```

The query returns all documents that contain, either in content or metadata, the word budget in lowercase or uppercase or in any combination (Budget, buDget, budgeT, and so forth).

2.20.16.2.3 Accent and diacritical marks

Some languages use accents and diacritical marks on some characters or syllables in words. Searches are insensitive to accent and diacritical marks. When you search on a word or phrase, the search returns all objects that contain the word or phrase, even if some matches also contain an accent or diacritical mark. Similarly, when you search on a word or phrase that contains such marks, the search ignores the marks and returns all objects that contain the word or phrase, spelled with or without the accent or diacritical mark.

For example, suppose you issue the following query:

```
SELECT owner_name,r_creation_date FROM dm_document
SEARCH DOCUMENT CONTAINS 'cote'
```

The query returns all documents that contain, in metadata or content, the word cote, including those with instances of the word with accents or diacritical marks (côte, côté, and so forth).

Now, suppose you issue the following query that specifies a search term that includes an accent:

```
SELECT owner_name,r_creation_date FROM dm_document
SEARCH DOCUMENT CONTAINS 'coté'
```

That query also returns all documents that contain, in metadata or content, the word cote, including those with instances of the word with accents or diacritical marks (côte, côté, and so forth).

2.20.16.2.4 Asterisk as wildcard character

You can use the asterisk (*) as a wildcard character in a search string. The asterisk matches any string of letters. You can use it in any position in the string. For example, foo* matches any word that begins with “foo”. The string *foo* matches any word that contains “foo” in its sequence of letters. The string faa*foo matches any word that begins with faa and ends with foo.

You may also use the asterisk as a wildcard in a phrase search. For example, suppose you provide the following search string:

```
'"n* is th* time*"'
```

That string matches phrases such as “now is the time”, “nothing is thoroughly timed”, and “November is thinking time”.

2.20.17 WHERE clause

Use the WHERE clause to restrict which 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, qualifiable property names, column names, scalar and date functions, arithmetic expressions, predicates, and full-text keywords. Multiple expressions are joined together using the logical operators. “Documentum Query Language language elements“ on page 7, contains descriptions of all accepted comparison and arithmetic operators, predicates, functions, and logical operators. Full-text keywords are described in “Full-Text keywords” on page 135.

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)
```

The qualification can be simple or 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
```

The next example references an aspect property. Note that the property name is fully qualified with the name of the aspect. All references to aspect properties must be qualified with the aspect name.

```
SELECT "r_object_id", "title" FROM "dm_document"
WHERE ANY AUTHOR IN ("JohnDoe") and grant_validation.grant_amount>100000
```

2.20.17.1 General constraint on the qualification

A qualification cannot reference any of the following properties of audit trail objects unless you have Superuser or View_audit privileges:

• acl_name	• object_name
• acl_domain	• object_type
• property_list	• owner_name
• property_list_id	• session_id
• chronicle_id	• version_label

2.20.17.2 Using repeating properties in qualifications

Referencing repeating properties in a qualification is supported. However, there are some rules and guidelines. This section discusses those rules and guidelines.

2.20.17.2.1 ANY keyword use

If an expression references a repeating property, you must include the ANY keyword in the expression unless the ROW_BASED hint is included in the query. For example, the following query includes the ANY keyword because it does not include ROW_BASED:

```
SELECT "r_object_id","title","subject" FROM "dm_document"
WHERE ANY "authors" IN ('gillian')
```

If ROW_BASED is included, the query may be written without the ANY predicate:

```
SELECT "r_object_id","title","subject" FROM "dm_document"
WHERE "authors" IN ('gillian') ENABLE(ROW_BASED)
```

Note: The ROW_BASED hint may not be included in FTDQL queries or queries that reference a lightweight object type in the FROM clause.

If you are referencing a repeating property 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” on page 377 contains an example that shows the differences between the options in the generated query.

2.20.17.2.2 Referencing repeating properties in compound expressions

A compound expression is multiple expressions linked by logical operators. For example, the following is a compound expression:

```
object_name = 'book_proposal' AND owner_name = 'john doe'
```

If two or more expressions in a compound expression reference repeating properties, the properties must be from the same object type unless the ROW_BASED hint is included in the query. For example, the following query contains a compound expression that has two expressions referencing repeating properties from dm_document:

```
SELECT r_object_id, object_name FROM dm_document

WHERE ANY authors IN ('joe','john') AND ANY keywords LIKE 'eng%'
```

However, suppose you want to select from two object types and use a compound expression that referenced repeating properties from both types. To do that, you must include the ROW_BASED hint:

```
SELECT a.r_object_id, b.i_acceptance_date
FROM dm_document a, dmi_package b
WHERE a.authors IN ('john doe') AND b.r_package_label = 'APPROVED' ENABLE(ROW_BASED)
```

Without the ROW_BASED hint, the query would fail with an error stating that the two repeating properties it references, authors and r_package_label, are not from the same type.

Additionally, when ROW_BASED is included, it is not necessary to include the ANY predicate. Including ANY does not generate an error in such cases, but it is not necessary.

Note: The ROW_BASED hint may not be included in FTDQL queries or queries that reference a lightweight object type in the FROM clause.

2.20.17.2.3 Referencing repeating properties in comparison expressions

A comparison expression is an expression that uses a comparison operator, such as = or >, to compare the values of two properties. The following rules apply when comparing a repeating property to another property in a qualification:

- You can compare a repeating property to a single-valued property from the same or a different object type.

  For example, the following queries are legal:

  ```
  SELECT r_object_id, title FROM dm_document
  WHERE ANY authors=object_owner

  SELECT a.r_object_id, b.user_name
  FROM dm_document a, dm_user b
  WHERE ANY a.authors=b.user_name
  ```

- You cannot compare a repeating property to another repeating property unless you include the ROW_BASED hint.

  For example, the following query is illegal:

```
SELECT a.r_object_id,b.r_object_id,a.title
FROM dm_document a,dm_sysobject b
WHERE ANY a.authors = b.keywords
```

To make that example a legal statement, you must include the ROW_BASED hint:

```
SELECT a.r_object_id,b.r_object_id,a.title
FROM dm_document a,dm_sysobject b
WHERE ANY a.authors = b.keywords
ENABLE(ROW_BASED)
```

Note: The ROW_BASED hint may not be included in FTDQL queries or queries that reference a lightweight object type in the FROM clause.

2.20.17.3 FTDQL requirements for where clauses

A WHERE clause in an FTDQL query is subject to the following rules:

- Any repeating properties referenced in the clause must be of type string or ID.
- Only the DQL UPPER and LOWER functions are allowed. The functions SUBSTR and MFILE_URL and all aggregate functions are not acceptable.
- The following predicates are not allowed:
  - BETWEEN
  - NOT LIKE
  - NOT FOLDER
  - [NOT] CABINET
  - ONLY
  - TYPE
- The IN and EXISTS keywords are not allowed.
- Any valid form of the FOLDER predicate (except NOT FOLDER) may be used. The DESCEND option is also allowed.
- The following rules apply to the LIKE predicate:
  - The LIKE predicate may be used with pattern matching characters.
  - The LIKE predicate can be used with an ESCAPE clause, but it is ignored.
- The keywords TODAY, YESTERDAY, TOMORROW may not be used in the DATE function.
- The following rules apply to all expressions in the WHERE clause:
  - Expressions may not contain the ISREPLICA or USER keywords.
  - Expressions may use any comparison operator.
  - Expressions may use an arithmetic operator, but they may not be used to form a compound expression.

  - Expressions that compare one property to another are not allowed. For example, subject=title is invalid.
  - Expressions in the following format are not allowed:

    ```
    property_name operator('literal' operator 'literal)
    ```

  - Expressions that force index correspondence between repeating properties are not allowed. Such expressions AND together expressions that reference repeating properties in the format:

    ```
    predicate(repeating_attr_expr AND repeating_attr_expr)
    ```

    For more information about forcing index correspondence, refer to "Forcing index correspondence in query results" on page 367.

- The following additional rules apply to expressions in the format *first_expression operator second_expression*
  - The *first_expression* is limited to one of the following:

    ```
    property name
    upper(property name)
    lower(property name)
    ```

  - The *second_expression* is limited to one of the following:

    ```
    literal value
    upper(literal value)
    lower(literal value)
    the DATE() function
    ```

  - The operator may be any valid operator.

2.20.18 GROUP BY clause

Note: You may not include a GROUP BY clause in an FTDQL SELECT statement.

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"
```

2.20.19 HAVING clause

Note: You may not include a HAVING clause in an FTDQL SELECT statement.

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.

2.20.20 UNION clause

Note: You may not include a UNION BY clause in an FTDQL SELECT statement.

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 properties or columns as the first SELECT statement, and each property or column must have the same datatype as its corresponding property or column in that SELECT statement. For example, if the first SELECT statement returns three properties, then the subselect argument in the UNION clause must also return three properties. The datatypes of the first properties returned by each must be the same, as must the datatypes of the second and third properties.

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, when you union two or more SELECT statements, the property 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 properties, name and address, taken from the value list of the first SELECT in the statement.

The query result objects for this statement have two properties. The first property is named 1, representing selected values from name and ret_name. The second property 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.

2.20.21 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 a property 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.

Repeating attributes included in the ORDER BY clause of a query but not in the SELECT clause may cause duplicated rows to appear in the results. Add the i_position to ensure that the date returned by the query always appear in the same order. Adding the i_position attribute to the SELECT clause shows four distinct rows. Instead of using the `distinct` modifier which may cause an error, it is recommended to add r_object_id before the repeating attribute. For example:

```
select r_object_id, object_name, authors
FROM dm_sysobject
WHERE a_content_type = 'jar'
and object_name like 'ATMOS Plugin'
ORDER BY object_name, authors, r_object_id, i_position
```

2.20.21.1 FTDQL requirements for ORDER BY

If you include an ORDER BY clause in an FTDQL query, the sort order must be specified by SCORE:

```
ORDER BY SCORE
```

You cannot reference SCORE by its position in the selected values list. You must reference it by name. You may sort in either ascending or descending order.

2.20.22 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.

The ROW_BASED hint may not be included in FTDQL queries or in queries that reference a lightweight object type in the FROM clause. All other hints are acceptable in FTDQL queries or when querying lightweight object types.

"DQL standard hints" on page 162, lists the standard hints and provides brief guidelines for their use. For detailed information about their implementation, refer to Appendix A, Using DQL hints on page 373.

Table 2-21: DQL standard hints

Standard hint	Description
CONVERT_FOLDER_LIST_TO_OR	Changes DQL FOLDER clause to the behavior of release 6.0 and earlier.

Standard hint	Description
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.
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.
[NO]FTDQL	FTDQL directs Documentum CM Server to execute the query as an FTDQL query. If the syntax of the query violates the rules for FTDQL syntax, an error is returned. NOFTDQL directs Documentum CM Server to execute the query as a standard query.
FT_CONTAIN_FRAGMENT	Expands the range of matches for full-text searches that include a search clause of the form LIKE '%*string*%'.
GROUP_LIST_LIMIT *N*	GROUP_LIST_LIMIT defines the maximum number of groups that may be referenced in the generated SQL statement to check permissions for the user executing the query. The default limit is 250. Using this hint overrides the default value and the DM_GROUP_LIST_LIMIT environment variable, if that is set, as well as flushes and rebuilds the group list cache in the new size. For users whose group membership number is below the specified limit, generated SQL statements display group names inline without subqueries.
HIDE_SHARED_PARENT	HIDE_SHARED_PARENT directs Documentum CM Server to return only the rows in the query results that are not shared parents.
OPTIMIZATION_LEVEL level_1 level_2	Allows you to change the query optimization level. Set level_1 to the optimization level you want for the current query and level_2 to the optimization level desired for the connection after the query completes.

Standard hint	Description
IN and EXISTS	Mutually exclusive hints used in a standard WHERE clause, in a repeating property predicate that contains a subquery. If you do not include either IN or EXISTS explicitly, when Documentum CM Server translates the query, it includes one or the other automatically.
OPTIMIZE_ON_BASE_TABLE	Optimizes queries that reference properties contained within one type in the type hierarchy.
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.
RETURN_RANGE *starting_row ending_row* [*optimize_top_row*] '*sorting_clause*'	Specifies which rows are returned by a query sorted by the returned values of specified properties. This hint is provided as a general way to paginate the results of a query. *Guidelines* • *optimize_top_rows* provides the top rows for optimization. • *sorting_clause* has the format – '*attribute_name* [ASC|DESC] [,*attribute_name* [ASC|DESC]...]' It uses ascending by default.

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.
ROW_BASED	Directs Documentum CM Server to return query results that contain repeating property values in a row-based format—one row for each returned repeating property value. In addition, the hint also affects the rules governing: • Selecting repeating property values • Using an asterisk as a selected value • Referencing repeating properties in WHERE clause qualifications More information about these effects is found in “Repeating properties” on page 130, “The asterisk (*) as a selected value” on page 140, and “Using repeating properties in qualifications” on page 156.

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.
TRY_FTDQL_FIRST	TRY_FTDQL_FIRST directs Documentum CM Server to first execute the query as an FTDQL query and, if a timeout or resource exceeded error occurs, to retry the query as a standard query. If this hint is included, the FTDQL and NOFTDQL hints are ignored if they are also included.
UNCOMMITTED_READ	Ensures that a read only query returns quickly even if another session is holding locks on the tables queried by the read only query. This hint is useful only on SQL Server database.
DM_LEFT_OUTER_JOIN_FOR_ACL	If the user is not a super user, this hint ensures that Documentum CM Server generates an SQL query that contains a LEFT OUTER JOIN on the dm_acl_s table. This operation improves the performance of DQL queries.

2.20.23 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"
```

The following FTDQL example returns those documents that contain the phrase "for publication:" for which the user has Write permission:

```
SELECT FOR WRITE object_name,title,owner_name FROM dm_document
SEARCH DOCUMENT CONTAINS 'for publication'
```

There are additional examples demonstrating the use of specific clauses in the descriptions of the clauses.

2.21 Unregister

2.21.1 Purpose

Removes a registered table from the repository.

2.21.2 Syntax

```
UNREGISTER [TABLE] [owner_name.]table_name
```

2.21.3 Arguments

Table 2-22: UNREGISTER argument descriptions

Argument	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 repository user, this value is found in the user's user_db_name property. If the RDBMS is Oracle and the owner is the DBA, you can use the alias dm_dbo. If the RDBMS is SQL Server and if the owner is the DBA, you can use the alias dbo.
table_name	Identifies a registered table to remove from the repository. Use the table name as it appears in the RDBMS.

2.21.4 Permissions

You must be the owner of the registered table or have Superuser privileges to unregister a table.

2.21.5 Description

Unregistering a table removes the object of type dm_registered that represents the table in the repository. 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.

2.21.6 Related statements

"Delete" on page 96
"Insert" on page 111
"Register" on page 113

2.21.7 Examples

The following example unregisters the table called departments:

```
UNREGISTER TABLE "departments"
```

2.22 Update

2.22.1 Purpose

Updates the rows of a registered table.

2.22.2 Syntax

```
UPDATE table_name SET column_assignments[WHERE qualification]
```

2.22.3 Arguments

Table 2-23: UPDATE argument descriptions

Argument	Description
table_name	Identifies the registered table to update. Refer to "Register" on page 113 for information about registering tables in the repository.
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" on page 170 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 "WHERE clause" on page 155 for a full description of WHERE clauses and qualifications.

2.22.4 Return value

The UPDATE statement returns a collection whose result object has one property, rows_updated, that contains the number of updated rows.

2.22.5 Permissions

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

- Your object-level permission for the dm_registered object in the repository 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 Documentum CM Server is running must have the appropriate RDBMS permission to update the specified table. The actual name of this permission depends on your RDBMS.

The *OpenText Documentum Content Management - Server Administration and Configuration Guide (EDCCS250400-AGD)* contains more information about security, object-level and table permissions.

2.22.6 Description

This section contains usability information.

2.22.6.1 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.

Note: The UPDATE statement with more than 100 different repeating attributes is not executed successfully.

2.22.6.2 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.

2.22.6.3 Identifying rows to update

The WHERE clause *qualification* determines which rows are updated. The *qualification* 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.

2.22.7 Related statements

"Delete" on page 96
"Insert" on page 111
"Register" on page 113
"Unregister" on page 167

2.22.8 Examples

This example updates the column called chef_name:

```
UPDATE "recipes" SET "chef_name" = 'carol'
WHERE "chef_name" = 'carole'
```

2.23 Update...Object

2.23.1 Purpose

Updates an object in the repository.

2.23.2 Syntax

```
UPDATE [PUBLIC]type_name [(ALL)][correlation_var]
[WITHIN PARTITION partition_id {,partition_id}]
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]
```

2.23.3 Arguments

Table 2-24: UPDATE...OBJECT argument descriptions

Argument	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 property references in the statement.
WITHIN PARTITION clause	Restricts the statement to objects in particular partitions. *partition_id* identifies the object partition. The property, i_partition, contains the partition value for an object.

update_list	Specifies the update operations to perform on the object. Valid formats are:
	
	`set` *`property_name`* `=` *`value`*`set`
	`property_name``[`*`[index]`*`]` `=` *`value`*`append [`*`n`*`]`
	`property_name` `=` *`value`*`insert`
	`property_name``[`*`[index]`*`]` `=` *`value`*`remove`
	`property_name``[`*`[index]`*`] truncate`
	`property_name``[`*`[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” on page 175 for details.
SETFILE clause WITH PAGE_NO option	Adds additional content to the object. Refer to “WITH PAGE_NO option” on page 175 for details.

Argument	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 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 "SEARCH clause" on page 151 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, property names, scalar and date functions, special keywords, predicates, and arithmetic expressions. Multiple expressions can be joined together using logical operators. Refer to "WHERE clause" on page 155 for a detailed description of the clause.

2.23.4 Return value

The UPDATE...OBJECT statement returns a collection whose result object has one property, called objects_updated, that contains the number of objects updated.

2.23.5 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.

2.23.6 Description

This section contains usability information.

2.23.6.1 General notes

To update an object, the object: cannot belong to a frozen assembly or a frozen or immutable virtual document.

Aspects cannot be updated using the UPDATE...OBJECT statement; instead you can use Foundation Java API or IAPI.

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 property 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.

2.23.6.2 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.

2.23.6.2.1 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 Documentum CM Server.

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

2.23.6.2.2 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 Documentum CM 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.

2.23.6.3 update operations

The *update_list* defines the operations to perform. You can:

- Set the value of a single-valued or repeating property
- Add a value for a repeating property
- Insert a value into the list of values for a repeating property
- Remove a value from a repeating property
- Truncate a repeating property (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 and write properties. These properties have names beginning with a_ or have no prefix at all. If you have Superuser user privileges, you can update read-only properties. These properties 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 properties and inserts a value into a repeating property. 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.

2.23.6.3.1 Setting a property value

To set the value of a single-valued or repeating property, use the following syntax:

```
set property_name[[index]] = value
```

The *property_name* argument is the name of the property. If the property is a repeating property, the *index* identifies the position of the new value in the property's list of values. The positions of the values for a repeating property are numbered from zero. Enclose the *index* value in square brackets.

The *value* argument is the value to assign to the property. 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.

2.23.6.3.2 Appending a value to a repeating property

To append a value to a repeating property, use the following syntax:

```
append [n]property_name = value
```

The *property_name* argument is the name of the property to which to append a new value. The *value* argument specifies what value to add to the property's ordered list of values. The value is automatically added to the end of the repeating property'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).

2.23.6.3.3 Inserting a value into a repeating property

To insert a value into the list of values for a repeating property, use the following syntax:

```
insert property_name[[index]] = value
```

The *property_name* argument identifies the property. The *index* argument defines where to insert the new value. The positions of all values for a repeating property 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 property (for example, if the statement is in a program loop), be sure to increase the index number each time you insert a value.

2.23.6.3.4 Removing values from repeating properties

To remove a value from a repeating property, use the following syntax:

```
remove property_name[[index]]
```

The *property_name* argument identifies the property. The *index*argumentindicates the position of the value to remove in the property's ordered list of values. The positions of all values for a repeating property are numbered from zero. If you do not include the index, the server removes the first value (position 0) in the property. 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 property (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.

2.23.6.3.5 Truncating a repeating property

To truncate a repeating property (remove its values), use the following syntax:

```
truncate property_name[[index]]
```

The *property_name* argument identifies the property. The *index* argument specifies where in the property's list of values to begin the truncation. For example, if you specify property_name[4], the server removes all values beginning with property_name[4]. If you do not specify an index level, the server removes all values.

2.23.6.3.6 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}
```

2.23.6.3.7 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.

2.23.7 Related statements

"Change...Object" on page 69
"Create...Object" on page 76
"Delete...Object" on page 97

2.23.8 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. "Administration methods by category" on page 182, lists the methods by task categories. They are executed in an application by invoking either the DQL EXECUTE statement or the IDfSession.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

  Only the DQL syntax is shown. For the IDfSession.apply method syntax, refer to the Javadocs.
- Arguments
- Return Value
- Permissions
- Description
- Related Administration Methods
- Examples

3.1 Invoking administration methods

To execute an administration method manually, use Documentum Administrator. All the methods (except DO_METHOD, WEBCACHE_PUBLISH, and ROLES_FOR_USER) are available through the Methods facility in repository Management. Many of the methods are also available through category-specific pages.

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.

To execute an administration method in an application, use either the IDfSession.apply method or the DQL EXECUTE statement. You can also use EXECUTE to invoke an administration method through IDQL.

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.

For information about using IDfSession.apply to invoke an administration method, refer to the Javadocs.

3.1.1 Scope of the administration methods

Administrative methods execute in the context of the current repository scope. You cannot connect to a repository and execute an administration method against a different repository.

3.2 Administration method operations

“Administration methods by category” on page 182, 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” on page 187	Promotes multiple objects to their next lifecycle states. **Note:** This method is not available through Documentum Administrator or using DQL EXECUTE.
	“CHECK_SECURITY” on page 197	Checks a user or group’s permissions level for one or more objects.
	“FIX_LINK_CNT” on page 224	Updates the r_link_cnt property for a specified folder.
	“GET_INBOX” on page 232	Returns items in user’s Inbox.
	“MARK_AS_ARCHIVED” on page 264	Sets the i_is_archived property of a dm_audittrail, dm_audittrail_acl, or dm_audittrail_group to T.
	“PURGE_AUDIT” on page 310	Removes audit trail entries from the repository.
	“RECOVER_AUTO_TASKS” on page 320	Recovers work items claimed by a workflow agent master session but not yet processed.

Category of operation	Administration Method	Description
	"ROLES_FOR_USER" on page 332	Returns the roles assigned to a user in a particular client domain.
Execute methods	"DO_METHOD" on page 207	Executes system-defined procedures such as lpq or who or user-defined procedures.
	"HTTP_POST" on page 240	Directs the execution of a method to an application server.
Content storage management	"CAN_FETCH" on page 188	Determines whether content in a distributed storage area component can be fetched by the server.
	"CHECK_RETENTION_EXPIRED" on page 193	Finds SysObjects in content-addressed storage that have an expired retention period or no retention period.
	"CLEAN_LINKS – Deprecated" on page 201 This method is deprecated. Foundation Java API 6.0 does not support linked storage areas nor link record objects, which consequently, deprecates this method.	Provides maintenance for linked store storage areas. On Linux, 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" on page 204	Removes a replica from a distributed storage area.
	"DESTROY_CONTENT" on page 206	Removes a content object and its associated file from the repository. (Do not use this for archiving; use PURGE_CONTENT instead.)
	"GET_FILE_URL" on page 230	Returns the URL to a content file.
	"GET_PATH" on page 237	Returns the path to a particular content file in a particular distributed storage area component.

Category of operation	Administration Method	Description
	"IMPORT_REPLICA" on page 245	Imports an external file as a replica of content already in the repository.
	"MIGRATE_CONTENT" on page 267	Moves content files from one storage area to another.
	"PURGE_CONTENT" on page 316	Deletes a content file from a storage area. Used as part of the archiving process.
	"PUSH_CONTENT_ATTRS" on page 317	Sets the content metadata in a content-addressed storage system for a document stored in that storage system.
	"REGISTER_ASSET" on page 321	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 Transformation Services - Media running.
	"REPLICATE" on page 327	Copies content in one component of a distributed storage area to another area.
	"RESTORE_CONTENT" on page 330	Moves a file or files from archived storage to the original storage location.
	"SET_CONTENT_ATTRS" on page 336	Sets the content_attr_name and content_attr_value properties in the content object associated with the content file.
	"SET_STORAGE_STATE" on page 343	Sets the state of a storage area to off-line, on-line, or read-only.
	"TRANSCODE_CONTENT" on page 347	Queues a request for a content transformation to the Media Server. This method is only available or useful if you have Transformation Services - Media running.

Category of operation	Administration Method	Description
Database Methods	"DB_STATS" on page 202	Provides database operation statistics for a session.
	"DROP_INDEX" on page 217	Drops an index.
	"EXEC_SQL" on page 220	Executes SQL statements.
	"FINISH_INDEX_MOVES" on page 223	Completes an interrupted move operation for an object type index.
	"GENERATE_PARTITION_SCHEME_SQL – Deprecated" on page 225	Creates an SQL script to partition a repository. (Deprecated)
	"MAKE_INDEX" on page 260	Creates an object type index.
	"MIGRATE_TO_LITE" on page 285	Migrates standard SysObjects to lightweight SysObjects and shareable parents.
	"MOVE_INDEX" on page 297	Moves an object type index from one tablespace or segment to another.
	"PARTITION_OPERATION" on page 299	Partitions a repository.
	"REORGANIZE_TABLE" on page 324	Reorganizes a database table for query performance.
	"UPDATE_STATISTICS" on page 350	Updates the statistics in a database table.
Full-Text Methods	"ESTIMATE_SEARCH" on page 218	Returns the number of results matching a particular SEARCH condition.
	"MARK_FOR_RETRY" on page 265	Finds all content objects that have a particular negative update_count and marks them as awaiting indexing.
	"MODIFY_TRACE" on page 295	Sets the tracing level for full-text indexing operations.
Session Management	"CHECK_CACHE_CONFIG" on page 190	Requests a consistency check on a particular cache config object.
	"GET_LAST_SQL" on page 236	Returns the last SQL statement issued.
	"GET_SESSION_DD_LOCALE" on page 239	Returns the locale in use for the current session.

Category of operation	Administration Method	Description
	"LIST_AUTH_PLUGINS" on page 248	Lists the authentication plug-ins loaded by Documentum CM Server.
	"LIST_RESOURCES" on page 249	Provides information about the server operating system environment.
	"LIST_SESSIONS" on page 253	Provides information about current, active sessions.
	"LIST_TARGETS" on page 256	Lists the connection brokers defined as targets for the server. The information is returned in a collection with one result object whose properties list the connection brokers defined as targets for the server.
	"LOG_ON" on page 259 and "LOG_OFF" on page 258	Turn server logging of information about RPC calls on or off.
	"PING" on page 309	Determine if a client has an active server connection.
	"SET_APIDEADLOCK" on page 333	Sets a deadlock trigger on a particular API method or operation.
	"SET_OPTIONS" on page 340	Turn various tracing options on or off.
	"SHOW_SESSIONS" on page 345	Provides information about current, active sessions and a user-specified number of timed-out sessions.
Web Publishing Management	"WEBCACHE_PUBLISH" on page 353	Invokes the dm_webcache_publish method to publish documents to a Web site.

3.3 BATCH_PROMOTE

3.3.1 Purpose

Promotes multiple objects to their next state.

3.3.2 Syntax

This method cannot be executed using the DQL EXECUTE statement.

3.3.3 Arguments

Table 3-2: 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.

3.3.4 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 IDfSession.getMessage to retrieve the error message.

If BATCH_PROMOTE returns a collection, check the value of the method_return_val property 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 an IDfSession.getMessage to retrieve the error message.

3.3.5 Permissions

The user issuing the method must have the necessary permissions to promote the objects listed in the argument.

3.3.6 Description

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 property in the repository configuration (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.

3.3.7 Related administration methods

None

3.4 CAN_FETCH

3.4.1 Purpose

Determines if the server can fetch a specified content file.

3.4.2 Syntax

```
EXECUTE can_fetch FOR 'content_object_id'
```

3.4.3 Arguments

CAN_FETCH has no arguments.

3.4.4 Return value

CAN_FETCH returns a collection with one query result object. The object has one Boolean property that is set to TRUE if the fetch is possible or FALSE if it is not.

3.4.5 Permissions

Anyone can use this method.

3.4.6 Description

If a repository 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. The *OpenText Documentum Content Management - Server and Server Extensions Installation Guide (EDCSY250400-IGD)* contains more information about configuring this capability. In such repositories, you can use the CAN_FETCH method to determine whether a server can fetch from a particular distributed storage area component.

In a Documentum CM Server configuration, the CAN_FETCH method is executed by the Documentum CM Server.

3.4.7 Related administration methods

"GET_PATH" on page 237

3.4.8 Examples

The following examples determine whether Documentum CM Server can fetch the content file associated with the content object 06000002472185e1:

```
EXECUTE can_fetch FOR '06000002472185e1'
```

3.5 CHECK_CACHE_CONFIG

3.5.1 Purpose

Requests a consistency check on a particular cache config object.

3.5.2 Syntax

```
EXECUTE check_cache_config [FOR 'cache_config_id']
[WITH argument = value ][,argument = value]
```

Do not include the FOR clause if you include the CONFIG_NAME argument.

3.5.3 Arguments

Table 3-3: 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 Documentum CM 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).

3.5.4 Return value

The method returns a collection with one query result object. The object has the properties listed in “Properties in the CHECK_CACHE_CONFIG result object” on page 191.

Table 3-4: Properties in the CHECK_CACHE_CONFIG result object

Property	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 Documentum CM Server last validated the query results and found that something had changed.
r_last_checked_date	Date/time	The date and time at which Documentum CM 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 methods that include this cache config 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.

3.5.5 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.

3.5.6 Description

CHECK_CACHE_CONFIG directs Documentum CM 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 property. 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 property. The method returns the new computation date and time in the query result object's r_last_checked_date property.

You can use the FORCE_CHECK argument to override the server check interval and force Documentum CM Server to execute the queries.

The *OpenText Documentum Content Management - Server Fundamentals Guide (EDCCS250400-GGD)* contains more information about persistent client caching and cache config objects and their use. You can execute the dm_CheckCacheConfig method manually, using a DO_METHOD administration method. The *OpenText Documentum Content Management - Server Administration and Configuration Guide (EDCCS250400-AGD)* contains examples of using DO_METHOD to call dm_CheckCacheConfig. 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 methods.

3.5.7 Related administration methods

None

3.5.8 Examples

The following example identifies the cache config object by its object ID and forces the recomputation of the data:

```
EXECUTE check_cache_config FOR '08000002723ae36b'
WITH force_check = true
```

This next example identifies the cache config object by its owner and name.

```
EXECUTE check_cache_config
WITH config_name='johndoe.report_app_config'
```

3.6 CHECK_RETENTION_EXPIRED

3.6.1 Purpose

Generates a list of objects whose content, stored in content-addressed storage, has an expired retention period or a zero retention period.

3.6.2 Syntax

```
EXECUTE CHECK_RETENTION_EXPIRED
WITH QUERY='where_clause'
[,SELECT_LIST='property_list']
[,INCLUDE_ZERO_RETENTION_OBJECTS=TRUE|FALSE]
```

3.6.3 Arguments

Table 3-5: CHECK_RETENTION_EXPIRED arguments

Argument	Datatype	Value	Description
QUERY	S	*where_clause*	Defines which SysObjects are to be checked for an expired or no retention period. Consists of a DQL SELECT where clause. Only literal values and properties defined for the dm_sysobject type may be referenced. The string must be enclosed in single quotes.

Argument	Datatype	Value	Description
SELECT_LIST	S	*property_list*	Identifies additional properties whose values are to be returned in the result object. Only properties defined for the dm_sysobject object type may be included. Separate multiple property names with commas and enclose the entire list in single quotes.
INCLUDE_ZERO_RETENTION_OBJECTS	B	T (TRUE) or F (FALSE)	If set to T, the method checks objects stored in content-addressed storage areas that allow but do not require a retention period. The default is F, meaning that only objects stored in content-addressed storage areas that require a retention period are checked.

3.6.4 Return value

CHECK_RETENTION_EXPIRED returns a collection of result objects, each representing one SysObject whose retention has expired or that had no retention period. The result objects have five default properties plus any specified in the SELECT_LIST argument. The default properties of the result object are:

- r_object_id
- object_name
- a_storage_type
- r_creation_date
- retention_date

  The retention_date property is a computed property. The date value is the GMT equivalent of the retention period as it is defined in the time zone of the Centera storage system. For example, suppose the Centera system is in the Eastern time zone (EST) and the client on which user is working is in the Pacific time zone (PST). If the user sets the retention value to March 15, 2004 11 a.m. PST, the retention value is stored as March 15, 2004 2 p.m.—the Eastern time zone equivalent of March 15, 2004 11 a.m. The value returned in the computed property is the GMT equivalent of March 15, 2004 2 p.m.

3.6.5 Permissions

You must have Sysadmin or Superuser privileges to execute this method.

3.6.6 Description

The CHECK_RETENTION_EXPIRED method is used by the RemoveExpiredRetnObjects administration job to find the content in content-addressed storage that has an expired retention period. The method returns a list of the objects. It does not remove the content from the storage area, nor does it remove from the repository any of the objects that contain the content.

A content-addressed storage area can have three possible retention period configurations:

- The storage area may require a retention period.

  In this case, the a_retention_attr name property is set and the a_retention_attr_req is set to T.
- The storage area may not allow a retention period.

  In this case, the a_retention_attr name property is not set and the a_retention_attr_req is set to F.
- The storage area may allow but not require a retention period.

  In this case, the a_retention_attr name property is set , but the a_retention_attr_req is set to F.

By default, the method operates only on objects that are stored in content-addressed storage areas that require a retention period. The method never includes objects stored in content-addressed storage areas that do not allow retention periods. If you want it to examine objects stored in content-addressed storage areas that allow but do not require a retention period, you must set the INCLUDE_ZERO_RETENTION_OBJECTS argument to true (refer to "INCLUDE_ZERO_RETENTION_OBJECTS argument" on page 196, for more information).

3.6.6.1 QUERY argument

The QUERY argument's where clause is a DQL where clause qualification. It is used to select objects for possible inclusion in the results returned by the method. The objects that fulfill the QUERY argument and are stored in an appropriate content-addressed storage area are then examined to determine whether the retention period has expired. If so, the object is included in the results.

The QUERY argument can reference only literal values or properties defined for the dm_sysobject object type. If the where clause includes single quotes, you must escape them with single quotes. For example:

```
...QUERY,S,'a_storage_type=''castore_1''
and r_creation_date > DATE(01/01/2003)'
```

3.6.6.2 INCLUDE_ZERO_RETENTION_OBJECTS argument

By default, the method does not include objects whose content has a 0 retention period because the assumption is that such content is meant to be kept forever. However, in a storage area that allows but does not require a retention period, a 0 retention period can be result from two possible causes:

- The user deliberately set no retention period, and consequently, the server set the retention period to 0.
- The user specified a retention date that had already elapsed. When this occurs, the server sets the retention period to 0.

Because the meaning of 0 is ambiguous in such storage areas, the method supports the INCLUDE_ZERO_RETENTION_OBJECTS argument to allow you to include content with a zero retention in storage areas that allow but do not require a retention period.

If you set INCLUDE_ZERO_RETENTION_OBJECTS to T, when the method examines objects in storage areas that allow but do not require a retention period and it will include in the results any object with an expired or zero retention period.

3.6.6.3 SELECT_LIST argument

The SELECT_LIST argument allows you to include additional SysObject properties in the result objects. You must enclose the argument's value in single quotes.

3.6.7 Related administration methods

None

3.6.8 Examples

The following example checks SysObjects stored in the content-addressed storage area named “castore_2” owned by the user named “John Arthur”. It also adds the title and subject property values to the result objects.

```
EXECUTE check_retention_expired
WITH QUERY='a_storage_type=''castore_2'' and
owner_name=''John Author''',
SELECT_LIST='title,subject'
```

This example directs the method to include objects in a storage area that allows but doesn't require a retention period:

```
EXECUTE check_retention_expired
WITH QUERY='a_storage_type=''castore_3'' and
owner_name=''Mary Writer''',
SELECT_LIST='title,subject',
INCLUDE_ZERO_RETENTION_OBJECTS=true
```

3.7 CHECK_SECURITY

3.7.1 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.

3.7.2 Syntax

```
EXECUTE check_security WITH user_name='name'|group_name='name',
level=security_level,object_list='list_of_objectids'
```

3.7.3 Arguments

Table 3-6: 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.

Argument	Datatype	Value	Description
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.

3.7.4 Return value

CHECK_SECURITY returns a collection of query result objects. The objects have one property, r_object_id. The property 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.

3.7.5 Permissions

You must have Superuser privileges to use this method.

3.7.6 Description

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).

Note: When CHECK_SECURITY DQL is executed, it only verifies the Superuser level default permission on SysObject. By design, Superuser has the default read permission (DM_PERMIT_READ) on SysObject and DM_PERMIT_DELETE for all ACLs and the query result returns the same. It does not check the actual granted permit on object for Superuser which is inherited from object owner. Currently, Superuser is granted the accessor permit on SysObject as object owner.

3.7.7 Related administration methods

None

3.7.8 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:

```
EXECUTE check_security WITH group_name='Engineering',
level=3,object_list='4500000112562e4c 450000017351213c'
```

3.8 CLEAN_DELETED_OBJECTS

3.8.1 Purpose

Removes deleted lightweight and parent objects from the repository.

3.8.2 Syntax

To remove a deleted parent and all its children:

```
EXECUTE clean_deleted_objects [[FOR] deleted_parent_object_id]
```

To remove private parents that no longer have lightweight children:

```
EXECUTE clean_deleted_objects WITH execution_mode='remove_orphaned_parents'
```

3.8.3 Arguments

Table 3-7: CLEAN_DELETED_OBJECTS arguments

Argument	Datatype	Value	Description
execution_mode	S	remove_orphaned_parents	The only value allowed is remove_orphaned_parent.

3.8.4 Return value

Returns a list of removed objects

3.8.5 Permissions

You must have Sysadmin or Superuser privileges to use this method.

3.8.6 Description

CLEAN_DELETED_OBJECTS removes deleted objects from the database. Since a shareable parent object can have many lightweight children, for performance reasons, all the lightweight children are not removed when a shareable parent is deleted. The dmClean job can be configured to remove these objects as part of the usual repository cleanup, or this method can be used to start removing the objects immediately.

When a lightweight object is reparented from its private parent to a shareable parent, the private parent is not deleted until the CLEAN_DELETED_OBJECTS method is run with remove_orphaned_parents set.

3.9 CLEAN_LINKS – Deprecated

3.9.1 Purpose

On Windows, this method removes unneeded dmi_linkrecord objects and resets security on file store objects to the original state. On Linux, this method removes unneeded dmi_linkrecord objects and the auxiliary directories and symbolic links.

3.9.2 Syntax

```
EXECUTE clean_links [WITH force_active=true|false]
```

3.9.3 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.

3.9.4 Return value

CLEAN_LINKS returns a collection with one query result object. The object has one Boolean property whose value indicates the success (TRUE) or failure (FALSE) of the operation.

3.9.5 Permissions

You must have Superuser privileges to use this method.

3.9.6 Description

Note: The CLEAN_LINKS method is deprecated. Foundation Java API 6.0 does not support linked storage areas. Consequently, the CLEAN_LINKS method, which supports those storage areas, is deprecated.

Linked storage areas are handled differently on Windows and Linux platforms. Consequently, the behavior of CLEAN_LINKS is slightly different on each platform.

On Linux, 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 Linux, 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 properties of the dmi_linkrecord objects. If the session_id property 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.

3.9.7 Related administration methods

"LIST_SESSIONS" on page 253

3.9.8 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 repository sessions, active and inactive:

```
EXECUTE clean_links WITH force_active=true
```

3.10 DB_STATS

3.10.1 Purpose

Returns a set of statistics about database operations for the current session.

3.10.2 Syntax

```
EXECUTE db_stats [WITH clear = true|false]
```

3.10.3 Arguments

Table 3-8: DB_STATS arguments

Argument	Datatype	Value	Description
clear	Boolean	true | false	Indicates whether you want to clear the counters. The default is FALSE.

3.10.4 Return value

DB_STATS returns a collection with one result object, described in "Query result object properties for DB_STATS administration method" on page 203.

Table 3-9: Query result object properties for DB_STATS administration method

Property	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 Documentum CM 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.

3.10.5 Permissions

Anyone can use this method.

3.10.6 Description

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

3.10.7 Related administration methods

None

3.10.8 Examples

The following example uses EXECUTE to invoke DB_STATS. It returns the statistics for the current repository session and resets the counters to zero:

```
EXECUTE db_stats WITH clear=true
```

3.11 DELETE_REPLICA

3.11.1 Purpose

Removes a content file from a component of a distributed storage area.

3.11.2 Syntax

```
EXECUTE delete_replica FOR 'content_object_id'
WITH STORE='storage_name'
```

3.11.3 Arguments

Table 3-10: DELETE_REPLICA arguments

Argument	Datatype	Value	Description
store	string	*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.

3.11.4 Return value

DELETE_REPLICA returns a collection with one query result object. The object has one Boolean property that indicates the success (TRUE) or failure (FALSE) of the operation.

3.11.5 Permissions

You must have Superuser privileges to use this method.

3.11.6 Description

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 Documentum Servers, you must issue a connection request that bypasses the Documentum CM Server to use DELETE_REPLICA in the session.

3.11.7 Related administration methods

"IMPORT_REPLICA" on page 245

3.11.8 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'
```

3.12 DESTROY_CONTENT

3.12.1 Purpose

Removes content objects from the repository and their associated content files from storage areas.

3.12.2 Syntax

```
EXECUTE destroy_content FOR 'content_obj_id'
```

3.12.3 Arguments

DESTROY_CONTENT has no arguments.

3.12.4 Return value

DESTROY_CONTENT returns a collection with one query result object. The object has one Boolean property that indicates the success (TRUE) or failure (FALSE) of the operation.

3.12.5 Permissions

You must have Sysadmin or Superuser privileges to use this method.

3.12.6 Description

DESTROY_CONTENT removes orphaned content objects. An orphaned content object is a content object that has no values in its parent_id property. The method also removes the content file referenced by the orphaned content object if there are no other content objects that reference that file.

The DESTROY_CONTENT method is the method invoked by dmclean.

To run DESTROY_CONTENT, you must be using one server for both data and content requests. If the configuration is set up for Documentum Servers, you must issue a connection request that bypasses the Documentum CM 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.

3.12.6.1 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), Documentum CM 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, Documentum CM Server checks the retention period stored with each address. The retention period for all addresses must be expired before Documentum CM Server can destroy the content.

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 property 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.

3.12.7 Related administration methods

"PURGE_CONTENT" on page 316

3.12.8 Examples

This example uses the EXECUTE statement to invoke DESTROY_CONTENT:

```
EXECUTE destroy_content FOR '06000001684537b1'
```

3.13 DO_METHOD

3.13.1 Purpose

Executes an external program, a Docbasic script, or a Java method. Note that use of Docbasic is deprecated.

3.13.2 Syntax

```
EXECUTE do_method WITH METHOD='method_name'
{,arguments='value'}
```

3.13.3 Arguments

Table 3-11: DO_METHOD arguments

Argument	Datatype	Value	Description
save_results	Boolean	true | false	Indicates if you wish to save the results into a document.
arguments	string	*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 Documentum CM Server, only characters from the server OS code page are accepted for the argument strings. Refer to “Specifying the arguments” on page 213, for more information.
time_out	integer	*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” on page 214 for more information.
method	string	*method_name*	Name of the method object representing the script, program, or Java method to execute. This argument is required.

Argument	Datatype	Value	Description
launch_direct	Boolean	true | false	Indicates whether to execute the program using the Windows 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" on page 214 for more information. This argument is ignored if the method's use_method_server property is TRUE.
launch_async	Boolean	true | 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" on page 214 for more information.

Argument	Datatype	Value	Description
run_as_server	Boolean	true | 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 property. 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 property set to FALSE. This argument must be TRUE for methods that have their use_method_server property set to TRUE.
trace_launch	Boolean	true | false	Indicates whether to generate tracing information for the method. “Recording tracing information and program output” on page 216 describes where the information is stored.

3.13.4 Return value

A DO_METHOD function returns a collection that contains one query result object. “Properties of the query result object returned by DO_METHOD” on page 211 lists the properties of this object.

Table 3-12: Properties of the query result object returned by DO_METHOD

Property	Datatype	Description
result	Integer or ID	If the program you launched was dmfilescan or dmclean, this property holds the object ID of the script run by the utility. Otherwise, the property 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 property 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 property is meaningless. F means that the program was launched and the return value is the exit code of the process when the method terminated.

Property	Datatype	Description
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 property 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.

3.13.5 Permissions

Anyone can use this method.

If the method you are executing has the run_as_server property set to FALSE in its method object, you must have at least Sysadmin privileges to override that setting in the DO_METHOD command line.

3.13.6 Description

Use DO_METHOD to execute a Java method, a Docbasic script, or other executable program. Note that Docbasic is deprecated. You can direct the method execution to the Java method server or Documentum CM Server. Docbasic programs are executed by Foundation Java API, using an emulation package. Which you choose depends on the language in which the program is written and how your site is configured. The *OpenText Documentum Content Management - Server Administration and Configuration Guide (EDCCS250400-AGD)* contains details about each of the execution agents and how to direct a method to the agents.

To use DO_METHOD, the invoked script or program must be defined in the repository by a method object. A method object contains properties 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. The *OpenText Documentum Content Management - Server Administration and Configuration Guide (EDCCS250400-AGD)* contains more information about creating method objects.

3.13.6.1 Specifying the arguments

There are no restrictions on the format of the argument list for methods executed by Documentum CM Server.

If the method is to be executed using the Java method 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:

```
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 Java method server, Documentum CM Server encodes the arguments using application/x-www-form-urlencoded format. For example, suppose you issued the following DO_METHOD:

```
EXECUTE do_method WITH METHOD='paymethod',
ARGUMENTS='-docbase accounting
 -user paymaster
-ticket DM_TICKET=0000000222a02054.accounting@host01
-document "weekly record"
```

Documentum CM Server sends the arguments in the following format:

```
docbase=accounting&user=paymaster&ticket=DM_TIICKET%
3D0000000222a02054.accounting%4Dhost01&document=weekly+record
```

3.13.6.2 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 property 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 property or less than the value assigned to the object's timeout_min property. 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.

3.13.6.3 Launching directly

When you execute DO_METHOD using Documentum CM Server as the execution agent, the server calls the 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 property must contain a fully qualified pathname.

3.13.6.4 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.

3.13.6.5 Saving results

For DO_METHOD methods that are executing on the application server, OpenText Documentum CM provides a simple, example interface that captures the program output so the output can be saved into the repository if SAVE_RESULTS is TRUE. The *OpenText Documentum Content Management - Server Administration and Configuration Guide (EDCCS250400-AGD)* contains the description of this interface.

3.13.6.6 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 Documentum CM Server's account, you can:

- Set the run_as_server property 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 property and the RUN_AS_SERVER argument are FALSE. To run as the server account, either both must TRUE or you must override the property setting by setting the RUN_AS_SERVER argument to TRUE. Overriding the property in the DO_METHOD command line by setting the RUN_AS_SERVER argument to TRUE requires Sysadmin or Superuser privileges. Overriding the property 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 emulation package or the Java method server as the execution agent, you must set both the method's run_as_server property 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 property, RUN_AS_SERVER must also be set to TRUE. If it is not, the method does not execute correctly.
- Documentum CM 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 property in the server config object to a blank), Documentum CM Server runs all procedures and all print jobs under its account.

3.13.6.7 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 Documentum CM Server by checking the IP address of the sender against a list of repositories. 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 repository host, the request fails.

The application server runs as the Documentum CM Server installation owner. Consequently, the servlet it invokes to execute DO_METHOD calls also runs as the installation owner. On Windows platforms, the current operating system user is allowed to log in to the repository without providing a password. Consequently, a servlet or an invoked Java method can log into the repository through the Documentum CM Server with superuser privileges without providing a password.

If you write a method that uses that process to log in, 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.

3.13.6.8 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, of up to 2047 characters, to the repository log. Setting SAVE_RESULTS to TRUE saves the execution output of the invoked program.

3.13.7 Related administration methods

"EXEC_SQL" on page 220
"HTTP_POST" on page 240

3.13.8 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")
```

3.14 DROP_INDEX

3.14.1 Purpose

Destroys a user-defined object type index.

3.14.2 Syntax

```
EXECUTE drop_index [[FOR] 'dmi_index_id']
[WITH name = 'index_name']
```

Do not include the FOR clause if you include the NAME argument.

3.14.3 Arguments

Table 3-13: DROP_INDEX arguments

Argument	Datatype	Value	Description
name	string	*index_name*	Identifies the index by the name of its index (dmi_index) object.

3.14.4 Return value

DROP_INDEX returns a collection that contains one query result object. The object has one Boolean property whose value indicates the success (TRUE) or failure (FALSE) of the operation.

3.14.5 Permissions

You must have Superuser privileges to use this method.

3.14.6 Description

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 repository.

3.14.7 Related administration methods

"FINISH_INDEX_MOVES" on page 223
"MAKE_INDEX" on page 260
"MOVE_INDEX" on page 297

3.14.8 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'
```

3.15 ESTIMATE_SEARCH

3.15.1 Purpose

Returns the number of results matching a particular SEARCH condition.

3.15.2 Syntax

```
EXECUTE estimate_search [[FOR] 'fulltext_index_obj_id']
WITH [name = 'index_name'] [,type = 'object_type']
[,query = 'value']
```

Do not include the FOR clause if you include the NAME argument.

3.15.3 Arguments

Table 3-14: ESTIMATE_SEARCH arguments

Argument	Datatype	Value	Description
name	string	*index_name*	Identifies the index to be searched. This is optional. You can specify the object ID of the fulltext index object instead.
type	string	*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	string	*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.

3.15.4 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.

3.15.5 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 TRUE.

3.15.6 Description

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.

3.15.7 Related administration methods

None

3.15.8 Examples

```
EXECUTE estimate_search WITH name='filestore2_indx',
type='dm_document',query='Competitor Evaluation'
```

3.16 EXEC_SQL

3.16.1 Purpose

Executes SQL statements.

3.16.2 Syntax

```
EXECUTE exec_sql WITH query='sql_query'
```

3.16.3 Arguments

Table 3-15: EXEC_SQL arguments

Argument	Datatype	Value	Description
query	string	*sql_query*	Defines the query that you want to execute.

3.16.4 Return value

EXEC_SQL returns a collection that contains one query result object. The object has one Boolean property whose value is TRUE if the query succeeded and FALSE if it was unsuccessful.

3.16.5 Permissions

You must have superuser privileges to use this method.

3.16.6 Description

EXEC_SQL executes any SQL statement with the exception of SQL SELECT statements.

If you use an IDfSession.apply method to execute the method and the query contains commas, you must enclose the entire query in single quotes.

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))'
```

3.16.7 Related administration methods

"DO_METHOD" on page 207

3.16.8 Examples

Refer to the Description.

3.17 EXPORT_TICKET_KEY

3.17.1 Purpose

Returns a login ticket key.

3.17.2 Syntax

```
EXECUTE export_ticket_key WITH PASSWORD='password'
```

3.17.3 Arguments

Table 3-16: EXPORT_TICKET_KEY arguments

Argument	Datatype	Value	Description
password	string	*password*	User-defined password used to encrypt the returned login ticket key.

3.17.4 Return value

If successful, the method returns the login ticket key used by the repository as an encrypted and ASCII-encoded string. The returned key is encrypted using the password provided as an argument and then encoded as an ASCII string.

If the method fails, the method returns NULL.

3.17.5 Permissions

You must have Superuser privileges to execute this method.

3.17.6 Description

Use EXPORT_TICKET_KEY when you want to copy a login ticket key from one repository to another, to configure a trust relationship between the two repositories. The *OpenText Documentum Content Management - Server Fundamentals Guide (EDCCS250400-GGD)* contains the description of trusted repositories.

3.17.7 Related methods

"IMPORT_TICKET_KEY" on page 246
"RESET_TICKET_KEY" on page 329

IDfSession.exportTicketKey()

3.17.8 Examples

```
EXECUTE export_ticket_key WITH PASSWORD='myword'
```

3.18 FINISH_INDEX_MOVES

3.18.1 Purpose

Completes all unfinished object type index moves.

3.18.2 Syntax

```
EXECUTE finish_index_moves
```

3.18.3 Arguments

FINISH_INDEX_MOVES has no arguments.

3.18.4 Return value

FINISH_INDEX_MOVES returns a collection with one query result object. The object has one Boolean property whose value indicates the success (TRUE) or failure (FALSE) of the operation.

3.18.5 Permissions

You must have Superuser privileges to use this method.

3.18.6 Description

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 repository 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.

3.18.7 Related administration methods

"DROP_INDEX" on page 217
"MAKE_INDEX" on page 260
"MOVE_INDEX" on page 297

3.18.8 Examples

Refer to the syntax description.

3.19 FIX_LINK_CNT

3.19.1 Purpose

Updates the r_link_cnt property for a folder object.

3.19.2 Syntax

```
EXECUTE fix_link_cnt FOR folder_object_id
```

3.19.3 Arguments

None

3.19.4 Return Value

FIX_LINK_CNT returns T (TRUE) if successful or F (FALSE) if unsuccessful.

3.19.5 Permissions

Executing this method requires either Superuser privileges or Write permission on the specified folder.

3.19.6 Description

Use this method if the value of a folder's r_link_cnt property is incorrect. The method determines how many documents are linked to the specified folder and then sets that value in the folder's r_link_cnt property.

3.19.7 Examples

```
EXECUTE fix_link_cnt FOR 0b0000023ca4574f1
```

3.20 GENERATE_PARTITION_SCHEME_SQL – Deprecated

3.20.1 Purpose

This method is deprecated, from the 6.6 release. See "PARTITION_OPERATION" on page 299, for its replacement. Creates an SQL script to control repository partitioning.

3.20.2 Syntax

To generate a script to partition a database:

```
EXECUTE generate_partition_scheme_sql WITH [operation='db_partition',]
[type_name='type_name',|table_name='regtable_name',[owner_name='owner_name',]]
[last_partition ='partition_name',last_tablespace='tablespace',]
partition_name='partition_name',range=integer,tablespace='tablespace'
{,partition_name='partition_name',range=integer,tablespace='tablespace'}
[,include_object_type={TRUE|FALSE}]
```

To generate a script to add partition(s) to a database:

```
EXECUTE generate_partition_scheme_sql WITH operation='add_partition',
[type_name='type_name',|table_name='regtable_name',[owner_name='owner_name',]]
partition_name='partition_name',range=integer,tablespace='tablespace'
{,partition_name='partition_name',range=integer,tablespace='tablespace'}
[,include_object_type={TRUE|FALSE}]
```

To generate a script to exchange a partition:

```
EXECUTE generate_partition_scheme_sql WITH operation='exchange_partition',
[temp_table_suffix='temp_table_suffix'],
type_name='type_name',
partition_name='partition_name'
,include_object_type={TRUE|FALSE}
```

```
EXECUTE generate_partition_scheme_sql WITH operation='exchange_partition',
[temp_table_suffix='temp_table_suffix'],
table_name='regtable_name',[owner_name='owner_name',]
partition_name='partition_name'
,include_object_type={TRUE|FALSE}
```

3.20.3 Arguments

Table 3-17: GENERATE_PARTITION_SCHEME_SQL arguments

Argument	Datatype	Value	Description
operation	string	db_partition, add_partition, or exchange_partition	Defines the operation that you want to execute. The only values allowed are: db_partition, add_partition, and exchange_partition. Db_partition creates a script to partition the database, add_partition creates a script to add a partition to the database, and exchange_partition creates a script to exchange a partition.
type_name	string	*type_name*	Specifies a type to partition. Must be a supertype.
table_name	string	*regtable_name*	Specifies a registered table. The table must already have an i_partition column. Do not use this for repository-created tables; use the type_name argument for those tables.
owner_name	string	*owner_name*	The table owner name.
last_partition	string	*partition_name*	The name of the partition for objects whose i_partition value is larger than the highest range defined.
last_tablespace	string	*tablespace*	The tablespace of the last partition.
partition_name	string	*partition_name*	The name of the partition.

Argument	Datatype	Value	Description
range	integer	*integer*	Uppermost i_partition value of an object placed in this partition.
tablespace	string	*tablespace*	The tablespace for the partition.
temp_table_suffix	string	*temp_table_suffix*	The suffix for the temporary table. The default is "x".
include_object_type	Boolean	TRUE | FALSE	Whether to include the internal table dmi_object_type in partitioning. The table can only be included if it has already been partitioned.

3.20.4 Return Value

GENERATE_PARTITION_SCHEME_SQL returns an object id for the text file containing the SQL script to generate the partitioning.

3.20.5 Permissions

You must have Sysadmin or Superuser privileges to use this method.

In Oracle installations, you must be logged in as sysdba to run the script.

3.20.6 Description

Note: This method is deprecated, from the 6.6 release. This method is replaced by PARTITION_OPERATION.

Use this method to generate a database partitioning script. After the script is generated, it is run against the underlying database, using the database's SQL facility. For all versions of this method, except for add_partition, stop Documentum CM Server before you run the script against the database, then restart Documentum CM Server for the changes to take effect. For the add_partition method, you do not need to stop or restart Documentum CM Server.

The first form of the command, in which operation='db_partition', allows you to specify a type or a table to partition. If you do not specify a type or table, the generated script will partition all the partitionable types in the repository if it is run. You would use this command to partition a repository that was upgraded from an earlier version. The first range value means that objects with i_partition values from 0 to the first value will be in the first partition. The second partition will contain objects with i_partition values greater than first range value up to the second range value. Each subsequent partition will contain all the objects with i_partition values greater than the previous partition and up to its value. The last partition contains those objects with i_partition values greater than any previously specified partition.

The second form of the command, in which operation='add_partition', allows you to add partitions. This form is similar to the first form, but the first added partition begins with values greater than previously defined partitions. If there happen to be objects in the last partition whose i_partition values fall into the new partition's range, they will be moved into the new partition.

If you create a new repository with partitioning enabled, there is only one partition, called the last partition. You can then customize your repository by adding partitions. In this case, the first added partition range goes from 0 to the value you specified.

The final two versions, in which operation='exchange_partition', allow you to exchange a partition with a schema-identical offline table in the database tablespace. Commonly, you would use this feature to load a large number of objects into a table and then swap the table into the partition.

3.20.6.1 Oracle installations

In order to use this method with an Oracle installation, you must run the script as SYSDBA. For example, if dmadmin is the installation owner:

```
C:\Documents and Settings\dmadmin>sqlplus "/as sysdba"
@ C:\Documentum\data\testenv\content_storage_01\00000057\80\00\01\19.txt
```

Additionally, if you are partitioning a non-partitioned database, you may want to increase the number of open database cursors, or the script may exit with the error:

```
ORA-0100 Max Opened Cursors Exceeded
```

To increase the number of cursors, consult your Oracle documentation. It may tell you to use a command similar to:

```
ALTER SYSTEM SET OPEN_CURSORS=2000 SID='*' SCOPE=MEMORY;
```

or for Oracle versions earlier than Oracle 11:

```
ALTER SYSTEM SET OPEN_CURSORS=2000 SCOPE=MEMORY SID='*';
```

to alter the number of open cursors.

If you exit the script with an error (from inadvertently exceeding the number of open cursors, for example), correct the error, and rerun the script, you may see an error message like:

```
ORA-12091: cannot online redefine table "TECHPUBS"."DMC_WFSD_ELEMENT_S" with
materialized views
```

caused by leftover temporary items from the previous script failure. One way to correct this error is to run a command similar to:

```
execute DBMS_REDEFINITION.ABORT_REDEF_TABLE('<Schema Name>', '<Table Name>'
,'<Table Name>I');
```

Where <Table Name>I is the intermediate table name used for the redefinition. From the previous error message, we would use this command:

```
execute DBMS_REDEFINITION.ABORT_REDEF_TABLE('TECHPUBS', 'DMC_WFSD_ELEMENT_S'
,'DMC_WFSD_ELEMENT_SI');
```

3.20.6.2 SQL Server installations

If you are using a SQL Server installation, use filegroup names as values for the tablespace attributes, since SQL Server filegroups correspond to tablespaces. For example, use:

```
tablespace='filegroup'
```

and

```
last_tablespace='filegroup'
```

for SQL Server installations.

3.20.7 Partition exchange

Partition exchange must be carefully planned. Typically, you will create a number of offline tables to load with data, use whatever native database method is available to load the tables, create the table indexes, and then swap the tables into the prepared repository partition. This technique can load large amounts of data into a repository while causing a minimum of disruption to normal use of the repository. This technique can also be used to remove large amounts of data from a repository by swapping out a partition for small offline tables.

The typical steps you would take to do a partition exchange involve the following:

1. Identify the offline tables to create.

   The *OpenText Documentum Content Management - High-Volume Server Development Guide (EDCCS250400-DGD)* contains more information.

2. Load the offline tables.

   Load the tables with data using whatever methods are available to you with your database.

3. Create the offline index tables.

   The offline tables must index the same properties as the online objects do. The schema must be identical. Create the offline index tables in the same tablespace as the current online indexes.

4. Exchange the partition for the offline tables.

   Run the GENERATE_PARTITION_SCHEME_SQL administration method to generate the partitioning script, stop Documentum CM Server, run the script against the RDBMS, and restart Documentum CM Server. After following these steps, the data that was previously in the offline tables is in the online partition, and the previously online data is now in the offline table.

5. Validate the exchange.

   Check the online data to be sure that the exchange was successful.

3.20.8 Examples

This example generates a script to partition all the partitionable types in a repository. After the script is run, partitionable objects with an i_partition value of 0 to 10 will be stored in partition P1 of the database. Partitionable objects with an i_partition value of 11 to 20 will be stored in partition P2 of the database.

```
EXECUTE generate_partition_scheme_sql WITH
"partition_name"='P1',"range"=10,"tablespace"='dm_techpubs_docbase',
"partition_name"='P2',"range"=20,
"tablespace"='dm_techpubs_docbase'
```

To get the script, you can issue the following DQL statement (assume that the result of the earlier method returned the object ID 0855706080007c19):

```
EXECUTE get_file_url FOR '0855706080007c19' WITH "format"='text'
```

3.21 GET_FILE_URL

3.21.1 Purpose

Returns the URL to a particular content file.

3.21.2 Syntax

```
EXECUTE get_file_url FOR 'object_id'
WITH format='format_name'[,page=page_number,]
[page_modifier='value']
```

object_id is the object ID of the document that contains the content file.

3.21.3 Arguments

Table 3-18: GET_FILE_URL arguments

Argument	Datatype	Value	Description
format	string	*format_name*	Name of the content format, as specified in the format object.

Argument	Datatype	Value	Description
category	integer	*value*	The kind of items to retrieve. Valid values are: 1, for workflow tasks 2, for router tasks (obsolete) 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.
page_modifier	S	*value*	Identifies the rendition. Refer to the General Notes for a detailed description of the purpose of the page modifier.

3.21.4 Return value

GET_FILE_URL returns a collection consisting of one object with five properties. One property indicates the success or failure of the method call. If the call is successful, the other property values can be concatenated to compose the URL.

The properties are:

result	Boolean property 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 property is the base URL used by the Web server to retrieve the content. This value is retrieved from the base_url property of the storage area that contains the file.

store	The value in the store property is the name of the storage area that contains the file.
path	The value in the path property is a path to the file. The path is relative to the storage area identified in the store property.
ticket	Contains an encryption of the path value plus a time stamp.

3.21.5 Permissions

You must have at least Read permission on the object or Sysadmin privileges to use this method.

3.21.6 Description

Use GET_FILE_URL in an application when you want to access content using a Web server or a streaming server.

3.21.7 Related administration methods

None

3.21.8 Examples

```
EXECUTE get_file_url FOR '090000215400ac12'
WITH format='jpeg_th',page=0
```

3.22 GET_INBOX

3.22.1 Purpose

Returns items from an Inbox.

3.22.2 Syntax

```
EXECUTE get_inbox [WITH name='user_name'][,category=value]
[,batch=value]{,order_by='attr_name [asc|desc]'}]
```

3.22.3 Arguments

Table 3-19: GET_INBOX arguments

Argument	Datatype	Value	Description
name	string	*user_name*	User whose Inbox items you want to retrieve. The default value is the session user.
category	integer	*value*	The kind of items to retrieve. Valid values are: 1, for workflow tasks 2, for router tasks (obsolete) 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.
batch	integer	*value*	Number of items returned by each IDfCollection.next method issued against the collection returned by GET_INBOX. The default value is 0, meaning return all rows.

Argument	Datatype	Value	Description
order_by	string	*attr_name*	Property by which to order the returned items. The property must be a property 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.

3.22.4 Return value

GET_INBOX returns a collection containing the Inbox items in query result objects. The query result objects have 49 properties. The first 41 are the properties of the dmi_queue_item object representing the Inbox item. The property names in the query result object match the names of the properties of the queue items. The remaining eight properties identify the SysObject associated with the Inbox item, if any. Generally, these properties have values if the Inbox item is a workflow task. Notification items have no values in these eight properties. Refer to the General Notes for a more detailed explanation.

“SysObject-related property names for GET_INBOX query result objects” on page 234, lists the SysObject-related properties and their corresponding SysObject property.

Table 3-20: SysObject-related property names for GET_INBOX query result objects

Query Result Property Name	Associated SysObject Property
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
pkg_lock_owner	r_lock_owner

Query Result Property Name	Associated SysObject Property
pkg_is_virtual_doc	r_is_virtual_doc

3.22.5 Permissions

Any user can issue this method to return either his or her own Inbox items or the Inbox items of another user.

3.22.6 Description

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 property values for the multiple objects will be the same. Only the values of the eight SysObject-related properties will be different.

The eight SysObject-related properties 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 repository.
- 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.
- The Inbox item represents an object in a remote repository.

3.22.7 Related administration methods

None

3.22.8 Examples

This example returns all tasks and notifications for the user issuing the method:

```
EXECUTE get_inbox WITH category=7
```

3.23 GET_LAST_SQL

3.23.1 Purpose

Retrieves the last SQL statement that was executed by the session in which this RPC is run.

3.23.2 Syntax

```
EXECUTE get_last_sql [WITH detail_info=true|false]
```

3.23.3 Arguments

Table 3-21: GET_LAST_SQL arguments

Argument	Datatype	Value	Description
detail_info	boolean	*TRUE | FALSE*	Whether to get all the recorded SQL statements and weather to enclose each statement within a pair of XML tags showing the index number. When the detail_info argument is not introduced, GET_LAST_SQL only returns the last SQL statement; when you set detail_info to TRUE, GET_LAST_SQL not only returns all the recorded SQL statements, but also enclose each statement within a pair of XML tags showing the current index number; for example, the first statement is enclosed within <0> and </0>.

3.23.4 Return value

GET_LAST_SQL returns a collection with one query result object by default. The result object has one property whose value is the last SQL statement. To see the statement, issue a Next on the collection and then dump the collection.

When the detail_info argument is not introduced, GET_LAST_SQL only returns the last SQL statement; when you set detail_info to TRUE, GET_LAST_SQL not only returns all the recorded SQL statements, but also enclose each statement within a pair of XML tags showing the current index number; for example, the first statement is enclosed within <0> and </0>.

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.

3.23.5 Permissions

Any one can issue this method.

3.23.6 Description

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" on page 340, for instructions.

3.23.7 Related administration methods

None

3.23.8 Examples

```
EXECUTE get_last_sql
```

3.24 GET_PATH

3.24.1 Purpose

Returns the directory location of a content file stored in a distributed storage area.

3.24.2 Syntax

```
EXECUTE get_path [FOR] 'content_obj_id'
[WITH store = 'value']
```

3.24.3 Arguments

Table 3-22: GET_PATH arguments

Argument	Datatype	Value	Description
store	string	*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.

3.24.4 Return value

Returns the directory path of the specified content file.

3.24.5 Permissions

Anyone can use this method.

3.24.6 Description

In a Documentum CM Server configuration, the GET_PATH function is executed by the Documentum CM 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.

3.24.7 Related administration methods

None

3.24.8 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'
```

3.25 GET_SESSION_DD_LOCALE

3.25.1 Purpose

Returns the locale in use for the current session.

3.25.2 Syntax

```
EXECUTE get_session_dd_locale
```

3.25.3 Arguments

None

3.25.4 Return value

The method returns a collection with one result object. The result object has one property, named dd_locale. The value of this property is the locale for the session.

3.25.5 Permissions

Anyone can use this method.

3.25.6 Description

None

3.25.7 Related administration methods

None

3.25.8 Examples

```
EXECUTE get_session_dd_locale
```

3.26 HTTP_POST

3.26.1 Purpose

Sends an HTTP_POST request invoking a Java servlet.

3.26.2 Syntax

```
EXECUTE http_post WITH app_server_name='name'
[,arguments=argument_list][,save_response=value]
[,time_out=value][,launch_asynch=value][,trace_launch=value]
```

3.26.3 Arguments

Table 3-23: HTTP_POST arguments

Argument	Datatype	Value	Description
app_server_name	string	*name*	Name of the Java servlet. This must match a servlet identified in the app_server_name property of the server config object.
arguments	string	*argument list*	Defines the command line arguments passed to the servlet or Java method. UTF-8 characters are acceptable for the argument strings.

Argument	Datatype	Value	Description
save_response	integer	*integer*	Indicates whether to save the response in a document in the repository. 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	integer	*integer*	The length of time, in seconds, to wait for a response to the HTTP_POST request. The default is 60 seconds.
launch_async	Boolean	true | 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	Boolean	true | 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.

3.26.4 Return value

HTTP_POST returns a collection with one query result object that has seven properties. "Query result object properties for HTTP_POST" on page 242, lists the properties:

Table 3-24: Query result object properties for HTTP_POST

Property 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 Error 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.
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 Documentum CM 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.

3.26.5 Permissions

You must have Superuser privileges to execute this method. Refer to "Preserving security" on page 243, for more information about security when using this method.

3.26.6 Description

Use HTTP_POST to invoke a servlet or Java method installed in an application server.

If you set LAUNCH_ASYNC to TRUE, Documentum CM 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. Documentum CM Server encodes the arguments using application/x-www-form-urlencoded format. For example, suppose you issued the following HTTP_POST method:

```
EXECUTE http_post WITH app_server_name='payroll',
save_response=true,trace_launch=true,
arguments='-docbase accounting -user paymaster
-ticket DM_TICKET=0000000222a02054.accounting@host01
-document "weekly record"'
```

Documentum CM Server sends the arguments in the following format:

```
docbase=accounting&user=paymaster&ticket=DM_TIICKET%
3D0000000222a02054.accounting%4Dhost01&document=weekly+record
```

3.26.6.1 Preserving security

The application server, all servlets that it invokes, and the methods invoked by the servlets run as the Documentum CM 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 CM Server. When you write a servlet, you must include security checking to ensure that the request comes from a Documentum CM Server. One recommended way is have the servlet ensure that the generated request comes from a machine that hosts a Documentum CM 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 Documentum CM Server installation for information about how that is set up.

On Windows platforms, the current operating system user is allowed to log in to the repository through Documentum CM Server without providing a password. Consequently, a servlet or an invoked Java method can log into the repository with Superuser privileges without providing a password.

If you write a servlet or method that uses this manner of login, 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.

3.26.6.2 Sample servlet and method

OpenText Documentum CM 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. The WEB-INF folder was set up when you installed the application server.

3.26.6.3 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 repository.

3.26.7 Related administration methods

"DO_METHOD" on page 207

3.26.8 Examples

The following examples send an HTTP_POST request to the Java servlet called DevAppSrv. The content of the request passes a repository 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"'
```

3.27 IMPORT_REPLICA

3.27.1 Purpose

Imports files from one distributed storage area into another distributed storage area.

3.27.2 Syntax

```
EXECUTE import_replica FOR 'content_object_id'
WITH store='storage_name',file='file_name'
[,other_file='other_file_name']
```

3.27.3 Arguments

Table 3-25: IMPORT_REPLICA arguments

Argument	Datatype	Value	Description
store	string	*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	string	*file_name*	Identifies the file to replicate.
other_file	string	*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.

3.27.4 Return value

IMPORT_REPLICA returns a collection with one query result object. The object has one property whose value indicates success (TRUE) or failure (FALSE).

3.27.5 Permissions

You must have Sysadmin or Superuser privileges to use this method.

3.27.6 Description

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 Documentum Servers, you must issue a connection request that bypasses the Documentum CM Server to use IMPORT_REPLICA in the session.

3.27.7 Related administration methods

"CAN_FETCH" on page 188
"DELETE_REPLICA" on page 204
"GET_PATH" on page 237
"REPLICATE" on page 327

3.27.8 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'
```

3.28 IMPORT_TICKET_KEY

3.28.1 Purpose

Imports a login ticket key into a repository.

3.28.2 Syntax

```
EXECUTE import_ticket_key
WITH KEY_STRING='string',
PASSWORD='password'
```

3.28.3 Arguments

Table 3-26: IMPORT_TICKET_KEY arguments

Argument	Datatype	Value	Description
key_string	string	*string*	The ASCII-encoded string returned by an EXPORT_TICKET_KEY method.
password	string	*password*	Password used when the key was exported.

3.28.4 Return value

This method returns a collection with one query result object. The object has one property, named result, that contains T (TRUE) if the method was successful or F (FALSE) if unsuccessful.

3.28.5 Permissions

You must have Superuser privileges to execute this method.

3.28.6 Description

Use IMPORT_TICKET_KEY to import a login ticket key into a repository. The key must have been exported from a repository using the EXPORT_TICKET_KEY method. When you import the key, the password argument in IMPORT_TICKET_KEY must be the same password that you used in the EXPORT_TICKET_KEY method.

You must restart Documentum CM Server after importing a login ticket key to make the key take effect.

Keys are typically exported from one repository and imported into another as part of the configuration process when setting up trusted repositories, to allow use of global login tickets or tokens. The *OpenText Documentum Content Management - Server Fundamentals Guide (EDCCS250400-GGD)* contains the description of trusted repositories, global login tickets, and global tokens.

3.28.7 Related administration methods

"EXPORT_TICKET_KEY" on page 222
"RESET_TICKET_KEY" on page 329

IDfSession.importTicketKey()

3.28.8 Example

```
EXECUTE import_ticket_key
WITH KEY_STRING='ticket_string',
PASSWORD='myword'
```

3.29 LIST_AUTH_PLUGINS

3.29.1 Purpose

Lists the authentication plug-ins that the server has loaded.

3.29.2 Syntax

```
EXECUTE list_auth_plugins
```

3.29.3 Arguments

None

3.29.4 Return value

The method returns a collection with one query result object. The object has two repeating string properties, plugin_id and plugin_filepath. The values in plugin_id are the unique plug-in identifiers and the values in plugin_filepath are the full paths of the plug-ins. The values at corresponding index positions represent one plug-in.

3.29.5 Permissions

You must have Sysadmin or Superuser privileges to execute this method.

3.29.6 Description

This method is useful only at sites with OpenText™ Documentum™ Content Management Trusted Content Services because the ability to use authentication plug-ins is a feature of OpenText Documentum Content Management (CM) Trusted Content Services.

3.29.7 Related administration methods

None

3.29.8 Examples

Refer to the syntax.

3.30 LIST_RESOURCES

3.30.1 Purpose

Lists a variety of information about the server's operating system environment and the server.

3.30.2 Syntax

```
EXECUTE list_resources [WITH reset=true|false]
```

3.30.3 Arguments

Table 3-27: 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.

3.30.4 Return value

LIST_RESOURCES returns a collection with one query result object. On Windows platforms, the query result object has twelve properties, described in "Collection properties for LIST_RESOURCES" on page 250.

Table 3-28: Collection properties for LIST_RESOURCES

Property	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 (Linux).
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.
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 0 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.

Property	Datatype	Description
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 Linux, the result object has eight properties, described in "Collection properties for LIST_RESOURCES" on page 251.

Table 3-29: Collection properties for LIST_RESOURCES

Property	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 (Linux).

Property	Datatype	Description
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.

3.30.5 Permissions

Anyone can use this method.

3.30.6 Description

LIST_RESOURCES lists a variety of information about the server operating system environment and the server (the information is described in "Return value" on page 250).

3.30.7 Related administration methods

"DB_STATS" on page 202
"LIST_SESSIONS" on page 253
"LIST_TARGETS" on page 256
"SHOW_SESSIONS" on page 345

3.30.8 Examples

The following examples return the resources information and reset the heap size counters:

```
EXECUTE list_resources WITH reset=true
```

3.31 LIST_SESSIONS

3.31.1 Purpose

Returns the sessions of the currently connected server (not all servers serving the repository).

3.31.2 Syntax

```
EXECUTE list_sessions[WITH brief_info=true|false]
```

3.31.3 Arguments

Table 3-30: LIST_SESSIONS arguments

Argument	Datatype	Value	Description
BRIEF_INFO	B	T (TRUE) or F (FALSE)	Although the method will accept this argument, setting the value has no effect. Regardless of the value set, the properties listed in the complete information table are returned. The default is FALSE.

3.31.4 Return value

LIST_SESSIONS returns a collection with one result object that describes the current active session.

3.31.4.1 If BRIEF_INFO is FALSE

"Complete information returned by LIST_SESSIONS" on page 254 lists the information returned for either value of BRIEF_INFO.

Table 3-31: Complete information returned by LIST_SESSIONS

Property	Description
root_start	The time when the main server thread (root server process) was started.
root_pid	On Windows, the process ID of the Documentum CM Server process. On Linux, the process ID of the root Documentum CM Server process.
shared_mem_id	The ID of the shared memory segment used by the servers. This property is returned only on Linux platforms.
semaphore_id	The ID of the semaphore used by the servers. This property is returned only on Linux platforms.
session	The object ID of the session begun by user_name.
db_session_id	The ID of the database session.
typelockdb_session_id	The database session ID for type locking.

Property	Description
tempdb_session_ids	List of temporary database sessions.
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 Foundation Java API 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.
last_rpc	The last RPC the server ran for the session.
current_rpc	The current RPC being run by the server for the session. Note that after the RPC is completed the server will not clear this field. Therefore, if last_rpc and current_rpc have the same value, either the server has completed an RPC for the session and has not executed another RPC, or the server is running the same RPC again.
last_completed_rpc	The last time the server completed an RPC for the session.

3.31.5 Permissions

Anyone can use this method.

3.31.6 Description

LIST_SESSIONS returns a collection of a single query result object whose properties contain information about the current active session.

3.31.7 Related administration methods

"SHOW_SESSIONS" on page 345

3.31.8 Examples

This example executes LIST_SESSIONS:

```
EXECUTE list_sessions
```

3.32 LIST_TARGETS

3.32.1 Purpose

Lists the connection brokers to which the server is currently projecting.

3.32.2 Syntax

```
EXECUTE list_targets
```

3.32.3 Arguments

LIST_TARGETS has no arguments.

3.32.4 Return value

LIST_TARGETS returns a collection with one query result object. The values across the properties at one index position represent the information about one connection broker to which the server is projecting.

Table 3-32: Query result object properties for LIST_TARGETS

Property	Datatype	Description
projection_targets	string(80)	A repeating property that contains the names of the hosts on which the target connection brokers are running.

Property	Datatype	Description
projection_proxval	integer	A repeating property that contains the proximity value projected to the connection broker identified in the corresponding index position in projection_targets.
projection_notes	string(80)	A repeating property that contains any user-defined note for the connection broker defined at the corresponding index position in projection_targets and projection_proxval.
docbroker_status	string(64)	A repeating property that records whether the connection broker identified in projection_targets at the corresponding index position is available. Valid values are: Available and Unavailable.
comments	string(64)	A repeating property that records whether the projection target entry was taken from the server config object or the server.ini file.

3.32.5 Permissions

Anyone can use this method.

3.32.6 Description

A server's connection broker targets are those connection brokers to which the server is sending checkpoint messages. Target connection brokers 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.

3.32.7 Related administration methods

None

3.32.8 Examples

The following examples list the current connection broker targets for the server:

```
EXECUTE list_targets
```

3.33 LOG_OFF

3.33.1 Purpose

Turns off the RPC logging.

3.33.2 Syntax

```
EXECUTE log_off
```

3.33.3 Arguments

LOG_OFF has no arguments.

3.33.4 Return value

LOG_OFF returns a collection with one query result object. The object has one Boolean property whose value indicates the success (TRUE) or failure (FALSE) of the operation.

3.33.5 Permissions

You must have Sysadmin or Superuser privileges to use this method.

3.33.6 Description

None

3.33.7 Related administration methods

"LOG_ON" on page 259

3.33.8 Examples

The following example turns off RPC logging:

```
EXECUTE log_off
```

3.34 LOG_ON

3.34.1 Purpose

Turns on logging for RPC calls.

3.34.2 Syntax

```
EXECUTE log_on [WITH detail=true|false]
```

3.34.3 Arguments

Table 3-33: 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.

3.34.4 Return value

LOG_ON returns a collection with one query result object. The object has one Boolean property whose value indicates the success (TRUE) or failure (FALSE) of the operation.

3.34.5 Permissions

You must have Sysadmin or Superuser privileges to use this method.

3.34.6 Description

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.

3.34.7 Related administration methods

"LOG_OFF" on page 258

3.34.8 Examples

These examples turn on detailed RPC logging:

```
EXECUTE log_on WITH detail=true
```

3.35 MAKE_INDEX

3.35.1 Purpose

Creates an index for a persistent object type.

3.35.2 Syntax

```
EXECUTE make_index
WITH type_name='object_type',attribute='property_name'
{,attribute='property_name'}
[,unique=true|false][,index_space='name'][,index_name='name']
[,use_id_col=true|false][,use_pos_col=true|false]
[,global_index=true|false]
[,PARALLEL_DEGREE=<integer value>][,NOLOGGING=[true|false][,ONLINE =true|false]
```

3.35.3 Arguments

Table 3-34: 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.

Argument	Datatype	Value	Description
ATTRIBUTE	S	*property_name*	Identifies the property or properties on which to build the index. You can specify multiple properties, but you cannot mix single-valued and repeating properties in the same statement.
UNIQUE	B	TRUE or FALSE	Indicates whether to create a unique index. The default is FALSE.
INDEX_SPACE	S	Name of tablespace or segment	Specifies the tablespace or segment in which to store the index. If unspecified, the default is the tablespace or segment associated with the repository.
INDEX_NAME	S	Name of index table	Specifies the name of the index table. If unspecified, the system generates this name.
USE_ID_COL	B	TRUE or FALSE	Indicates whether to include the r_object_id column value in the index. The default is FALSE.
USE_POS_COL	B	TRUE or FALSE	Indicates whether to include the i_position column value in the index. The default is FALSE.
GLOBAL_INDEX	B	TRUE or FALSE	Indicates whether the index being created is a global index. This parameter can only be used for partitioned types. The default is FALSE.

Argument	Datatype	Value	Description
PARALLEL_DEGREE	integer	*integer value*	Indicates the degree of parallelism to be used by the underlying Oracle database. The index creation runs faster depending on the number of CPUs, table partitioning and disk fragmentation of the database.
NOLOGGING	B	TRUE or FALSE	This argument helps in faster insertion and index creation. This bypasses the writing of the redo log helping in improving performance. The default is TRUE.
ONLINE	B	TRUE or FALSE	Indicates whether the index being created online. During an online index rebuild, Oracle takes a snapshot log on the target table to hold DML activity, read the table in a full-table scan (read consistent), build the new index and then apply the changes from the snapshot log after the index has been rebuilt. During a regular index rebuild, an exclusive lock occurs as the existing index is read. This command is designed for scheduled downtime periods where there is no DML activity. The default is FALSE.

3.35.4 Return value

MAKE_INDEX returns a collection with one query result object. The object has one property, 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 property specified did not exist, the result property contains F (FALSE).

If the specified index already exists, the result property contains '0000000000000000'.

3.35.5 Permissions

You must have Superuser privileges to use this method.

3.35.6 Description

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 properties. However, if you specify multiple properties, you must specify all single-valued properties or all repeating properties. You cannot mix single and repeating valued properties in the same argument list. This is because the underlying RDBMS stores an object's single and repeating properties in separate tables.

If you specify multiple properties, the sort order within the index corresponds to the order in which the properties are specified in the statement. To include the r_object_id column in the index, include the USE_ID_COL argument.

The properties you specify must be defined for the type you specify. They cannot be inherited properties. For example, if you want to create an index on the subject and title properties, you would specify dm_sysobject in TYPE_NAME, as these properties are defined for dm_sysobject. The*OpenText Documentum Content Management - Server System Object Reference Guide (EDCCS250400-ORD)* lists the properties defined for each object type.

A unique index is an index in which the values in the columns to be indexed are unique within the index. If you include the UNIQUE argument, the RDBMS server enforces the uniqueness.

GLOBAL_INDEX is a boolean parameter that indicates whether the index being created is a global index. The default value is FALSE. This parameter can be used only when the type is a partitioned type, and is only useful if UNIQUE=TRUE is also specified. If an index of a partitioned table is a unique local index, it must include the i_partition column as part of the index. Otherwise, the database server will return an error. This is a requirement of the database for unique local indexes in a partitioned table. Since i_partition is an internal attribute, both single value tables and repeating value tables have this column. The internal attribute i_partition will not be included for unique indexes created for a partitioned type if GLOBAL_INDEX is TRUE.

Note: If you plan to use partition exchange, do not create global indexes on the types to use for the exchange. The *OpenText Documentum Content Management - High-Volume Server Development Guide (EDCCS250400-DGD)* contains more details on partition exchange.

3.35.7 Related administration methods

“DROP_INDEX” on page 217
“FINISH_INDEX_MOVES” on page 223
“MOVE_INDEX” on page 297

3.35.8 Examples

This example creates an index for the dm_sysobject object type, indexing on the subject and title properties:

```
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 property r_creator_name:

```
EXECUTE make_index WITH type_name='dm_sysobject',
attribute='r_creator_name'
```

The next example creates an index for the dm_sysobject type on the properties r_creator_name and r_creation_date:

```
EXECUTE make_index WITH type_name='dm_sysobject',
attribute='r_creator_name',
attribute='r_creation_date'
```

3.36 MARK_AS_ARCHIVED

3.36.1 Purpose

Sets the i_is_archived property of an audit trail entry to TRUE.

3.36.2 Syntax

```
EXECUTE mark_as_archived FOR audit_obj_id
```

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.

3.36.3 Arguments

This method has no arguments.

3.36.4 Return value

The method returns TRUE if successful or FALSE if unsuccessful.

3.36.5 Permissions

You must have Superuser privileges to execute this method.

3.36.6 Description

Use this method to set an audit trail entry's i_is_archived property to TRUE after you archive the entry.

3.36.7 Related administration methods

"PURGE_AUDIT" on page 310

3.36.8 Examples

```
EXECUTE mark_as_archived FOR 5f000012ae6217ce
```

3.37 MARK_FOR_RETRY

3.37.1 Purpose

Finds queue items representing events queued to the Index Agent that are in the acquired or failed state and resets them to the pending state.

3.37.2 Syntax

```
EXECUTE mark_for_retry
WITH NAME = 'index_name'
```

3.37.3 Arguments

Table 3-35: 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.

3.37.4 Return value

MARK_FOR_RETRY returns a collection with one query result object. The object has one Boolean property whose value indicates the success (TRUE) or failure (FALSE) of the operation.

3.37.5 Permissions

You must have Sysadmin or Superuser privileges to use this method.

3.37.6 Description

Use MARK_FOR_RETRY in the recovery procedure if you encounter problems with the index agent. The method scans the repository to find queue items representing events queued to the index agent that are in the acquired or failed state and resets them to the pending state.

The index agent registers itself for certain events on all SysObjects. The events serve as notice to the index agent that the objects need to be indexed or updated in the index. If there is a problem with the index agent, for example, if it crashes, some of the queue items representing those events may be left in the acquired or failed state. MARK_FOR_RETRY is used to reset those events to pending so that the index agent will pick up those events and properly process the objects after the problem is resolved.

3.37.7 Related administration methods

None

3.37.8 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'
```

3.38 MIGRATE_CONTENT

3.38.1 Purpose

Moves content files from one storage area to another.

3.38.2 Syntax

To migrate a single object:

```
EXECUTE migrate_content [FOR]object_id
WITH target_store='target_storage_name'
[,renditions=value][,remove_original=TRUE|FALSE][,log_file='log_file_path']
[,log_level=value][,source_direct_access_ok=T[,direct_copy=T
    [,update_only=T,command_file_name='command_file_name']]
|,source_direct_access_ok=T[,direct_move=T
    [,update_only=T,command_file_name='command_file_name']]
]
```

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'
[,log_level=value][,max_migrate_count=value][,batch_size=value]
[,remove_original=TRUE|FALSE][,parallel_degree=value]
[,source_direct_access_ok=T[,direct_copy=T
    [,update_only=T,command_file_name='command_file_name']]
|,source_direct_access_ok=T[,direct_move=T
    [,update_only=T,command_file_name='command_file_name']]
]
```

To migrate a set of objects identified by a DQL query:

```
EXECUTE migrate_content WITH
target_store='target_storage_name',query='DQL_predicate'[,sysobject_query=T
[,type_to_query='type_name']],log_file='log_file_path'
[,log_level=value][,renditions=value][,max_migrate_count=integer][,batch_size=value]
[,remove_original=TRUE|FALSE][,parallel_degree=value]
[,source_direct_access_ok=T[,direct_copy=T
    [,update_only=T,command_file_name='command_file_name']]
|,source_direct_access_ok=T[,direct_move=T
    [,update_only=T,command_file_name='command_file_name']]
]
```

3.38.3 Arguments

Table 3-36: MIGRATE_CONTENT arguments

Argument	Datatype	Value	Description
object_id	ID	*content_object ID*or *SysObject ID*	When migrating a single content file, you can identify the file using its content object ID or the object ID of the SysObject (or SysObject subtype) that contains the content file. Specifying a content object ID requires Superuser privileges. Specifying a SysObject object ID requires Write permission on the object.
SOURCE_STORE	S	*source_storage_name*	Name of the storage area whose content you wish to migrate. The storage area may be a file store, a ca store, a blobstore, a distributed store, or an external store. Use the name of the storage area’s object.

Argument	Datatype	Value	Description
TARGET_STORE	S	*target_storage_name*	Name of the storage area to which you are migrating the content. The storage area may be a file store, a ca store, or a distributed store. Use the name of the storage area's object. Blobstores and external stores cannot be specified as a target storage area, and ca stores cannot be specified if DIRECT_MOVE is specified when migrating from an external store.
RENDITIONS	S	all, primary, or secondary	Indicates which content files associated with the SysObject or SysObjects you wish to move. This argument is used only when *object_id* references a SysObject or when SYSOBJECT_QUERY is set to T. Valid values are: *all*, meaning move the first primary content file (page 0) and its renditions *primary*, meaning move only the first primary content file (page 0) *secondary*, meaning move only the renditions of the first primary file (page 0) The default is primary.

Argument	Datatype	Value	Description
REMOVE_ORIGINAL	B	T (TRUE) or F (FALSE)	Whether to remove the content file from the source storage area. T directs Documentum CM Server to remove the original content file if possible. F directs Documentum CM Server not to remove the content. If set to F, you must include the LOG_FILE argument also.
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.
LOG_LEVEL	I	*value*	Level of detail in the log file. Valid values are: • 1: full tracing (recommended). • 2: print out warning messages and above. • 3: print out only error messages. The default is 1.
MAX_MIGRATE_COUNT	I	*value*	Defines the maximum number of content files to migrate.

Argument	Datatype	Value	Description
BATCH_SIZE	I	*value*	Defines how many content files to include in a single transaction during the migration operation (or how many files per worker session if PARALLEL_DEGREE is greater than 1). The default is 500.
PARALLEL_DEGREE	I	*value*	Defines how many content migration sessions are created to do the migration. The default value is 0, indicating that a single session is created and migration will not be done in parallel. The maximum value default is 10, but can be modified by setting the max_cm_parallel_degree parameter in dm_server_config. The valid range for this parameter is 2 to 50. PARALLEL_DEGREE is set to a value greater than max_cm_parallel_degree, the server will use the max_cm_parallel_degree value instead.

Argument	Datatype	Value	Description
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. The predicate is applied to either content objects or SysObjects, depending on the setting of the SYSOBJECT_QUERY argument. For more information, refer to the Usage Notes.
SYSOBJECT_QUERY	B	T (TRUE) or F (FALSE)	T directs the method to apply the DQL predicate specified in the QUERY argument to SysObjects or the subtype identified in TYPE_TO_QUERY. F directs the method to apply the predicate to content objects. The default is F.
TYPE_TO_QUERY	S	*type_name*	Name of a SysObject subtype. You can include this only if SYSOBJECT_QUERY is set to T. If specified, the DQL predicate is applied to the subtype, rather than dm_sysobject type. For information about the uses of this argument, refer to “Including a query” on page 280.

Argument	Datatype	Value	Description
SOURCE_DIRECT_ACCESS_OK	B	T (TRUE) or F (FALSE)	Indicates whether the files in this storage area can be accessed directly using a path available in Documentum CM Server. Source store must be an external store. The default is F.
DIRECT_COPY	B	T (TRUE) or F (FALSE)	Indicates whether to migrate the content by using the underlying OS file system command to directly copy the content to a new location. Source store must be an external store. The default is F.
DIRECT_MOVE	B	T (TRUE) or F (FALSE)	Indicates whether to migrate the content by using the underlying OS file system command to directly move the content to a new location. Source store must be an external store. The default is F.

Argument	Datatype	Value	Description
UPDATE_ONLY	B	T (TRUE) or F (FALSE)	Indicates whether to migrate the content, or to instead save the OS file system commands required to migrate the content into a command file for later execution. If UPDATE_ONLY is T, the content is not migrated, but the saved commands in the command file must be executed later. Source store must be an external store. The default is F.
COMMAND_FILE_NAME	S	*command_file_name*	Specifies the name of the file used to store the OS file system commands required to migrate the content. Supply this argument when UPDATE_ONLY is set to T. Source store must be an external store. There is no default for this parameter, so you must supply a file name.

Argument	Datatype	Value	Description
IGNORE_CONTENT_VALIDATION_FAILURE	B	T (TRUE) or F (FALSE)	Indicates if the content migration occurs on failed content validation. For example, if 1000 content objects are migrated and the second object fails the content validation, then the following action occurs based on the value of the parameter: • F: Migration is aborted and the remaining 998 objects are not migrated. • T: Failed content objects are ignored and the remaining 998 objects are migrated. The default is F.
QUERY_BATCH_SIZE	I	*value*	Specifies the number of objects processed per query. The default is 1000.

3.38.4 Return value

The method returns a collection with one query result object. The object has one integer property, named result, whose value is the number of contents successfully migrated.

3.38.5 Permissions

Superuser permissions are required to:

- Specify a content object ID when moving a single content file.
- Move all content from one storage area to another.
- Move content using a DQL predicate that operates on content objects (SYSOBJECT_QUERY=F).

Write permission on the object is required to:

- Specify a SysObject ID when moving the content of a single object.
- Move content using a DQL predicate that operates on a SysObject (SYSOBJECT_QUERY=T).

3.38.6 Description

This section contains usability information.

3.38.6.1 General notes

MIGRATE_CONTENT is the preferred way to move content files from a file store, ca store, blobstore, distributed store, or an external store storage area to a file store, ca store, or distributed store storage area. You cannot move content to a blob store or external store storage area. Additionally, you cannot use this method to move files to or from turbo storage. If content is stored in a retention-enabled ca store storage area, you can only move the content to another retention-enabled ca store storage area. If you move a file to a distributed storage area, the file is placed in the distributed component local to the Documentum CM Server to which you are connected when executing the method.

The method moves content files associated with immutable objects as well as changeable objects. It does not update a SysObject's r_modify_date property when the object's content is moved. 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.

The method operates on dmr_content objects, resetting their storage_id property to the new storage area and resetting the data ticket property. It also updates the i_vstamp property. The i_vstamp property 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.

The method is aware of lightweight SysObjects (LWSOs), so you can move LWSOs using this method without materializing the objects.

Here are some general guidelines for using MIGRATE_CONTENT:

- If you want to move content to or from a distributed storage area, you must specify the dm_distributedstore object as the target or source storage area—do not specify the actual distributed component.
- Make sure 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.
- If the content to be migrated is stored in a content-addressed storage area with a retention date, make sure the retention date has expired. If not, the method fails with an error.
- If the content's current storage area and the target storage area are the same content-addressed storage area and the storage area has a retention period, the method updates the content's metadata and retention period and creates a new address for the content. However, the content is not physically moved.
- Back up the repository 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 choose not to remove the original content, that content becomes an orphaned content file and can be removed using dmfilescan.
- If the MIGRATE_CONTENT method fails with an error, the entire batch transaction is rolled back. If the destination has content files that were created from the successful migrations within the batch, you can clean up those files by running the Dmfilescan job.

3.38.6.2 Migrating from an external store

Content stored in an external store is accessed through a custom plug-in that takes the request from Documentum CM Server and returns the content. Typically, the content stored in an external filestore was generated by some other application, or managed by a legacy content management system, and at some point, management was transferred to Documentum CM Server. The storage plug-in is responsible for translating the content storage location specified by Documentum CM Server into the location of the content in the storage device and retrieving the content.

In many cases, the content in an external store is accessed through the underlying file system, but is stored in a different file storage scheme than a standard Documentum CM Server filestore. In these cases, Documentum CM Server knows the path to content files in the external storage area. Typically, migrating content directly, using the underlying operating system facilities, is much more efficient than fetching the file to Documentum CM Server and writing it out to the target storage location. In some cases, the target store is on the same physical device, so that migrating content, in effect, can be done by restructuring the paths to the content files without making copies. In most operating systems, when you move a file from one location to another on the same physical device, the file does not move, but the directory structure pointing to it changes.

To support this more efficient migration for external storage areas, some arguments are only supported for content migrated from an external store:

- SOURCE_DIRECT_ACCESS_OK
- DIRECT_COPY
- DIRECT_MOVE
- UPDATE_ONLY
- COMMAND_FILE_NAME

The argument, SOURCE_DIRECT_ACCESS_OK, indicates that the path to access the content file is available in Documentum CM Server and can be used to access the file. If you select this argument, you can optionally also select either DIRECT_MOVE or DIRECT_COPY. If you select DIRECT_MOVE or DIRECT_COPY, Documentum CM Server will use the file path and the OS file system move or copy command to migrate the specified content files to their new filestore. If you do not select DIRECT_MOVE or DIRECT_COPY, Documentum CM Server will use the underlying OS system calls to read in the content files and write them out to their new location. You cannot use DIRECT_MOVE or DIRECT_COPY if the target store is configured for compression, encryption, or de-duplication.

Setting SOURCE_DIRECT_ACCESS_OK to TRUE and not setting DIRECT_MOVE or DIRECT_COPY to TRUE is useful when migrating content from an external store to a CA store. In this case, the content in external store is directly accessed by the Centera SDK to pump the content to Centera. Setting SOURCE_DIRECT_ACCESS_OK to TRUE and setting DIRECT_COPY to TRUE is useful in migrating content from an external store to a filestore, where efficiency of content copying performed by the OS is leveraged, instead of chunking the content. Note that in this case, you will have to provision double the storage capacity (the disk space for content in the original store plus the disk space for content in the destination file store). Setting SOURCE_DIRECT_ACCESS_OK to TRUE and setting DIRECT_MOVE to TRUE enables the server to move/rename the file instead of copying it, useful in migrating the content from an external store to a filestore on the same volume, without having to provision the double the storage.

If you select UPDATE_ONLY, the content is not migrated, but the operating system commands to do so are saved in the file specified by COMMAND_FILE_NAME and must be executed later (you must specify this file name if you select UPDATE_ONLY). After you have run the MIGRATE_CONTENT method with UPDATE_ONLY selected, the content specified to be migrated will not be accessible until the commands in the command file are executed.

3.38.6.3 Migrating XML content to an OpenText™ Documentum™ Content Management XML Store

XML content files can be moved into an OpenText Documentum Content Management (CM) XML Store using the MIGRATE_CONTENT method. This transactional method moves content files associated with immutable objects as well as changeable objects. 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. You can move files from a file store, a ca store, a blobstore, or a distributed store.

Note: The MIGRATE_CONTENT method can be used to migrate content between different XML Store or from an XML Store to a file store. However, it is not possible to move content that was previously moved from an XML Store to a file store to another file store.

3.38.6.3.1 XML content migration restrictions

The following restrictions apply for migrating XML content:

- The XML content must be well-formed XML.

  If the content is not well-formed, XML Store throws an error specifying the content parts that are not well-formed.
- Migrating XML documents into the XML Store only works with XML documents that are stored in a single page. If you have legacy content that is stored in the two-page model, you need to convert the content into the single-page model before you start the migration process, as described in "Converting XML content to single-page format" on page 279.
- The argument SOURCE_DIRECT_ACCESS_OK is not supported for XML Stores.

3.38.6.3.2 Converting XML content to single-page format

XML content that is stored in two-page format must be converted into single-page format before the content can be migrated into an XML Store.

To convert XML content into single-page format:

1. Change the *oldest_client_version* attribute of the *dm_docbase_config* object of the repository to *"5.3"*.
2. Check out the XML content that needs to be converted using Foundation Java API/Webtop. Repeat this step for each XML object that needs to be converted.
3. Check in the XML content that was checked out in Step 2.

3.38.6.4 Migrating to content-addressed storage

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

If you are moving content files to a content-addressed storage area that requires a retention date, you must set the retention date for the content before migrating the content. Use either a setContentAttrs method or the SET_CONTENT_ATTRS administration method to set the retention date.

Note: A content-addressed storage area requires a retention date if:

- The storage area's a_retention_attr_name property is set and the a_default_retention_date is not null, or
- The a_retention attr_name property is set and the a_retention_attr_required property is set to T

3.38.6.5 Including a query

You can include a DQL predicate in the method arguments. Depending on the value of the SYSOBJECT_QUERY argument, the predicate is run against instances of a specified object type or against content objects.

SYSOBJECT_QUERY is F (FALSE) by default, meaning that the predicate is applied to content objects. You must have Superuser permissions to run the predicate against content objects. When you run the query against content objects, it is recommended that you use a predicate that references a unique value in the content object. For example, reference the storage area ID—there is only one storage area in the repository with any given storeage ID. You cannot include the TYPE_TO_QUERY argument if SYSOBJECT_QUERY is set to F.

If SYSOBJECT_QUERY is T and the TYPE_TO_QUERY argument is not included, the predicate executes against instances of the dm_sysobject type. However, if SYSOBJECT_QUERY is T and the TYPE_TO_QUERY argument is included, the predicate executes against instances of the object type specified in the TYPE_TO_QUERY argument. When the query is executed, the method affects the content of only those objects that match the query and for which you have at least Write permission.

When the query is run against instances of an object type, the properties specified in the DQL predicate must be defined for the object type. Consequently, if you want to specify custom properties of a SysObject subtype in the predicate, you must include the TYPE_TO_QUERY argument and identify the subtype in that argument.

For example, suppose you want to migrate all content of a subtype called “loan_documents”, and that subtype has a property named customer_state. The following MIGRATE_CONTENT statement will execute against all loan_document objects and moves the content for those belonging to customers in CA (California) state:

```
EXECUTE migrate_content
WITH target_store='storage_005',
query='customer_state=''CA''',
sysobject_query=T,
type_to_query='loan_documents'
```

Note: Since there are additional arguments for migrating content from an external store, if your query selects content in external stores and other stores, some undefined conditions can occur. Be sure that your query selection is all external store content, or no external store content.

3.38.6.6 Affected content when referencing SysObjects or its subtypes

When you execute the method against SysObjects or subtypes of SysObject, the method acts only on the first primary content page (page 0) of each object, the renditions of the primary pages, or both. If a SysObject has multiple primary content pages, the primary pages other than page 0 and their renditions are not affected by the method. If you want to move all the content, use a method syntax that references the content objects, rather than the SysObject.

You can specify which of these files you want the method to act on by setting the RENDITIONS argument. The default for RENDITIONS is primary, which means that the method will only operate on the first primary content files of the SysObjects. If you set the argument to all, the method moves both the primary content page and its renditions. If you set the argument to secondary, the method moves only the renditions but not the primary content.

3.38.6.7 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 operation. BATCH_SIZE controls how many content files are moved in a single transaction within the migration operation (or how many files per worker session if PARALLEL_DEGREE is greater than 1). 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.

3.38.6.8 Parallel content migration

In previous releases, the MIGRATE_CONTENT method performed content migration sequentially, in a single session. Migrating a large number of objects in a production environment could take a long time. To achieve better performance, an administrator would execute the method in multiple sessions concurrently (in parallel), with each session migrating a unique set of objects. This workaround can be challenging to implement because of the difficulty of identifying individual sets of objects to migrate. Additionally, this approach may lead to database deadlocks because of database update collisions caused by multiple sessions attempting to migrate the same object.

The PARALLEL_DEGREE parameter controls how many content migration sessions are created to do the migration. Using the built-in parallel migration facility avoids the difficulties caused by manually implementing concurrent sessions. Use of the PARALLEL_DEGREE parameter with a value greater than zero requires Content Storage Services.

For any value of the PARALLEL_DEGREE parameter, a master session is created. If the value of the parameter is 0, the master session does all the migration; the migration is not done in parallel. For positive values of the parameter, the master session creates at least that number of worker sessions. The master session validates the method arguments, finds all the qualified content objects, and assigns individual content objects for migration to the individual worker sessions. The master session waits until all the worker sessions have terminated. Then it aggregates the results and statistics, cleans up worker sessions if necessary, destroys the task queue, and returns.

There are two special cases that will cause problems if migrated by different worker sessions:

1. Any parent of a migrating content object has more content objects to migrate.
2. The content object has more than one parent and the qualification predicate is specified against dm_sysobject.

In case 1, the different worker sessions will attempt to migrate the different content objects. One session will succeed, but the others will fail due to an i_vstamp mismatch.

In case 2, a content object shared by two or more sysobjects may get assigned to multiple worker sessions. As a result, it is possible for the sessions to migrate the same content object simultaneously, and thus cause a deadlock or some indefinite behaviors.

To avoid those difficulties, the master session will create a special worker session to migrate any content that is in a special case. The special session will migrate all of the special case content objects. If there is a special worker session, there will be total of PARALLEL_DEGREE + 1 worker sessions. For example, if PARALLEL_DEGREE is 3, there will be total of 4 worker sessions. These worker sessions are numbered as 0, 1, 2, and 3. Worker session 0 always acts as the special worker session.

The special session will always try to acquire tasks from the special task queue. If the queue is empty, it will try to acquire a task from the regular task queue. In this way, the special session will share the workload with the other sessions. However, if most of the migration objects are in one of the special cases, the special session will perform most of the migration sequentially. If the content to be migrated falls mostly into the special cases, the migration will be mostly sequential.

The BATCH_SIZE parameter applies to each worker session individually. For example, if the BATCH_SIZE is 400, each worker session will migrate 400 content objects in a single transaction.

3.38.6.9 Removing the original content

When you move content, you can choose whether to remove the content from the current (source) storage area or not. However, you cannot remove the original content when the source storage area is an external store, so the REMOVE_ORIGINAL argument is set to FALSE when the source is an external store.

If you choose not to remove the content from the current storage area and no other content objects reference that file, Documentum CM Server writes a message to the log file to identify the content file as an orphaned file that must be removed manually. (The file is now an orphan because it is no longer referenced by any content objects.) If the source is an external store, the log file will not identify the orphan content files.

If you choose to remove the content from the current storage area and the storage area is a file store storage area, Documentum CM Server first checks to ensure that no other content objects reference that content. If another content object is found that references the content, the content is not removed. Multiple content objects may reference one content file if content duplication checking and prevention is enabled for the file store storage area. The *OpenText Documentum Content Management - Server Administration and Configuration Guide (EDCCS250400-AGD)* contains more information.

3.38.6.10 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.

Note: If you are migrating a single content file, you can retrieve messages using a IDfSession.getMessage method.

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.
- Performance metrics about the amount of time spent in the RDBMS and in the storage area. The information is logged for each object successfully migrated, and accummulated totals for ever 100 objects, and a summary total for all objects.

Here are samples of the performance messages. The first sample shows the column headers for entries for individual objects and the information about one object:

```
Fri Jan 12 10:03:52 2007 953000
[DM_CONTENT_T_MIGRATE_CONTENT_PERF_REC]info:  "Total Time (secs)
Total Storage Time (secs)    Total Database Time (secs)    Storage Time
for current file (secs)    Database Time for current file (secs)    Total
Objects    Current file size    Xput (bytes/sec) for current file    Max
storage time (secs)    Max database time (secs)    Max file size    Migration
Xput (docs/sec)    Storage Xput (bytes/sec)    Total KBytes"

Fri Jan 12 10:03:52 2007 953000
[DM_CONTENT_T_MIGRATE_CONTENT_PERF_REC]info:  "6.765    6.703    0.062
6.703    0.062    1    8182    1209.46045824095    6.703    0.062
8182    0.147819660014782    1220.64747128152    7"
```

This sample shows the summarizing entry for 100 objects:

```
Fri Jan 12 10:05:19 2007 674000
[DM_CONTENT_T_MIGRATE_CONTENT_PERF_FREC]info:  "100 Objects Migrated.
256.513    1.289    3974"
```

Finally, here is a sample of the final summarizing entry:

```
Fri Jan 12 10:07:02 2007 270000
[DM_CONTENT_T_MIGRATE_CONTENT_PERF_FINAL]info:
"250 Migrated. Storage Time(secs): 547.891, Total Database Time: 3.141,
Total Content Size (KBytes): 9415
```

3.38.7 Related administration methods

None

3.38.8 Examples

This example migrates a single content file, represented by the content object 0600000272ac100d:

```
EXECUTE migrate_content FOR 0600000272ac100d
WITH target_store='filestore_02'
```

The next example migrates 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'
```

The next example uses 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'
```

The final example migrates all the content from the external filestore legacy_filestore to filestore_03. The external content files allow direct access, and the files are moved directly by issuing the underlying OS filesystem commands:

```
EXECUTE migrate_content
WITH source_store='legacy_filestore', target_store='filestore_03',
source_direct_access_ok=TRUE, direct_move=TRUE
```

3.39 MIGRATE_TO_LITE

3.39.1 Purpose

Migrates standard objects to lightweight objects.

3.39.2 Syntax

To generate, and optionally run, a script to split standard objects into a shareable parent object and a lightweight object:

```
EXECUTE migrate_to_lite WITH source_type='source_type'

,shared_parent_type='shared_parent_type'
,execution_mode='execution_mode_split'[,recovery_mode=TRUE|FALSE]
,parent_attributes='parent_properties'
[ft_lite_add='property_list' | ft_base_add='property_list']
```

To generate, and optionally run, a script to migrate standard objects to lightweight objects:

```
EXECUTE migrate_to_lite WITH

shared_parent_type='shared_parent_type'

[,lightweight_type='lightweight_type']

,execution_mode='execution_mode'[,recovery_mode=TRUE|FALSE]
{,parent_sql_predicate='parent_sql_predicate'}

{,parent_id='ID'}[,default_parent_id='ID']
[ft_lite_add='property_list' | ft_base_add='property_list']
```

3.39.3 Arguments

Table 3-37: MIGRATE_TO_LITE arguments

Argument	Datatype	Value	Description
source_type	string	*source_type*	Specifies the name of the source type which needs to be split. This type must be a dm_sysobject subtype, not the dm_sysobject itself. After the conversion, this type will become the lightweight type whose shared parent type is specified in the following parameter, shared_parent_type.
shared_parent_type	string	*shared_parent_type*	Defines the shareable type to use. If the specified type is not a shareable type, then it will be converted into one. The specified type needs to be a subtype of dm_sysobject and cannot be dm_sysobject. If the specified type is already a shareable type, then the lightweight_type parameter must be specified.

Argument	Datatype	Value	Description
execution_mode	string	*execution_mode_split*	Specifies the operation to perform. The allowable values for the first form of the method are: • Split and Finalize • Split without Finalize • Finalize These values are described in detail in "Execution mode split values" on page 291.
		execution_mode	Specifies the operation to perform. The allowable values for the second form of the method are: • Generate Script Only • Run and Finalize • Run without Finalize • Cancel • Finalize These values are described in detail in "Execution mode values" on page 292.

Argument	Datatype	Value	Description
recovery_mode	boolean	TRUE|FALSE	Use this flag when the EXECUTION_MODE is either Run and Finalize, Run without Finalize, Split and Finalize, or Split without Finalize. The default is FALSE. If the flag is set to TRUE, then the internally-created interim types (and tables) will be dropped before the process begins. For Split and Finalize and Split without Finalize the shared parent will be reused but the interim child types (and tables) will be dropped.
parent_attributes	string	*parent_properties*	Specifies properties in the source type which will be split into the parent type specified in shared_parent_type. The parameter contains a list of comma-separated property names from the source type. The remainder of the properties in the source_type will remain there.

Argument	Datatype	Value	Description
lightweight_type	string	*lightweight_type*	If the specified type is a standard type, it will be converted to a lightweight type. The type needs to be a direct subtype of the one specified in shared_parent_type. If the specified type is already a lightweight type, then you must also set recovery_mode to TRUE, and specify which lightweight objects go with which parents. Use the parent_sql_predicate, parent_id, and default_parent_id arguments to specify lightweight objects and parents.
parent_sql_predicate	string	*parent_sql_predicate*	This repeating string parameter specifies an SQL (not DQL), predicate qualifying a set of lightweight SysObjects whose parent will be set to the following parameter, parent_id. This parameter is allowed only when the execution_mode is either Run and Finalize or Finalize.

Argument	Datatype	Value	Description
parent_id	ID	*ID*	This repeating ID parameter corresponds to the above parent_sql_predicate parameter. The lightweight SysObjects that match the SQL predicate will be assigned to this parent. The specified ID needs to be a legal ID. This parameter is allowed only when the execution_mode is either Run and Finalize or Finalize.
default_parent_id	ID	*ID*	This ID parameter is used for the remaining materialized lightweight SysObjects created by this method that are not qualified with any predicate specified in a parent_sql_predicate parameter. If this parameter is not specified, then the i_sharing_parent property of all the remaining lightweight SysObjects will be set to the ID of the lightweight object itself. That is, they will be considered as materialized. This parameter is allowed only when the execution_mode is either Run and Finalize or Finalize.

Argument	Datatype	Value	Description
ft_lite_add	string	*property_list*	A comma-separated list of the properties to add to the fulltext index for the lightweight type, or the string 'ALL' to add all the lightweight properties.
ft_base_add	string	*property_list*	A comma-separated list of the properties to add to the fulltext index for the shareable parent type, or the string 'ALL' to add all the shareable properties.

3.39.4 Return value

MIGRATE_TO_LITE returns an object id for the text file containing the SQL script to do the migration. The script will be generated regardless of the execution_mode specified.

3.39.5 Execution mode split values

For the first form of the method, execution_mode can take on the following values:

- Split and Finalize—Generates the script and immediately runs it.
- Split without Finalize—Generates the script, but the original types are not changed. You can then validate the changes before making the changes permanent.
- Finalize—Makes the changes. For this mode, only the source_type, shared_parent_type, and execution_mode parameters are required.

When you use this command to split a type, the indexes on the type attributes are dropped. You will need to evaluate which attributes to index, and create those indexes, after splitting the type into a lightweight and a shareable type.

3.39.6 Execution mode values

For the second form of the method, execution_mode can take on the following values:

- Generate Script Only—Generates the script, but does not execute it.
- Run and Finalize—Generates the script and immediately runs it.
- Run without Finalize—Generates the script and immediately runs it, but the original types are not changed. All the changes will be applied to corresponding internally-created types (and tables) with the name dm_lw_*id_of_my_parent_type* and dm_lw_*id_of_my_child_type* created with the changes. You can then validate the changes before making the changes permanent.
- Cancel—Allows you to remove the interim types and tables created by the Run without Finalize mode. Typically you would do this if you decided not to finalize the changes. After removing the interim types and tables, the method ends.
- Finalize—Makes the changes permanent by replacing the original types with the internal types and dropped the internal types afterwards. If this option is provided when there is nothing to swap, a warning will be reported back to client and recorded in server log.

3.39.7 Permissions

You must have Superuser privileges, or be the owner of the involved types, to use this method. But also see the note for Oracle installation users following the description section.

3.39.8 Description

Use this method to migrate standard types and objects to shareable and lightweight types and objects. This method can also be used to reparent lightweight objects.

The first form of the command is used to convert a standard type to a lightweight and a shareable type. In this form of the command, the shareable type is formed by splitting off properties from the original standard type. The remaining properties are used to create the lightweight type. Specify the name of the standard type to split in the source_type parameter, the new shareable type in the shared_parent_type parameter, and list the properties to split off of the original type in the parent_attributes parameter. When you use this method to split standard objects, each lightweight object has its own private parent. In other words, each lightweight object is materialized. You can then reparent the lightweight objects by using the second form of the command, specifying parent_sql_predicates and parent_ids to point the lightweight objects to shareable parents.

The second form of the command does not move properties from one type to another. You will use it when all the non-inherited standard type properties will remain with the lightweight type. You can also use this form to reparent lightweight objects.

If you use the second form of the command and do not specify a lightweight type, and you specify a standard type in the shared_parent_type argument, the standard type is changed into a shareable type.

The second form can convert standard objects to lightweight objects and parent them with specified shareable parents. To do this, you specify a shared_parent_type and a lightweight_type. Conceptually, you can imagine that all the standard objects specified by the lightweight_type argument are converted to materialized lightweight objects pointing to private parents of the shared_parent_type. Next, all the objects of the lightweight type that match the parent_sql_predicate argument are made to point to the shareable parent with the associated parent_id. The method continues associating each group of lightweight objects that matches each subsequent parent_sql_predicate and parenting that group with the associated shareable parent. If any unconverted materialized lightweight objects remain after converting each parent_sql_predicate group, the remaining objects are parented with the default_parent_id shareable parent, if specified. Typically, you would create some shareable parents ahead of time, and then use this method to associate that list of objects with the groups defined by the parent_sql_predicate.

To use this method to reparent lightweight objects, specify the shared_parent_type and the lightweight_type, the parent_sql_predicate to select which objects to reparent and the parent_id to reparent to. You must also specify recovery_mode as TRUE, or the method won't work with already converted objects of lightweight_type.

Fulltext support

Use the ft_lite_add and ft_base_add arguments to specify which properties are included in the fulltext index. These arguments add the shareable parent type properties and the lightweight type properties to fulltext indexing. The ft_lite_add variations add the properties specified for the lightweight type, and the ft_base_add variations add the properties specified for the shareable parent type.

Note: In order to use this method with an Oracle installation, the user account that Documentum CM Server uses to access the database must have enhanced privileges. The following example assumes that you have logged into SQLPLUS as the SYS user, in SYSDBA mode to grant these privileges to the repository_1 user:

```
grant DBA to repository_1;
```

These are powerful privileges and should be revoked when the migration is completed.

3.39.9 Examples

This example generates a script to convert usertype_1 to a shareable type. You do not execute this script against your database. It is created so you can examine the commands that will be issued when you later execute this method using another form of the method, either Run and Finalize or Run without Finalize:

```
EXECUTE  migrate_to_lite WITH "shared_parent_type"='usertype_1'
,"execution_mode"='Generate Script Only'
```

To get the script, you can issue the following DQL statement (assume that the result of the earlier method returned the object ID 0855706080007c19):

```
EXECUTE get_file_url FOR '0855706080007c19' WITH "format"='text'
```

This example converts usertype_2 to a shareable type in one operation:

```
EXECUTE  migrate_to_lite WITH "shared_parent_type"='usertype_2'
,"execution_mode"='Run and Finalize'
```

This example converts usertype_3 to a shareable type in two steps. After step one, you can examine the script and the temporary types and tables to verify the repository changes. For step one, use the Run without Finalize execution_mode to run the script and create the temporary tables:

```
EXECUTE  migrate_to_lite WITH "shared_parent_type"='usertype_3'
,"execution_mode"='Run without Finalize'
```

Then, for step two, use the Finalize execution_mode to swap in the temporary tables and commit the change:

```
EXECUTE  migrate_to_lite WITH "shared_parent_type"='usertype_3'
,"execution_mode"='Finalize'
```

This example converts an existing standard type, test_payment_check, and all its objects, into a lightweight type and a shareable type, test_payment_bank. Each lightweight object will have its own private parent object. The parent type will take the attributes bank_code and routing, but the other attributes will be part of the lightweight type.

In this example, there are already objects created of type test_payment_check. You can create the type using the following command:

```
 CREATE TYPE "test_payment_check" (
 account integer,
 check_number integer,
 transaction_date date,
 amount float,
 bank_code integer,
 routing integer
 ) WITH SUPERTYPE "dm_document" PUBLISH
```

The *OpenText Documentum Content Management - High-Volume Server Development Guide (EDCCS250400-DGD)* contains more information about the batch example program.

Use the following command to split the test_payment_check objects into lightweight objects of type test_payment_check, and private parents of type test_payment_bank.

The bank_code and routing attribute values will be associated with the private parent, and the other attributes will be associated with the lightweight objects. The attributes account and check_number of the (now lightweight) type test_payment_check will be fulltext indexed:

```
EXECUTE migrate_to_lite WITH "source_type"='test_payment_check'
,"shared_parent_type"='test_payment_bank'
,"execution_mode"='Split and Finalize'
,"parent_attributes"='bank_code,routing'
,"ft_lite_add"='account,check_number'
```

The following example takes the materialized lightweight objects converted by the last example and reparents them. Objects with check_numbers less than five are reparented to a separately created test_payment_bank shareable parent, and the remaining materialized objects are reparented to another separately created shareable parent:

```
EXECUTE "migrate_to_lite" WITH
"shared_parent_type"='test_payment_bank',"lightweight_type"='test_payment_check'
,"execution_mode"='Run and Finalize',"parent_sql_predicate"='check_number<5'
,"parent_id"='0925392d80001d5d'
,"default_parent_id"='0925392d80001d5e'
```

3.40 MODIFY_TRACE

3.40.1 Purpose

Turns tracing on and off for full-text index query operations.

3.40.2 Syntax

```
EXECUTE modify_trace
WITH subsystem='fulltext',value='tracing_level'
```

3.40.3 Arguments

Table 3-38: 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 querying operations.

Argument	Datatype	Value	Description
VALUE	S	*tracing_level*	The tracing_level must be one of: *none*, to turn off tracing *all*, to log both Documentum CM Server and full-text querying messages

3.40.4 Return value

MODIFY_TRACE returns a collection with one query result object. The object has one Boolean property that indicates the success (TRUE) or failure (FALSE) of the operation.

3.40.5 Permissions

You must have Sysadmin or Superuser privileges to use this method.

3.40.6 Description

MODIFY_TRACE turns tracing on and off for full-text index operations. When tracing is on, tracing information for all full-text queries is logged. The trace information is placed in the server's log file.

The tracing information includes informational, verbose, and debug messages. 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 none, but the method returned FALSE, the trace level remains at the all level.

3.40.7 Related administration methods

"LOG_OFF" on page 258
"LOG_ON" on page 259
"SET_OPTIONS" on page 340

3.40.8 Examples

The following example turns on full tracing:

```
EXECUTE modify_trace
WITH subsystem = 'fulltext',value = 'all'
```

3.41 MOVE_INDEX

3.41.1 Purpose

Moves an existing object type index from one tablespace or segment to another.

3.41.2 Syntax

```
EXECUTE move_index FOR 'dmi_index_obj_id'
WITH name = 'new_home'
[,global_index=true|false]
```

3.41.3 Arguments

Table 3-39: 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.
GLOBAL_INDEX	B	TRUE or FALSE	Indicates whether the index being created is a global index. This parameter can only be used for partitioned types. The default is FALSE.

3.41.4 Return value

MOVE_INDEX returns a collection with one query result object. The object contains one Boolean property that indicates the success (TRUE) or failure (FALSE) of the operation.

3.41.5 Permissions

You must have Sysadmin or Superuser privileges to use this method.

3.41.6 Description

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. The *OpenText Documentum Content Management - Server System Object Reference Guide (EDCCS250400-ORD)* contains information about the list of the properties defined for the index type.

GLOBAL_INDEX is a boolean parameter that indicates whether the index being moved is a global index. The default value is FALSE. This parameter can be used only when the type is a partitioned type, and is only useful if the index is a unique index.

MOVE_INDEX only operates on object type indexes that are represented in the repository by dmi_index objects.

Note: Those indexes that do not have a dmi_index object (DMI_OBJECT_TPE_UNIQUE and DM_FED_LOG_INDEX for example) may be moved manually. However, be sure to shut down Documentum CM Server before moving them.

3.41.7 Related administration methods

“DROP_INDEX” on page 217
“FINISH_INDEX_MOVES” on page 223
“MAKE_INDEX” on page 260

3.41.8 Examples

This example moves the index represented by 1f0000012643124c to the index2 tablespace:

```
EXECUTE move_index FOR '1f0000012643124c'
WITH name = 'index2'
```

3.42 PARTITION_OPERATION

3.42.1 Purpose

Creates partitioning scheme objects and SQL scripts to control repository partitioning.

3.42.2 Syntax

To create a partitioning scheme object:

```
EXECUTE partition_operation WITH
operation='create_scheme',partition_scheme='scheme_name'{,partition_name='partition_name'
,range=right_limit,tablespace='storage_name'}
```

To generate a script to apply a partition scheme to types and/or registered tables:

```
EXECUTE partition_operation WITH
operation='db_partition',partition_scheme='scheme_name'{,type_name='type_name'}
{,table_name=table_name,table_own='table_owner'}
```

To add a partition (range must be greater than the upper bound of the scheme):

```
EXECUTE partition_operation WITH
operation='add_partition',partition_scheme='partition_scheme'{,partition_name='partition_
name',range='right_limit',tablespace='storage_name'}
```

To delete a partition (must be last partition in the scheme):

```
EXECUTE partition_operation WITH
operation='delete_partition',partition_scheme='partition_scheme',partition_name='partitio
n_name'
```

To split a partition (range must be one of the values defined for the partition to be split, the source partition):

```
EXECUTE partition_operation WITH
operation='split_partition',partition_scheme='partition_scheme',source_name='source_name'
,partition_name='partition_name',range='right_limit',tablespace='storage_name'
```

To merge a partition with the one to its left (with a lower range):

```
EXECUTE partition_operation WITH
operation='merge_partition',partition_scheme='partition_scheme',partition_name='partition
_name'
```

To switch out a partition to a temporary table:

```
EXECUTE partition_operation WITH
operation='switch_out',partition_scheme='partition_scheme',temp_table_suffix='temp_table_
suffix,partition_name='partition_name',execute_now=TRUE|FALSE
```

To switch data into a partition from a temporary table:

```
EXECUTE partition_operation WITH
operation='switch_in',partition_scheme='partition_scheme',temp_table_suffix='temp_table_s
uffix,partition_name='partition_name',execute_now=TRUE|FALSE
```

3.42.3 Arguments

Table 3-40: PARTITION_OPERATION arguments

Argument	Datatype	Value	Description
operation	string	listed in description column	Defines the operation that you want to execute. The only values allowed are: • create_scheme • db_partition • add_partition • delete_partition • split_partition • merge_partition • switch_out • switch_in
partition_scheme	string	*scheme_name*	
partition_name	string	*partition_name*	The name of the partition. Some databases may restrict the choices for a partition name. For example, choosing default as a partition name causes an error when you execute the partitioning script.
range	integer	*right_limit*	Uppermost i_partition value of an object placed in this partition is equal to (*right_limit* - 1).
tablespace	string	*storage_name*	The tablespace for the partition. You can find the tablespace name for your repository by accessing the underlying database. By default, the name is dm_<repository_name>_docbase.

Argument	Datatype	Value	Description
type_name	string	*type_name*	Specifies a type to partition. Must be a partitionable supertype. Create the type with CREATE PARTITIONABLE TYPE, or modify a type to be partitionable with ALTER TYPE ... ENABLE PARTITION.
table_name	string	*table_name*	Specifies a registered table. The table must already have an i_partition column. Do not use this for repository-created tables; use the type_name argument for those tables.
owner_name	string	*owner_name*	The table owner name.
source_name	string	*source_name*	The name of the partition to split.
temp_table_suffix	string	*temp_table_suffix*	The suffix for the temporary table. When tables are switched in or out, the table name must be the name of the type or registered table with the suffix applied. Use a short, one or two character suffix.

3.42.4 Return value

For create_scheme, PARTITION_OPERATION returns a Boolean value and an object id. The Boolean value indicates if the operation succeeded. The object ID is either the ID of the partition scheme object, or the object ID of the SQL script that is run against the database to perform the partitioning operation.

Note: One way to retrieve the script is to issue the following DQL statement (assume that the result of PARTITION_OPERATION returned the object ID 0855706080007c19):

```
EXECUTE get_file_url FOR '0855706080007c19' WITH "format"='text'
```

That statement will return the location of the file.

3.42.5 Permissions

You must have Sysadmin or Superuser privileges to use all the operations of this method. If you have type owner privileges, you can use the db_partition operation for that type.

In Oracle installations, you must be logged in as sysdba to run the script.

3.42.6 Description

Use this method to create a partitioning scheme object (dm_partition_scheme). This object will record the types and tables that use the scheme, the name and range of each partition, and the tablespace (or filegroup) where each partition is stored. When a partitioning scheme object is first created, there are no types or registered tables associated with it, just the name, range and tablespace for the partitions in the scheme. When you associate types or registered tables with a partitioning scheme, a script is created that must be run against the underlying database to partition the database tables to match the partitioning scheme. Documentum CM Server does not run this script since it may take a long time to perform the partitioning on a large database, and you may want to examine the script before running it.

You also use this method to modify a partition scheme. Modifying the scheme also creates a database partitioning script that Documentum CM Server runs against the database for you.

When you run the script-creating variations of this method, the method returns the r_object_id of the partitioning script to run in your database. The partition scheme object information and the database are not consistent until the script is run, so the is_validated property of the partitioning scheme object is set to FALSE. After the script is created, you must run it against the underlying database, using the database's SQL facility. The scripts make the changes to the database that reflect the information in the partition scheme object. As the script is successfully run against the database, the is_validated property of the partitioning scheme object is set to TRUE.

If you did not enable partitioning when you created your repository, create a scheme, and then use the db_partition operation without specifying any tables or types to partition the repository intrinsic types.

The following are the intrinsic type hierarchies that are partitioned when you create a repository with partitioning enabled, or you later enable partitioning:

- dm_acl
- dm_sysobject
- dmr_containment
- dm_assembly
- dm_relation
- dm_relation_type
- dmi_otherfile
- dmi_replica_record
- dmr_content
- dmi_subcontent
- dmi_queue_item

3.42.6.1 Originally partition-enabled repository compared to originally non-partitioned repository

You can use partitioning even if you do not create a partition-enabled repository. You can partition the repository after it has been created. In either case, you will have a partitioned repository.

If you create a partition-enabled repository, the eleven intrinsic type hierarchies listed above will be partitioned, each in its own partition scheme. If you want more than one of those hierarchies to be partitioned identically, you will need to run PARTITION_OPERATION on each hierarchy you want to modify separately. However, you can use separate partitioning schemes on those hierarchies if you need to for your application. There is no way to merge the partition schemes so that multiple hierarchies are configured with a single scheme. You can add custom type hierarchies (where the topmost custom type specifies supertype NULL), to any of the intrinsic type schemes and manage them together, if needed.

In contrast, if you partition-enable your repository after creation, all the intrinsic types are configured with a single partitioning scheme. In this case, all the type hierarchies are partitioned identically. There is no way to separate them into different partition schemes if you later decide you need to do this.

You should plan your repository partitioning carefully before deciding which alternative meets your needs.

3.42.7 Create_scheme

To create a partition scheme, use the create_scheme operation. You will specify a name for the scheme. Each scheme has a unique name. In this guide we use the convention that the name of a partition scheme ends with the characters _sch. In addition to the scheme, you will specify one or more partition name, range, and tablespace. The name is used to refer to the partition. The range is a single integer that is one larger than the largest value in the i_partition attribute for any objects in that partition. The tablespace specifies where the partition is stored. The r_object_id of the partition scheme object is returned.

The following command creates a partition scheme named my_partition_sch, with four partitions named zero, first, second, and third. Partition zero contains objects with i_partition values from 0 to 9, partition first contains objects with i_partition values from 10 to 19, partition second has 20 to 29, and partition third has 30 to 39. Notice that the range for partition zero is set to 10, but partition zero contains objects with i_partition values from 0 to 9:

```
EXECUTE partition_operation with "operation"='create_scheme',
"partition_scheme"='my_partition_sch',
"partition_name"='zero',range=10,"tablespace"='dm_techpubsglobal_docbase',
"partition_name"='first',range=20,"tablespace"='dm_techpubsglobal_docbase',
"partition_name"='second',range=30,"tablespace"='dm_techpubsglobal_docbase',
"partition_name"='third',range=40,"tablespace"='dm_techpubsglobal_docbase'
```

3.42.8 Db_partition

To apply a partition scheme, use the db_partition operation. You specify the partition scheme to apply, and the types and tables to partition. This operation adds the specified types and registered tables to the partition scheme object, and creates a script that partitions the database tables. A type or registered table can be associated with only one partition scheme, so the command will fail if a type or registered table is listed in any other partition scheme.

Use this operation to partition-enable a repository that was not originally created as partition-enabled. If you run this operation and do not specify any types or registered tables, the base types will be partitioned. Also, see the note below about increasing the number of cursors for an Oracle installation.

After you have created a partition scheme, perform these steps to partition the repository:

1. Use the db_partition operation of the PARTITION_OPERATION administrative method to create the partitioning script.
2. Stop Documentum CM Server.
3. Execute the partitioning script against the database.
4. Restart Documentum CM Server.

Note: Do not execute the script if there have been changes to the repository schema after it was generated. Create a new script if the schema changes.

The following command creates a script to apply my_partition_sch to the type, my_type. The command returns the r_object_id of the script, and sets the is_validated property of my_partition_sch to FALSE:

```
EXECUTE partition_operation with "operation"='db_partition',
"partition_scheme"='my_partition_sch',"type_name"='my_type'
```

After running the script, the is_validated property is TRUE, and the tables in the underlying database are partitioned as set out in the my_partition_sch object.

3.42.9 Add_partition

To add a partition, use the add_partition operation. You specify the partition name, range, and tablespace for each partition to add. The range of the new partition must be greater than the upper bound of the current partition scheme. Any types or registered tables already attached to this partition scheme will be modified. This operation modifies the partition information in the partition scheme object, and creates and runs a script to modify the partitioning in the database.

The following command adds a partition to the partition scheme my_partition_sch, and runs a script to add the additional partition to the database tables. In this example assume that my_type was partitioned in the earlier db_partition example:

```
EXECUTE partition_operation WITH "operation"='add_partition',
"partition_scheme"='my_partition_sch'
,"partition_name"='fourth',range=50,"tablespace"='dm_techpubsglobal_docbase'
```

3.42.10 Delete_partition

To delete a partition, use the delete_partition operation. The partition must be the last partition in the partition scheme. Like add_partition, this operation modifies the partition information in the partition scheme object, and runs a script to modify the partitioning in the database.

The following command removes the last partition (created in the add_partition example):

```
EXECUTE partition_operation WITH "operation"='delete_partition',
"partition_scheme"='my_partition_sch',"partition_name"='fourth'
```

3.42.11 Split_partition

To split a partition, use the split_partition operation. You specify a source partition to split, and the name, range, and tablespace of the new partition. The range value must fall into the defined range of the source partition. The new partition is to the left of the source partition (the new partition range is lower than the modified source partition).

The following command splits the zero partition into partitions five and zero:

```
EXECUTE partition_operation WITH "operation"='split_partition',
"partition_scheme"='my_partition_sch',
"source_name"='zero',
"partition_name"='five',range=5,"tablespace"='dm_techpubsglobal_docbase'
```

If you used the previous examples, before executing this command, the partition names and ranges are: zero (10), first (20), second (30), third (40). After executing this command, they are: five (5), zero (10), first (20), second (30), third (40).

3.42.12 Merge_partition

To merge two partitions, use the merge_partition operation. This operation merges the partition specified into the partition on its left (into the lower range numbered partition).

The following command merges the partition named five into the next lower range partition (partition zero, if you've used the previous examples):

```
EXECUTE partition_operation WITH "operation"='merge_partition',
"partition_scheme"='my_partition_sch',"partition_name"='five'
```

3.42.13 Switch_out

To move data out of a partition into temporary tables, use the switch_out operation. For some databases (SQL Server), the temporary tables (if they've already been created), must be empty. After the switch_out operation, the temporary table contents will be in the partition, and the data previously in the partition will reside in the temporary tables for the types and registered tables in that partition scheme. If you specify execute_now as TRUE, Documentum CM Server will execute the SQL script and switch out the data, otherwise you will need to run the script yourself. The temporary table names will match the type or registered table name that you switch out with the temp_table_suffix added.

While switch_out is executing, the data in the partition is not consistent, so you must insure that there is no repository access to that partition during the switch.

The following example switches out the data in the second partition into a temporary table named my_type_x (my_partition_sch only has one type, my_type, in it). Since execute_now is specified, you do not need to run the SQL script; it is done for you:

```
EXECUTE partition_operation WITH "operation"='switch_out',
"partition_scheme"='my_partition_sch',
"temp_table_suffix"='_x',"partition_name"='second',"execute_now"=TRUE
```

3.42.14 Switch_in

To move data into a partition from temporary tables, use the switch_in operation. The partition must be empty for SQL Server. Create the temporary tables and fill them with the data to switch into the partition. After the switch_in operation, the temporary table contents will be in the partition, and the data previously in the partition will reside in the temporary tables. If you specify execute_now as TRUE, Documentum CM Server will execute the SQL script and switch out the data, otherwise you will need to run the script yourself. The temporary table names will match the type or registered table name that you switch out with the temp_table_suffix added.

You must have created all the temporary tables to switch into the empty partition. One quick way to create those tables, is to use the switch_out operation on an empty partition. When the operation executes, it will create all the required tables. Since the partition was empty, all the tables will be, too. You can then fill the tables with data.

While switch_in is executing, the data in the partition is not consistent, so you must insure that there is no repository access to that partition during the switch.

The following example switches in the data in the from the temporary table named my_type_x (my_partition_sch only has one type, my_type, in it) into the partition named second. Since execute_now is specified, you do not need to run the SQL script; it is done for you:

```
EXECUTE partition_operation WITH "operation"='switch_in',
"partition_scheme"='my_partition_sch',
"temp_table_suffix"='_x',"partition_name"='second',"execute_now"=TRUE
```

3.42.15 Oracle installations

Note: If you experience difficulties with the following information, please consult the Oracle documentation or Oracle support for assistance.

In order to use this method with an Oracle installation, you must run the script as SYSDBA. Additionally, if you are partitioning a non-partitioned database, you may want to increase the number of open database cursors, or the script may exit with the error:

```
ORA-0100 Max Opened Cursors Exceeded
```

To increase the number of cursors, consult your Oracle documentation. It may tell you to use a command similar to:

```
ALTER SYSTEM SET OPEN_CURSORS=2000 SCOPE=MEMORY SID='*';
```

For example, if dmadmin is the installation owner:

```
C:\Documents and Settings\dmadmin>sqlplus "/as sysdba"
@ C:\Documentum\data\testenv\content_storage_01\00000057\80\00\01\19.txt
```

Additionally, if you are partitioning a non-partitioned database, you may want to increase the number of open database cursors, or the script may exit with the error:

```
ORA-0100 Max Opened Cursors Exceeded
```

to alter the number of open cursors.

If you exit the script with an error (from inadvertently exceeding the number of open cursors, for example), correct the error, and rerun the script, you may see an error message like:

```
ORA-12091: cannot online redefine table "TECHPUBS"."DMC_WFSD_ELEMENT_S" with
materialized views
```

or like:

```
ORA-23539: table "TECHPUBS"."DM_PLUGIN_R" currently being redefined
```

caused by leftover temporary items from the previous script failure. One way to correct this error is to run a command similar to:

```
execute DBMS_REDEFINITION.ABORT_REDEF_TABLE('<Schema Name>','<Table Name>'
,'<Table Name>I');
```

Where <Table Name>I is the intermediate table name used for the redefinition. From the first error message, we would use this command:

```
execute DBMS_REDEFINITION.ABORT_REDEF_TABLE('TECHPUBS','DMC_WFSD_ELEMENT_S'
,'DMC_WFSD_ELEMENT_SI');
```

or from the second:

```
execute DBMS_REDEFINITION.ABORT_REDEF_TABLE('TECHPUBS','DM_PLUGIN_R'
,'DM_PLUGIN_RI');
```

3.42.16 Switch in and switch out partitions

Switching data into a partition must be carefully planned. Typically, you will create a number of offline tables to load with data, use whatever native database method is available to load the tables into temporary tables in the database, create the table indexes, and then swap the tables into the prepared repository partition. This technique can load large amounts of data into a repository while causing a minimum of disruption to normal use of the repository. This technique can also be used to remove large amounts of data from a repository by switching out a partition for empty offline tables.

The typical steps you would take to do a partition exchange involve the following:

1. Create the offline tables.

   The *OpenText Documentum Content Management - High-Volume Server Development Guide (EDCCS250400-DGD)* contains more information.

2. Load the offline tables.

   Load the tables with data using whatever methods are available to you with your database.

3. Create the offline index tables.

   The offline tables must index the same properties as the online objects do. The schema must be identical. Create the offline index tables in the same tablespace as the current online indexes.

4. Exchange the partition for the offline tables.

   Run the PARTITION_OPERATION administration method to switch in or switch out the data. After following these steps, the data that was previously in the offline tables is in the online partition, and the previously online data is now in the offline table.

5. Validate the exchange.

   Check the online data to be sure that the exchange was successful.

The *OpenText Documentum Content Management - High-Volume Server Development Guide (EDCCS250400-DGD)* contains more details about switching data into a partition.

3.43 PING

3.43.1 Purpose

Determines if the client still has an active server connection.

3.43.2 Syntax

```
dmAPIGet("apply,c,NULL,PING[,RETRY_ON_TIMEOUT,B,T|F]")
```

You cannot use the EXECUTE statement to invoke PING.

3.43.3 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.

3.43.4 Return value

PING returns a collection with one query result object. The object has one property, called result, that is TRUE if the connection is alive. If the connection has timed out, a null collection is returned.

3.43.5 Permissions

Anyone can use this method.

3.43.6 Description

None

3.43.7 Related administration methods

None

3.43.8 Examples

```
dmAPIGet("apply,c,NULL,PING,RETRY_ON_TIMEOUT,B,T")
```

3.44 PURGE_AUDIT

3.44.1 Purpose

Removes an audit trail entry from the repository.

3.44.2 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=TRUE|FALSE][,purge_signed=TRUE|FALSE][,commit_size=value]
```

3.44.3 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.
			• PREDICATE Deletes all audit trail entries that satisfy a DQL predicate. If you include PREDICATE, include the DQL_PREDICATE argument.

Argument	Datatype	Value	Description
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 property 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 a pattern specification. Refer to "Date literals" on page 10, 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.
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 property 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 a pattern specification. Refer to "Date literals" on page 10, 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, or AUDIT_RECORD. For all modes except AUDIT_RECORD, *object_id* is the object ID recorded in the audited_obj_id property 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 Documentum CM Server to delete audit trail entries whose i_is_archived property is set to F (FALSE). The default for this argument is F (FALSE), meaning that only entries whose i_is_archived property is set to T (TRUE) are removed. You cannot set this to T if the DQL_PREDICATE argument is included.
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.

Argument	Datatype	Value	Description
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.

3.44.4 Return value

The PURGE_AUDIT method returns a collection with one query result object. The query result object has two properties, result and deleted_objects. The result property 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.

3.44.5 Permissions

You must have Purge Audit privileges to execute this method.

3.44.6 Description

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 properties 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

3.44.7 Related administration methods

None

3.44.8 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'
```

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=TRUE,commit_size=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'
```

This example deletes the single audit trail entry whose object ID is 5f0000021372ac6f:

```
EXECUTE purge_audit WITH delete_mode='AUDIT_RECORD',
object_id='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=TRUE
```

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-quotes in the predicate string, (for example, 'dcm%'), escape the single-quotes with single-quotes. This is illustrated in the preceding example.

3.45 PURGE_CONTENT

3.45.1 Purpose

Sets a content file off line and deletes the file from its storage area.

3.45.2 Syntax

```
EXECUTE purge_content FOR 'content_object_id'
```

3.45.3 Arguments

PURGE_CONTENT has no arguments.

3.45.4 Return value

PURGE_CONTENT returns a collection with one query result object. The object has one Boolean property whose value indicates the success (TRUE) or failure (FALSE) of the operation.

3.45.5 Permissions

You must have Sysadmin or Superuser user privileges to use this method.

3.45.6 Description

PURGE_CONTENT marks a content file as off-line. Additionally, if the file is referenced by only one content object, it deletes the file from the storage area. If the file is referenced by other content objects, the file is not deleted. A file may be referenced by multiple content objects if the storage area is a file store storage area configured to use content duplication checking and prevention. The *OpenText Documentum Content Management - Server Administration and Configuration Guide (EDCCS250400-AGD)* contains more information.

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 property 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 Documentum Servers, you must issue a connection request that bypasses the Documentum CM Server to use PURGE_CONTENT in the session.

3.45.7 Examples

This example deletes the content file associated with the content object represented by 060000018745231c and set the is_offline property in the associated content object to TRUE:

```
EXECUTE purge_content FOR '060000018745231c'
```

3.46 PUSH_CONTENT_ATTRS

3.46.1 Purpose

Updates content metatdata in a content addressable and S3-compatible storage systems.

3.46.2 Syntax

```
EXECUTE push_content_attrs FOR object_id
WITH format=format_name [,page=number]
[,page_modifier=value]
```

object_id is the ID of a document or other SysObject.

3.46.3 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.

Argument	Datatype	Value	Description
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 property in the content object.

3.46.4 Return value

PUSH_CONTENT_ATTRS returns a collection with one query result object. The object has one Boolean property that contains TRUE if the method was successful and FALSE if unsuccessful.

3.46.5 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*.

3.46.6 Description

PUSH_CONTENT_ATTRS updates the metadata fields in a content-addressed and S3-compatible storage systems for a particular content file. First, the method reads the values in the following content-related properties 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 property 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 properties, depending on the field's datatype. For example, if the field name is "title" and its datatype is character string, the content_attr_value property contains the actual title value. The remaining properties, content_attr_num_value and content_attr_date_value are set to the default NULLINT and NULLDATE values.

The *OpenText Documentum Content Management - Server Fundamentals Guide (EDCCS250400-GGD)* contains complete information about how to save a document to content-addressed and S3-compatible storage systems.

3.46.6.1 For Centera store

After the method reads the content object properties, it invokes the content-addressed plug-in library to update the storage system metadata fields.

Note: Before updating a field, ensure it is defined in both the content object's content_attr_name property and in the storage object's a_content_attr_name property. If a field is named in the content object's content_attr_name property 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 property but not in the content object's content_attr_name property, 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 property of the subcontent object that records content addresses for the content file.

3.46.6.2 For S3-compatible stores

After the method reads the content object attributes, it collects all user metadata available in the corresponding content object, and prepends "X-AMZ-metadata-" with the collected metadata. Documentum CM Server generates the PUT request containing headers with "X-AMZ-metadata-*" and sends the request to S3-compatible store.

If PUSH_CONTENT_ATTRS completes successfully, in case of versioning-enabled S3-compatible store, the S3-compatible store generates new version of object. Documentum CM Server then updates the i_contents atrribute with the new version of object.

3.46.7 Related administration methods

"SET_CONTENT_ATTRS" on page 336

3.46.8 Examples

```
EXECUTE push_content_attrs FOR 090000026158a4fc
WITH format=txt,page=1
```

3.47 RECOVER_AUTO_TASKS

3.47.1 Purpose

Recovers work items that have been claimed, but not yet processed, by a workflow agent associated with a failed Documentum CM Server.

3.47.2 Syntax

```
EXECUTE recover_auto_tasks
WITH server_config_name=name
```

3.47.3 Arguments

Table 3-44: RECOVER_AUTO_TASKS arguments

Argument	Datatype	Value	Description
SERVER_CONFIG_NAME	I	*name of the server config object*	Use the name of the server config object representing the Documentum CM Server that crashed or failed.

3.47.4 Return value

The method returns T (TRUE) if it completes successfully and F (FALSE) if not.

3.47.5 Permissions

This method must be executed by a user with Sysadmin or Superuser privileges.

3.47.6 Description

If a Documentum CM Server fails, its workflow agent is stopped also. When the server is restarted, the workflow agent will recognize and process any work items it had claimed but not processed before the failure. However, if you cannot restart the Documentum CM Server, you must recover those work items already claimed by its associated workflow agent so that another workflow agent can process them. RECOVER_AUTO_TASKS performs that recovery.

Running RECOVER_AUTO_TASKS executes a query to update all work items claimed but unprocessed by the failed server's workflow agent. The query resets the a_wq_name property in the work items to empty. This allows other workflow agents running against the repository to claim those work items for processing.

Before executing this method, make sure that the specified Documentum CM Server is not running.

3.47.7 Related administration methods

None

3.47.8 Examples

```
EXECUTE recover_auto_tasks
WITH server_config_name='DevRepository_1'
```

3.48 REGISTER_ASSET

3.48.1 Purpose

Queues a request to the Media Server to generate a thumbnail, proxies, and metadata for a rich media content file.

3.48.2 Syntax

```
EXECUTE register_asset FOR object_idWITH [page=page_number][,priority=priority_level]
```

object_id is the object ID of the document that contains the content file.

3.48.3 Arguments

Table 3-45: 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.

3.48.4 Return value

REGISTER_ASSET returns a collection with one query result object. The object has properties, result and result_id. The result property is a Boolean property whose value indicates success (TRUE) or failure (FALSE) of the operation. The result_id property records the object ID of the newly created queue item object.

3.48.5 Permissions

You must have at least Version permission on the object or Sysadmin privileges to use this method.

3.48.6 Description

REGISTER_ASSET is called by Documentum CM Server whenever an object with rich media content is saved or checked in to the repository. 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 Transformation Services - Media is installed. REGISTER_ASSET sets the following properties in the queue item:

Table 3-46: Queue item property values set by REGISTER_ASSET

Property	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

3.48.7 Related administration methods

"SET_CONTENT_ATTRS" on page 336
"TRANSCODE_CONTENT" on page 347

3.48.8 Example

```
EXECUTE register_asset FOR 090002410063ec21
WITH page=0,priority=8
```

3.49 REINDEX_PARTITIONABLE_TYPE

3.49.1 Purpose

Reindex a partitionable type to increase performance.

3.49.2 Syntax

```
EXECUTE reindex_partitionable_type [WITH type_name='type_name']
```

3.49.3 Arguments

Table 3-47: REGISTER_ASSET arguments

Argument	Datatype	Value	Description
type_name	S	*type_name*	Type to reindex.

3.49.4 Return value

The method returns T (TRUE) if it completes successfully and F (FALSE) if not.

3.49.5 Permissions

This method must be executed by a user with Sysadmin or Superuser privileges.

3.49.6 Description

REINDEX_PARTITIONABLE_TYPE is used for databases upgraded from release 6.0 or 6.0 SP1 to 6.5. In 6.0, indexes for types that have the i_partition attribute included a column for that property in indexes that included r_object_id. This can cause a performance problem for non-partitioned repositories. In 6.5 databases, these indexes do not include the i_partition value unless the repository is partitioned. Only use this method for repositories upgraded to 6.5 that are showing performance issues.

You can reindex all the partitionable types, or any specific partitionable type.

3.49.7 Related administration methods

None.

3.49.8 Example

```
EXECUTE reindex_partitionable_type WITH "type_name"='my_type'
```

3.50 REORGANIZE_TABLE

3.50.1 Purpose

Reorganizes a database table for query performance.

3.50.2 Syntax

```
EXECUTE reorganize_table WITH table_name='name',
[,index='index_name']
[,PARALLEL_DEGREE=<integer value>][,NOLOGGING=[true|false][,ONLINE =true|false]
```

3.50.3 Arguments

Table 3-48: 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 SQL Server and PostgreSQL databases table. The argument is used differently on each database. Refer to the General Notes for details.

Argument	Datatype	Value	Description
PARALLEL_DEGREE	integer	*integer value*	Indicates the degree of parallelism to be used by the underlying Oracle database. The index creation runs faster depending on the number of CPUs, table partitioning and disk fragmentation of the database.
NOLOGGING	B	TRUE or FALSE	This argument helps in faster insertion and index creation. This bypasses the writing of the redo log helping in improving performance. The default is TRUE.
ONLINE	B	TRUE or FALSE	Indicates whether the index being created online. During an online index rebuild, Oracle takes a snapshot log on the target table to hold DML activity, read the table in a full-table scan (read consistent), build the new index and then apply the changes from the snapshot log after the index has been rebuilt. During a regular index rebuild, an exclusive lock occurs as the existing index is read. This command is designed for scheduled downtime periods where there is no DML activity. The default is FALSE.

3.50.4 Return value

REORGANIZE_TABLE returns a collection with one query result object. The object has one property, named result, that contains T if the method was successful or F if the method was unsuccessful.

3.50.5 Permissions

You must be a superuser to execute this method.

Caution
Do not use this method unless directed to do so by OpenText Global Technical Services.

3.50.6 Description

This method is used by the UpdateStatistics administration tool, in conjunction with UPDATE_STATISTICS, when the tool is run on any supported database.

3.50.6.1 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.

Note: Indexes created by Documentum CM 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.

3.50.7 Related administration methods

"UPDATE_STATISTICS" on page 350

3.50.8 Examples

This example reorganizes 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'
```

This example rebuilds an index associated with the dm_sysobject_s table.

```
EXECUTE reorganize_table
WITH table_name='dm_sysobject_s,
index='D_1f0C9fda80000108'
```

3.51 REPLICATE

3.51.1 Purpose

Copies content files in distributed storage areas.

3.51.2 Syntax

```
EXECUTE replicate
WITH query='value',store='value'
[,type='value']
```

3.51.3 Arguments

Table 3-49: 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 "WHERE clause" on page 155).

Argument	Datatype	Value	Description
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.

3.51.4 Return value

REPLICATE returns a collection with one query result object. The object has one Boolean property whose value indicates the success (TRUE) or failure (FALSE) of the operation.

3.51.5 Permissions

You must have Sysadmin or Superuser privileges to use REPLICATE.

3.51.6 Description

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. The *OpenText Documentum Content Management - Server Administration and Configuration Guide (EDCCS250400-AGD)* contains more information about the ContentReplication. The*OpenText Documentum Content Management - Server and Server Extensions Installation Guide (EDCSY250400-IGD)* contains more 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. The*OpenText Documentum Content Management - Server and Server Extensions Installation Guide (EDCSY250400-IGD)* contains more information.

To use REPLICATE, you must be using one server for both data and content requests. If the configuration is set up for Documentum Servers, you must issue a connection request that bypasses the Documentum CM Server to use REPLICATE in the session.

3.51.7 Related administration methods

"DELETE_REPLICA" on page 204
"IMPORT_REPLICA" on page 245

3.51.8 Examples

This example replicates all content for documents of the subtype proposals owned by jennyk:

```
EXECUTE replicate WITH query='owner_name=jennyk',
store='distcomp_1',type='proposals'
```

The next example replicates the content of all objects whose content is authored by Jane:

```
EXECUTE replicate WITH query='any authors in (''Jane'')',
store='diststorage3'
```

3.52 RESET_TICKET_KEY

3.52.1 Purpose

Generates a login ticket key and stores it in the repository.

3.52.2 Syntax

```
EXECUTE reset_ticket_key
```

3.52.3 Arguments

None

3.52.4 Return value

The method returns a collection with one query result object. The object has one property, result, that contains T (TRUE) if the method was successful or F (FALSE) if the method was unsuccessful.

3.52.5 Permissions

You must have Superuser privileges to execute this method.

3.52.6 Description

Use this method when you need to replace a login ticket key in a repository with a new key. Any login tickets generated by the repository before the LTK was reset cannot be used within the repository.

You must restart Documentum CM Server after resetting the login ticket key.

3.52.7 Related administration methods

"EXPORT_TICKET_KEY" on page 222
"IMPORT_TICKET_KEY" on page 246

IDfSession.resetTicketKey()

3.52.8 Examples

```
EXECUTE reset_ticket_key
```

3.53 RESTORE_CONTENT

3.53.1 Purpose

Restores an off-line content file to its original storage area.

3.53.2 Syntax

```
EXECUTE restore_content FOR 'content_object_id'
WITH file = 'path_name' [,other file = 'other_path_name']
```

content_obj_id is the object ID of the content object associated with the specified file.

3.53.3 Arguments

Table 3-50: RESTORE_CONTENT arguments

Argument	Datatype	Value	Description
FILE	S	*file_path*	Specifies the current location of the content file. Use a full path specification.

Argument	Datatype	Value	Description
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.

3.53.4 Return value

RESTORE_CONTENT returns a collection with one query result object. The object has one Boolean property whose value indicates the success (TRUE) or failure (FALSE) of the operation.

3.53.5 Permissions

You must have Sysadmin or Superuser user privileges to use this method.

3.53.6 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 an IDfSession.restore method.

The method places the file in the storage area indicated by the storage_id property of the file's content object and sets the content object's is_offline property 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 Documentum Servers, you must issue a connection request that bypasses the Documentum CM Server to use RESTORE_CONTENT in the session.

3.53.7 Examples

The following examples restore the file named Myproposal using an EXECUTE statement:

```
EXECUTE restore_content
WITH file='c:\archive1\Myproposal'
```

or, on Linux:

```
EXECUTE restore_content
WITH file='u02/archive1/Myproposal'
```

3.54 ROLES_FOR_USER

3.54.1 Purpose

Retrieves the roles assigned to the user in a particular client domain.

3.54.2 Syntax

```
EXECUTE roles_for_user [[FOR] 'dm_user_id']
WITH [USER_NAME=value][,DOMAIN=domain_name]
```

Do not include the FOR clause if you include the NAME argument.

3.54.3 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.

3.54.4 Return value

This returns a collection of one query result object that has three properties: 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 property that contains the roles for the user.

3.54.5 Permissions

Any one can use this method.

3.54.6 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.

The *OpenText Documentum Content Management - Server Fundamentals Guide (EDCCS250400-GGD)* contains more information about groups, roles, and domains.

3.54.7 Related administration methods

None

3.54.8 Examples

```
EXECUTE roles_for_user
WITH user_name=MaryJean,domain="HR_app"
```

3.55 SET_APIDEADLOCK

3.55.1 Purpose

Sets a deadlock trigger on a particular operation.

3.55.2 Syntax

```
EXECUTE set_apideadlockWITH API=api_name,VALUE=TRUE|FALSE{,API=operation_name,VALUE=TRUE|
FALSE}
```

3.55.3 Arguments

Table 3-52: SET_APIDEADLOCK arguments

Argument	Datatype	Value	Description
API	S	*api_name*	Name of the operation on which to set the deadlock trigger.“Valid operation names for SET_APIDEADLOCK” on page 335 lists the operations on which you can set a trigger.
VALUE	BOOLEAN	T (TRUE) or F (FALSE)	TRUE sets the trigger. FALSE removes the trigger.

3.55.4 Return value

SET_APIDEADLOCK returns a collection with one query result object. The object has one Boolean property whose value indicates the success (TRUE) or failure (FALSE) of the operation.

3.55.5 Permissions

Anyone can use this administration method.

3.55.6 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 Documentum CM 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, Documentum CM 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 property and issues a rollback to the database.

The deadlock trigger is removed automatically from the method or operation which triggered the simulated deadlock.

“Valid operation names for SET_APIDEADLOCK” on page 335, lists the operations and methods on which you can place a deadlock trigger.

Table 3-53: Valid operation names for SET_APIDEADLOCK

Operation	Representing
acquire	Acquisition of a work item
bp_transition	Completion of the bp_transition method
branch	Branching of a version tree
checkin	A checkin operation
complete	Completion of a work item
demote	Demotion of an object
dequeue	Dequeuing an object
destroy	Destruction of an object
exec	Any RPC EXEC call (on behalf of a Query, Readquery, Execquery, or Cachequery operation)
next	A next method
promote	Promotion of an object
queue	A queue method
resume	Resumption of an object
revert	A 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	Suspension of an object

3.55.7 Related administration methods

None

3.55.8 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:

```
EXECUTE set_apideadlock
WITH api='promote',value=T,
api='revert',value=T
```

This example removes the deadlock trigger from the Checkin method:

```
EXECUTE set_apideadlock
WITH api='checkin',value=F
```

3.56 SET_CONTENT_ATTRS

3.56.1 Purpose

Sets the content-related properties of a content object.

3.56.2 Syntax

```
EXECUTE set_content_attrs FOR object_idWITH format='format_name',[page=page_number,]
[page_modifier='value',]
parameters='name="value"{,name="value"}'[,truncate=TRUE|FALSE]
```

object_id is the object ID of the document that contains the content file.

3.56.3 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.

Argument	Datatype	Value	Description
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 property 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 properties before setting the new values. The default value is F.

3.56.4 Return value

SET_CONTENT_ATTRS returns a collection with one query result object. The object has one Boolean property whose value indicates the success (TRUE) or failure (FALSE) of the operation.

3.56.5 Permissions

You must have at least Write permission on the object or Sysadmin privileges to use this method.

3.56.6 General notes

SET_CONTENT_ATTRS sets five properties 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.

3.56.6.1 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 properties 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 properties (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 properties (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 properties (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)'
```

"Example settings for content metadata properties in content objects" on page 339, shows the resulting values in the properties in the content object.

Table 3-55: Example settings for content metadata properties in content objects

Property	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 properties are handled. If TRUNCATE is set to T (TRUE), existing values in the properties are removed before the properties 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 properties, the name's value is overwritten. Names specified in the PARAMETERS argument that are not currently present in the properties are appended to the properties. The default for TRUNCATE is F.

3.56.6.2 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 property of the content object. This property, 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.

3.56.7 Related administration methods

"PUSH_CONTENT_ATTRS" on page 317

3.56.8 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)'
```

3.57 SET_OPTIONS

3.57.1 Purpose

Turns tracing options off or on.

3.57.2 Syntax

```
EXECUTE set_options
WITH option='option_name',"value"=true|false
```

3.57.3 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 "Trace options for SET_OPTIONS" on page 341.
VALUE	B	T (TRUE) or F (FALSE)	TRUE turns tracing on. FALSE turns tracing off.

3.57.4 Return value

SET_OPTIONS returns a collection with on result object. The object has one Boolean property whose value indicates the success (TRUE) or failure (FALSE) of the operation.

3.57.5 Permissions

You must have Sysadmin or Superuser privileges to use this method.

3.57.6 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.

To turn off the nettrace option, disconnect all client sessions. After about one minute, the tracing turns off.

“Trace options for SET_OPTIONS” on page 341, 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. The trace messages, up to 2047 characters, 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 during startup mode.
debug	Traces session shutdown, change check, launch and fork information.
docbroker_trace	Traces connection broker information.
file_store_trace	Traces digital shredding of content files. Also traces content checking and duplication prevention operations.

Value for the OPTION argument	Meaning
i18n_trace	Traces client session locale and code page. 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 code page 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 OpenText Global Technical Services 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.
retention_trace	Turns on tracing for the following retention-related operations: • Assigning a retention policy to a document • Recording a retention period applied by a retention policy at the storage area level • Removing a retention policy from a document
rpctrace	Traces RPC calls.
sqltrace	Traces SQL commands sent to the underlying RDBMS for subsequent sessions, including the repository session ID and the database connection ID for each SQL statement.
ticket_trace	Traces import and export operations for the login ticket key and operations the use a single-use login ticket.
trace_authentication	Traces detailed authentication information.
trace_complete_launch	Traces Linux process launch information.

Value for the OPTION argument	Meaning
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.

3.57.7 Related administration methods

"LOG_OFF" on page 258
"LOG_ON" on page 259
"MODIFY_TRACE" on page 295

3.57.8 Examples

This example turns on SQL tracing:

```
EXECUTE set_options
WITH option='sqltrace',"value"=true
```

The following example turns on logon authentication tracing:

```
EXECUTE set_options
WITH option='trace_authentication',
"value"=true
```

This example turns off tracing for logon authentication:

```
EXECUTE set_options
WITH option='trace_authentication",
"value"=false
```

3.58 SET_STORAGE_STATE

3.58.1 Purpose

Takes a storage area offline, places an off-line storage area back online, or makes a storage area read-only.

3.58.2 Syntax

```
EXECUTE set_storage_state [[FOR] 'storage_area_obj_id']
[WITH store=storage_area_name{,argument=value}]
```

If you include the STORE argument, do not include the FOR clause.

3.58.3 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 offline.
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.
WORM	B	T (TRUE) or F (FALSE)	If set to TRUE, the storage state is WORM-enabled.

3.58.4 Return value

SET_STORAGE_STATE returns a collection with one result object. The object has one Boolean property whose value indicates the success (TRUE) or failure (FALSE) of the operation.

3.58.5 Permissions

You must have Sysadmin or Superuser privileges to use this method.

3.58.6 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 Documentum Servers, you must issue a connection request that bypasses the Documentum CM Server to use SET_STORAGE_STATE in the session.

3.58.7 Related administration methods

None

3.58.8 Example

The following example moves the storage area called manfred offline:

```
EXECUTE set_storage_state
WITH store = 'manfred', offline = T
```

3.59 SHOW_SESSIONS

3.59.1 Purpose

Lists information about all the currently active sessions and a user-specified number of historical sessions.

3.59.2 Syntax

```
EXECUTE show_sessions
```

3.59.3 Arguments

SHOW_SESSIONS has no arguments.

3.59.4 Return value

SHOW_SESSIONS returns a collection. Each query result object in the collection represents one currently active or inactive repository session. The properties of the objects contain information about the sessions. The properties returned by SHOW_SESSIONS are the same as those for LIST_SESSIONS. For a description of the properties, refer to "Complete information returned by LIST_SESSIONS" on page 254.

3.59.5 Permissions

Anyone can use this method.

3.59.6 General notes

SHOW_SESSIONS returns information about currently active sessions and historical (timed out) sessions. The information for each session is identical to the information returned by LIST_SESSIONS.

The maximum number of historical sessions returned by SHOW_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 SHOW_SESSIONS will not return any historical sessions older than 15 minutes.

The *OpenText Documentum Content Management - Server Administration and Configuration Guide (EDCCS250400-AGD)* contains the description of the server.ini file and the parameters you can set in it.

Note: You can use SHOW_SESSIONS to obtain the process ID if you need to shutdown or kill a session or server.

3.59.7 Related administration methods

"LIST_SESSIONS" on page 253

3.59.8 Examples

Refer to the syntax description for examples.

3.60 TRANSCODE_CONTENT

3.60.1 Purpose

Queues a transformation request for a content file to the Media Server.

3.60.2 Syntax

```
EXECUTE transcode_content FOR 'object_id'
 WITH [page=page_number,] [priority=priority_level,]
message='message',source_format='format_name',
target_format='format_name'
```

object_id is the object ID of the document that contains the content file.

3.60.3 Arguments

Table 3-59: 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.

Argument	Datatype	Value	Description
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. You must enclose the object ID in double quotes, and the entire value in single quotes. Refer to the usage notes for an example. Refer to the General Notes for a more detailed 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.

3.60.4 Return value

TRANSCODE_CONTENT returns a collection with one query result object. The object has two properties, result and result_id. The result property is a Boolean property whose value indicates the success (TRUE) or failure (FALSE) of the operation. The result_id property records the object ID of the queue item that is created as by the method.

3.60.5 Permissions

You must have at least Write permission on the object or Sysadmin privileges to use this method.

3.60.6 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 Transformation Services - Media is installed. TRANSCODE_CONTENT sets the following properties in the queue item:

Table 3-60: Queue item properties set by TRANSCODE_CONTENT

Property	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
sent_by	session user
priority	priority level identified in the REGISTER_ASSET call

The MESSAGE argument specifies the transformation operation to perform by specifying a profile. A profile is an XML document, stored in the repository, 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'
```

3.60.7 Related administration methods

"REGISTER_ASSET" on page 321

3.60.8 Example

```
EXECUTE transcode_content FOR '090000017456ae1c'
WITH page=0,priority=5,
message='-profile_id="090000018125ec2b"',
source_format='jpeg',
target_format='jpeg_lres'
```

3.61 UPDATE_STATISTICS

3.61.1 Purpose

Updates statistics for RDBMS tables in the repository.

3.61.2 Syntax

```
EXECUTE update_statistics WITH table_name='name'
[,count=integer][,extra_data=value]
```

3.61.3 Arguments

Table 3-61: 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 for the Oracle, SQL Server, and PostgreSQL databases. The argument is used differently for each database. Refer to the General Notes for details.

Argument	Datatype	Value	Description
EXTRA_DATA	S	*value*	The argument is used differently for each database. Refer to the General Notes for details.

3.61.4 Return value

UPDATE_STATISTICS returns a collection with one query result object. The object has one property, called result, that contains T if the method was successful or F if the method was unsuccessful.

3.61.5 Permissions

You must be a Superuser to execute this method.

Caution
Do not use this method unless directed to do so by OpenText Global Technical Services.

3.61.6 General notes

UPDATE_STATISTICS is used by the UpdateStatistics administration tool when the tool is run on any supported database.

3.61.6.1 The COUNT argument

This is an optional argument.

For Oracle, this argument 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 PostgreSQL, this argument enables the verbose tracing.

3.61.6.2 The EXTRA_DATA argument

The EXTRA_DATA argument is an optional argument.

For Oracle, 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:

```
EXECUTE update_statistics
WITH table_name='dm_sysobject_s',
extra_data='keywords,authors'
```

If you do not include EXTRA_DATA, all columns in the table are used.

For SQL Server, 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.

For PostgreSQL, this argument 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 (similar to Oracle).

3.61.7 Related administration methods

"REORGANIZE_TABLE" on page 324

3.61.8 Examples

This example updates statistics for the dm_sysobject_s table:

```
EXECUTE update_statistics
WITH table_name='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'
```

3.62 WEBCACHE_PUBLISH

3.62.1 Purpose

Invokes the dm_webcache_publish method to publish documents to a Web site. This administration method cannot be run using the EXECUTE statement.

3.62.2 Syntax

```
apply,session,webc_config_obj_id,
WEBCACHE_PUBLISH[,ARGUMENTS,S,argument_list]"
```

3.62.3 Arguments

Table 3-62: 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 entries for argument_list are described in "Valid arguments for ARGUMENTS" on page 354.

"Valid arguments for ARGUMENTS" on page 354, lists the valid entries for the ARGUMENTS argument.

Table 3-63: Valid arguments for ARGUMENTS

Argument	Value	Description
-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 do not include this argument, the entire source data set is refreshed. This option is ignored if -full_refresh is set to TRUE.
-full_refresh	TRUE or FALSE	When this is set to TRUE, the content and property 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 or FALSE	When 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.
-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 IDfSession.setServerTraceLevel method.

Argument	Value	Description
-recreate_property_schema	TRUE or FALSE	Performs 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_table	TRUE or FALSE	TRUE deletes 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_log	TRUE or FALSE	Creates a log file object in the repository for each publish operation. By default, this flag is TRUE for all full refresh and incremental publishing operations and FALSE when the method is issued for a single item.

3.62.4 Return value

The WEBCACHE_PUBLISH administration method returns a collection identifier for a collection with one property. The property, method_return_val, contains 0 if the operation was successful.

3.62.5 Permissions

You must have Sysadmin or Superuser privileges to use this administration method.

3.62.6 Generating a log file

When the operations generate a log file, the document is stored in the repository 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.

3.62.7 Example

```
dmAPIGet("apply,s0,080015b38001a43c,WEBCACHE_PUBLISH,
ARGUMENTS,S,-source_object 090015b38007bf19")
```

Chapter 4

Using DQL

This chapter introduces DQL and then provides usability information for DQL.

4.1 Introducing DQL

DQL is the acronym for Document Query Language, the SQL-like language that you use to query the objects in a repository. Using DQL, you can retrieve, update, and delete objects and create new objects. DQL also allows you to search indexed content and metadata (property values).

You can also use DQL to access registered tables—tables in the underlying RDBMS that are known to Documentum CM Server but that are not part of the repository. The *OpenText Documentum Content Management - Server Fundamentals Guide (EDCCS250400-GGD)* describes the database tables that make up a repository.

The basic DQL query statements retrieve information about the objects in a repository and manipulate those objects. "DQL basic query statements" on page 357, describes the basic query statements.

Table 4-1: DQL basic query statements

Statement	Description
SELECT	The SELECT statement is the primary query statement. SELECT statements return a wide variety of information from the repository. SELECT statements are not only stand-alone statements but are also used in other DQL statements. There are two variants of the SELECT statement. One is the standard SELECT statement, and the other is called an FTDQL SELECT statement. FTDQL queries are run entirely against the fulltext index, which provide performance benefits. However, the syntax for an FTDQL query is a subset of that for a standard query. For more information, refer to "Select" on page 120 or to the summary of the syntax in Appendix E, DQL quick reference on page 419.
UPDATE	The UPDATE statement changes the RDBMS table that underlies a registered table.

Statement	Description
UPDATE...OBJECT	The UPDATE...OBJECT[S] statement modifies objects in the repository. Using this statement, you can: • Set the value of a single-valued property • Append or insert values in a repeating property • Remove a value from a repeating property • Truncate a repeating property (remove all values) • Link an object to a folder or cabinet or unlink an object • Move an object to a new folder or cabinet
CHANGE...OBJECT	The CHANGE...OBJECT[S] statement moves one or more objects from one object type to another. With some constraints, you can change an object of a particular type to any type that is a subtype or supertype of the current type. For example, you can change an object of type dm_document to a user-defined document subtype.
INSERT	The INSERT statement adds a row to the RDBMS table that underlies a registered table.
DELETE	The DELETE statement destroys rows in the RDBMS table that underlies a registered table.
DELETE...OBJECT	The DELETE...OBJECT[S] statement deletes objects from the repository.

When you issue one of these statements, only those objects for which you have the appropriate permissions are affected. For example, if you issue an UPDATE...OBJECT statement to update document objects, only those documents for which you have at least Write permission are considered for updating. In addition, the statements have optional clauses that let you identify specifically which objects are the target of the operation.

4.2 Quoting object type and property names

Place double quotes around all references to object type names and property names in DQL queries. While not required, this best practice ensures that the names will not conflict with words reserved by DQL or the underlying RDBMS.

To encourage this practice, object type and property names are double quoted in all Documentum CM Server documentation.

4.3 NULLs, default values, and DQL

In releases prior to 6.6, OpenText Documentum CM did not allow you to assign NULLs as a property value because NULLs are used to terminate the lists of values in repeating properties. In 6.6, support for assigning NULLs as a property value was added. If you specify the SPACEOPTIMIZE keyword when creating or modifying a property, NULL values can be assigned to that property.

When you create an object, any properties that you do not set are assigned default values by the server. If a default value is defined in the data dictionary, the server assigns that value. Otherwise, the default value depends on the data type of the property. If SPACEOPTIMIZE was not specified, the default value from the table below is stored.

If SPACEOPTIMIZE was specified, then a property that you do not set is stored as NULL, and if you explicitly set a property to the default value in the table below, it is also stored as NULL. For strings, a single blank (' ') and the empty string (”) are both treated as the default value and are stored as NULL. This is different from previous behavior where the empty string is distinct from the default value.

For Oracle installations, SPACEOPTIMIZE can modify all properties. For SQL Server, only the character and string properties and the ID properties can be set to SPACEOPTIMIZE and only save space for NULL on VARCHAR.

Note: The recommendation is that ID attributes should only have SPACEOPTIMIZE applied when it is expected that most of the time the value will be NULLID (that is, 0000000000000000). This is because when a non-null ID value is stored, VARCHAR(16) causes 17 bytes of data to be allocated in SQL Server versus 16 for CHAR(16). This is why we do not put SPACEOPTIMIZE on r_object_id since every object has a non-null value.

“Default property values by datatype” on page 359, lists the default values by data type.

Table 4-2: Default property values by datatype

Datatype	Default value
String	A single blank
Numeric data type	0

Datatype	Default value
Date	The NULLDATE value: • For Oracle, it is 01/01/0001. • For SQL Server, it is 1/1/1753. • For PostgreSQL, it is 1/1/1753.
ID	'0000000000000000' (sixteen zeros)

4.3.1 Default values returned without SPACEOPTIMIZE

When you query a property containing a default value using DQL, the default values are returned in the following manner:

- The default character string value (a single blank) is returned as an empty string.
- DQL also returns a single blank found in a registered table column as an empty string.
- The default numeric value (zero) is returned as zero.
- NULLDATE values are returned as the word NULLDATE.

If you are using SQL (for example, in a user-written report writer), the default values are returned in the following manner:

- The default character string value (a single blank) is returned as a single blank.
- SQL also returns a single blank found in a registered table column as a single blank.
- The default numeric value (zero) is returned as zero.
- NULLDATE values are returned as the actual date (for example, 1/1/1 in Oracle).

4.3.2 Default values returned with SPACEOPTIMIZE

For single valued properties, default values stored as NULLs return NULLs. For repeating properties, the situation is a little more complicated.

In releases earlier than 6.6, a NULL could not be stored as a value of a repeating property, since the NULL indicted the end of the repeating values. For example, “Repeating values without SPACEOPTIMIZE” on page 361 shows an object with ID obj1 and five repeating integer attributes (properties). Property attr1 has three values, the first value being the default value, attr2 has one value, attr3 has two values, attr4 has three values with the last one being the default value, and attr5 has four values, the second and fourth have default values.

Table 4-3: Repeating values without SPACEOPTIMIZE

r_object_id	i_position	attr1	attr2	attr3	attr4	attr5
obj1	-1	0	1	2	3	4
obj1	–2	1	NULL	1	4	0
obj1	–3	2	NULL	NULL	0	2
obj1	–4	NULL	NULL	NULL	NULL	0

In the next table, “Repeating values with SPACEOPTIMIZE” on page 361, the same object is shown as it would if SPACEOPTIMIZE had been specified on these five repeating integer properties. Default values are shown in bold. Note that the first value for attr1 and the second value for attr5 are NULL, and the third value for attr4 and the fourth value for attr5 are zero. In this case, a default value might be returned as a NULL or as a zero. Since either value could be returned for the default, there is new behavior for NULL comparisons in 6.6. See section “Testing for default and NULL values” on page 362, for a description of this new behavior.

Table 4-4: Repeating values with SPACEOPTIMIZE

r_object_id	i_position	attr1	attr2	attr3	attr4	attr5
obj1	-1	*NULL*	1	2	3	4
obj1	–2	1	NULL	1	4	*NULL*
obj1	–3	2	NULL	NULL	*0*	2
obj1	–4	NULL	NULL	NULL	NULL	*0*

If a new value is added to attr4 and a new default value is added to attr5, the following changes shown in “Added repeating values with SPACEOPTIMIZE” on page 361 will occur. As before, the default values are shown in bold. Notice that the third value of attr4 and the fourth value of attr5 have been changed to NULL.

Table 4-5: Added repeating values with SPACEOPTIMIZE

r_object_id	i_position	attr1	attr2	attr3	attr4	attr5
obj1	-1	*NULL*	1	2	3	4
obj1	-2	1	NULL	1	4	*NULL*
obj1	-3	2	NULL	NULL	*NULL*	2
obj1	-4	NULL	NULL	NULL	6	*NULL*
obj1	-5	NULL	NULL	NULL	NULL	*0*

So, in summary, for repeating properties that use SPACEOPTIMIZE, all default values are stored as NULLs, unless the last value is a default value, and that property uses NULLs to fill out its column of repeating values. In that case, the default value is stored as was done in releases prior to 6.6, and is still done for properties that do not use SPACEOPTIMIZE.

4.3.3 Testing for default and NULL values

OpenText Documentum CM provides predicates for use in WHERE clauses to test for the presence of default values or NULLs. “Predicates that test for NULL and default values” on page 362, briefly describes these predicates. “Predicates” on page 29 contains more information about predicates.

Table 4-6: Predicates that test for NULL and default values

Predicate	Description
IS [NOT]NULL	Used only for columns in registered tables. Tests for the presence of NULL.
ANY...IS [NOT]NULL	Tests for the presence of the NULL terminator in a repeating property. This is primarily useful if you are testing values in two or more repeating properties at corresponding index levels.
[ANY]...IS [NOT]NULLDATE	Tests for the presence of the NULLDATE or a NULL. Use the ANY option if the property is a repeating property.
[ANY]...IS [NOT]NULLSTRING	Tests for the presence of the default string value (a single blank) or a NULL in a property. Use the ANY option if the property is a repeating property.
[ANY]...IS [NOT]NULLINT	Tests for the presence of the default numeric value (a zero) or a NULL in a property. Use the ANY option if the property is a repeating property.
[ANY]...IS [NOT]NULLID	Tests for the presence of the default ID value ('0000000000000000') or a NULL in a property. Use the ANY option if the property is a repeating property. NULLID is available from the 6.6 release.

In the 6.6 release, true NULL storage was introduced. A server.ini flag was also added to control how predicates that qualify a single-valued property behave. The flag, *extend_default_predicate_to_null*, is true by default. When this flag is true, for any single-valued property that uses SPACEOPTIMIZE, a query that uses the default value to qualify the property will be extended to also cover the NULL value as well.

For example, for an integer property attr1, that uses SPACEOPTIMIZE, the following predicate:

```
attr1 = 0
```

will be converted to:

```
(attr1 = 0 OR attr1 IS NULL)
```

so that this property will be qualified as if it has the default value.

For more information about querying repeating properties, refer to "Repeating properties in queries" on page 364.

4.3.4 Default values and aggregate functions

Aggregate functions are functions that operate on a group of values and return a single value. For example, count is an aggregate function. It counts the values in a property and returns the number.

Aggregate functions do not ignore the default values assigned to a property. These values are included in any calculations performed by the function. For example, suppose a repeating property value can range from 1 to 10. If an object has three actual values (4, 7, and 6) and one default value (one zero) for the property, the AVG function adds 4, 7, 6, and 0, then divides the total by 4 to obtain the average:

```
(4 + 7 + 6 + 0)/4 = 4.25
```

If the function ignores the default value, the average is:

```
(4 + 7 + 6)/3 = 5.66
```

As another example, look at the following SELECT statement:

```
SELECT COUNT(DISTINCT ratings) FROM “recipe”
WHERE “object_name” = ‘lemon cake’
```

This statement (assuming the property values for ratings described for the previous example) returns the number 4 because the default value is counted.

You can use the WHERE clause to exclude the default value when you are selecting aggregate functions. For example, the following statement returns the correct number of ratings for the lemon cake recipe:

```
SELECT COUNT(DISTINCT rating) FROM “recipe”
WHERE “object_name” = ‘lemon cake’
AND “rating” IS NOT NULLINT
```

4.3.5 Sorting and nulls

Different databases handle NULLs differently when they are sorting values in a table. Here is how each repository handles NULLs if the sort order is ascending:

- In Oracle, NULLs are sorted to the bottom of the list.
- In SQL Server, NULLs are sorted to the top of the list.
- In PostgreSQL, NULLs are sorted to the top of the list.

4.4 Repeating properties in queries

Repeating properties store multiple values. In OpenText Documentum CM, many object types have repeating properties. For example, because a document can have multiple authors, the authors property of the dm_document type is a repeating property.

You can reference repeating properties in:

- The selected values list in a SELECT statement
- WHERE clause qualifications for SELECT, UPDATE...OBJECT, CHANGE...OBJECT, and DELETE...OBJECT statements
- The update clauses of the CREATE...OBJECT, UPDATE...OBJECT, and CHANGE...OBJECT statements

For information about using a repeating property in SELECT statement, as a selected value or in the WHERE clause qualification, refer to “Select” on page 120.

4.4.1 Modifying repeating attributes

You can use the UPDATE...OBJECT statement to add, delete, and modify individual values for repeating properties. In most of these operations, you must specify the *index position* of the value you want to change. The index indicates the value's position in the list of values assigned to the repeating property. Index numbers begin with zero for the first value and increment by 1 for each additional value.

For example, suppose the a recipe object type has a repeating property called ingredients. One recipe has the following ingredients: eggs, sugar, cream cheese, and amaretto. Assuming the ingredients were added to the ingredients property for a recipe object in the order listed, they would have the following index values:

ingredient[0] (eggs)
ingredient[1] (sugar)
ingredient[2] (cream cheese)
ingredient[3] (amaretto)

Index positions are always specified inside square brackets after the name of the property.

4.4.1.1 Adding new values

To add a value to a repeating property, you can:

- Insert the new value, which lets you choose where to put the new value in the property’s value list
- Append the value, which automatically places the value at the end of the property’s value list

4.4.1.1.1 Inserting values

When you want to control where the new value is placed in the repeating property, use the insert option as the update clause in the UPDATE...OBJECT statement. The syntax is:

```
UPDATE object_type OBJECTS
INSERT property_name[x] = value...
```

For example, suppose that the authors of the Espresso Cheesecake recipe forgot to include one of the ingredients for the crust. The ingredients for the crust are listed ahead of the ingredients for the filling in the document. Consequently, the authors want to insert the forgotten ingredient ahead of the filling ingredients so that it appears with the other crust ingredients. The ingredients for the crust occupy positions 0 through 3 in the ingredients property, and the filling ingredients begin at position 4. The following statement inserts the forgotten ingredient at position 4.

```
UPDATE “recipe” OBJECTS
INSERT ingredients[4] = ‘2 T ground bittersweet chocolate’
WHERE “title” = ‘Espresso Cheesecake’
```

The server inserts the requested value and renumbers all other values from position 4 on. The filling ingredients now begin at position 5.

Inserting values into a repeating property never overwrites a current value. Any value currently at the specified insertion point and any following values are always renumbered.

If you do not specify an insertion point, the server automatically inserts the new value in the first position (*property_name*[0]).

You can identify the value you want to insert either as literal value or by using a subquery. If you use a subquery, it can return only one value. If it returns multiple values, the INSERT statement fails with an error.

4.4.1.1.2 Appending values

When you append values, the server automatically adds the new value to the end of the list of values in the property. You do not have to specify an index value when you append. To illustrate, suppose the authors want to add another ingredient to the Espresso Cheesecake recipe: strawberries to be used as a garnish. To append this ingredient, they use the following statement:

```
UPDATE “recipe” OBJECT
APPEND “ingredients” = ‘1 pint strawberries’
WHERE “title” = ‘Espresso Cheesecake’
```

You can identify the appended value as a literal value or by using a subquery. For example, perhaps your company is reorganizing and employees currently working for Mr. Rico are being moved to a group called frontline. The following statement uses a subquery to find those employees and append them to the list of users in the frontline group:

```
UPDATE “dm_group” OBJECTS
APPEND “users_names”=(SELECT “user_name”
FROM “dm_user”,“employees” e
 WHERE e.“manager”=‘rico’ AND “r_is_group”=F)
WHERE “group_name” = ‘frontline’
```

When using a subquery, you can specify the maximum number of values that you want to append. The syntax in the update clause is:

```
APPEND n property_name = subquery
```

where *n* represents the maximum number of appended values.

Note: To append more than 20 users, use the following query format:

```
update dm_document OBJECTS APPEND 1000 authors=
(SELECT “user_name” FROM “dm_user”) where
r_object_id = ‘090e266c80001675’
```

4.4.1.2 Updating values

To update a value, use the set option as the update clause.

For example, the authors of Heavenly Cheesecakes have decided to update one of the recipes in the book. They want to replace the cream cheese requirement with a lower-fat substitute—ricotta cheese. Knowing that cream cheese is the third ingredient in the ingredients list, they use the following statement to make the substitution:

```
UPDATE “recipe” OBJECT
SET ingredients[2] = ‘1 lb ricotta cheese’
WHERE “title” = ‘mocha cheesecake’
```

Note that the index value for cream cheese is [2]. This is because index values are counted from zero, so the first two ingredients (eggs and sugar) have index values of zero and one ([0] and [1]), respectively.

4.4.1.3 Deleting values

To delete a value in a repeating property, use the remove option as the update clause and specify the index value associated with the value. To illustrate, the following example removes the fifth ingredient in a list of ingredients for brownies:

```
UPDATE “recipe” OBJECT
REMOVE ingredients[4]
WHERE “object_name” = ‘Mandarin Brownies’
```

If you do not specify which value to remove, the system automatically removes the first ([0]) value.

4.4.2 Forcing index correspondence in query results

Note: The query syntax described in this section may not be used in an FTDQL SELECT statement. For information about FTDQL SELECT statements, refer to “Select” on page 120.

In some queries, you may only want objects for which the repeating property values are in the same relative positions. For example, perhaps you want only those recipes for which a particular author and keyword occupy the same index position.

To require index correspondence for expressions referencing repeating properties, use the following syntax:

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

To force index correspondence, the predicates must be ANDed inside the parentheses. Using the OR operator to link the predicates inside the parentheses increases the query’s performance but does not force index correspondence. The query returns any object that contains one of the ORed values.

To illustrate, assume that the keywords untried, tested, approved, and rejected are assigned to recipes in various stages of acceptance and that these keywords are always placed in the first position in the keywords list, to correspond to the implicit version label (the numeric label) associated with the recipe. The following statement finds only original recipes (version 1.0) that were rejected by the testers:

```
SELECT “r_object_id”, “object_name” FROM “recipe”
WHERE ANY (“r_version_label” = ‘1.0’
AND “keywords” = ‘rejected’)
```

In the above example, for the recipes returned, r_version_label[0] = 1.0 and keywords[0] = rejected.

You cannot specify which position is searched when you force index correspondence. You can only specify that the values be found in the same position. In the above example, it was easy to know that the values are in the first position for all pairs because the implicit version label is always stored in the first position. However, for other repeating properties this will not be true. For example, look at the following statement:

```
UPDATE “recipe” OBJECTS
SET “subject” = ‘lowfat meals’
WHERE ANY (“ingredients” IN (‘skim milk’,‘margarine’)
AND “keywords” = ‘fat free’)
```

The statement updates the subject property for any recipe for which skim milk and fat free or margarine and fat free occupy the same respective positions within their individual properties. For one recipe, the positions might be the fourth: ingredients[3] = margarine and keywords[3] = fat free. For another, the values might be found in the seventh position: ingredients[6] = skim milk and keywords[6] = fat free. In all instances, the values are at the same index position within their properties.

It is possible to combine both forms of the syntax. For example:

```
SELECT “r_object_id”, “title” FROM “recipe”
WHERE ANY “ingredients” IN (‘cream cheese’,‘chocolate’)
AND ANY(“authors” = ‘daphne’ AND “keywords” IN (‘cheesecake’,‘brownies’,‘mousse’))
```

This statement returns all recipes that have cream cheese or chocolate as an ingredient and also have the author daphne paired with the keyword cheesecake, brownies, or mousse.

4.4.3 Performance note for SQL Server users

DQL turns repeating property predicates (and the FOLDER predicate) into ORed subselects internally. Queries that contain ORed subselects run slowly against SQL Server, and performance degrades in direct proportion to the number of SysObjects in your repository.

To improve performance, here are some suggested alternative ways to formulate queries:

Instead of:

```
SELECT “r_object_id” FROM “dm_sysobject”
WHERE ANY “authors” = ‘a’ OR ANY authors = ‘b’
```

Use:

```
SELECT “r_object_id” FROM “dm_sysobject”
WHERE ANY (“authors” = ‘a’ OR “authors” = ‘b’)
```

Instead of:

```
SELECT “r_object_id” FROM “dm_sysobject”
WHERE FOLDER (‘/Temp) OR FOLDER (‘/System’)
```

Use:

```
SELECT “r_object_id” FROM “dm_sysobject”
WHERE FOLDER (‘/Temp', ‘/System’)
```

4.5 Querying virtual documents

Virtual documents are documents that contain other documents. The documents they contain can be simple documents or other virtual documents. Nesting virtual documents inside other virtual documents creates a hierarchy of components within the top-level virtual document. OpenText Documentum CM allows you to nest virtual documents to any depth you choose.

To identify the virtual document you want to query, you can use either the IN DOCUMENT clause or the IN ASSEMBLY clause in a SELECT statement. The IN DOCUMENT clause allows you to choose which set of the document's components to query at runtime. The IN ASSEMBLY clause directs the query to the virtual document's snapshot, which is a previously selected set of the document’s components.

If you are querying a virtual document in an UPDATE...OBJECT or DELETE...OBJECT statement, you must use the IN ASSEMBLY clause.

Both the IN DOCUMENT and IN ASSEMBLY clauses include the optional keyword DESCEND. This keyword directs the server to search any components contained by components that are themselves virtual documents. The only exception is if the component is a reference link. In such cases, the server cannot search for the component’s contained components. The search returns only the reference link.

OpenText Documentum CM also allows you to specify one particular component of a virtual document as the target of a query. This feature can make it easy for users who are working on a virtual document to pull out a specific part of the document for work. The NODE option of the IN ASSEMBLY clause implements this feature.

4.6 Full-text searching and virtual documents

Like simple documents, virtual documents can have associated content files. Marking a virtual document for indexing directs Documentum CM Server to index the content files associated with the virtual document—*not* the components of the virtual document.

In SELECT statements, if you include a SEARCH clause and an IN DOCUMENT clause, the server first searches to see whether *the content files of the specified virtual document*meet the full-text search criteria. Then, as the server assembles the virtual document, the criteria in the SEARCH clause are applied to *the content files* associated with each component.

If you want to conduct fulltext searches on the assembled document components, you must create a file from the assembled document and associate it with the virtual document as a content file. Use the following procedure:

To index an assembled document as a whole:

1. Assemble the document.

2. Print the document to a file.
3. Use Setfile or Setcontent to associate the resulting file with the virtual document as a content file.

   Now you have a content file for the virtual document that represents the assembled document and that can be indexed.

4.7 Querying registered tables

Registered tables are tables in the underlying RDBMS that have been registered with the repository. You must register any RDBMS table that you wish to query using DQL. Registering a table creates an object of type dm_registered for that table.

To obtain information about the repository object that represents the table, specify the dm_registered type in the FROM clause of the SELECT statement. The server searches for the object representing the table. To query the table itself, specify the registered table’s name in the FROM clause. When you query the table, the server searches the actual underlying RDBMS table—*not* the dm_registered object type.

4.7.1 Referencing registered tables in queries

To avoid ambiguity in a query, it is often useful to reference a database table by its fully qualified name; that is, *owner_name.table_name*.

The owner name is the name of the person who created the table. Tables created by the system when a repository is created (for example, dm_sysobject_r or dm_sysobject_s) are owned by the repository owner. For these tables, Oracle uses the value in the repository owner’s user_db_name property as the owner name. PostgreSQL has the same behavior as Oracle. SQL Server prefixes the names of all tables created by the repository owner with the alias dbo. This alias makes it possible to write applications that are portable across SQL Server databases.

For application portability across all repositories, OpenText Documentum CM provides an alias, dm_dbo, that you can substitute for the repository owner's name in any fully qualified reference to registered tables. For SQL Server, when referencing registered tables, the aliases dm_dbo and dbo are equivalent.

4.7.2 Security controls

Access to a registered table is controlled by the object-level permissions and table permits defined for the table's dm_registered object. The object-level permissions must give you at least Browse access to the dm_registered object and the table permits must give you permission for the operation that you want to perform. For example, if you want to update a table, the table permits must grant you DM_TABLE_UPDATE permission. Table permits are defined for three user levels: owner, group, and world. The *OpenText Documentum Content Management - Server Administration and Configuration Guide (EDCCS250400-AGD)* contains complete information about table permits and OpenText Documentum CM security.

Additionally, the user account under which Documentum CM Server is running must have the appropriate RDBMS permission to perform the requested operation on the specified table. The actual name of this permission depends on your RDBMS. For example, if you want to update the table, the server account must have permission in the RDBMS to update the table.

All three of these conditions must be met to gain access to a registered table. Even if a user has permission to access the underlying table through the RDBMS, the user will not have access through DQL unless the object-level permissions and table permits on the dm_registered object permit access. Similarly, a user might have the correct object-level permissions and table permits, but if the server's user account in the RDBMS does not have permission, then access is denied.

4.7.2.1 Default object-level permissions and table permits

The REGISTER statement automatically sets the object-level permissions for a table's dm_registered object to default values. The statement also sets the default table permit to SELECT for the owner (group and world have no default table permit).

You can change the object-level permissions and table permits by setting the appropriate properties directly. The *OpenText Documentum Content Management - Server Administration and Configuration Guide (EDCCS250400-AGD)* contains more information about these properties.

4.8 Caching queries

Some queries return the same results every time you run the query. For example, a payroll application may ask for the names and social security numbers of the employees in the company. Although the query results may change over a long period of time, they may not change from week to week. Rather than incur the performance cost of rerunning the query that returns the users and social security numbers each time the payroll application executes, you can cache the query results on the client host. The *OpenText Documentum Content Management - Server Fundamentals Guide (EDCCS250400-GGD)* contains more information about persistent caching.

4.9 Privileges, permissions, and queries

DQL query statements affect only those objects for which users have the appropriate permissions or privileges.

When a SELECT statement references a SysObject type (dm_sysobject or any of its subtypes), by default, only those objects of the referenced type for which the user has at least Browse permission are retrieved. If the SELECT statement includes a SEARCH clause that accesses the object's content, then the user must have at least Read permission. However, the SELECT statement supports an option to allow you to specify another base permission level. For example, you might use a query that returns only objects for which a user has at least Relate permission.

If the object type specified in the SELECT statement is modified by the keyword PUBLIC, then the statement retrieves only objects that have a world permission of at least Browse (or Read for full-text searches) or objects that are owned by the user.

When a SELECT statement references a registered table, the user must have at least Browse permission on that registered table for the query to succeed. In addition, Documentum CM Server must have the SELECT privilege in the RDBMS.

The Superuser user privilege bypasses all OpenText Documentum CM security checks. However, Documentum CM Server must still have the SELECT privilege in the RDBMS for a superuser to query registered tables.

You must have Write permission to update an object with the UPDATE...OBJECT statement. You must have Delete permission for an object to delete it using the DELETE...OBJECT statement.

Appendix A. Using DQL hints

This appendix describes how DQL hints are implemented and how to use them in your queries.

A.1 General guidelines 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.

You can use any of the hints except ROW_BASED and RETURN_RANGE in an FTDQL query. The ROW_BASED hint may not be included in an FTDQL query and RETURN_RANGE is not FTDQL compliant.

Additionally, you may not use the ROW_BASED hint in query that includes a lightweight object type in the FROM clause.

A.2 CONVERT_FOLDER_LIST_TO_OR

A new DQL hint is added as a platform compatibility update to Documentum CM Server in the 6.7 SP1 release. It provides backward compatibility to the behavior of release 6.0 and earlier. From the 6.5 release, a DQL FOLDER clause like:

```
... FOLDER ('/Temp', '/Temp/test_folder_1', '/System')...
```

is translated to an SQL statement with a clause like:

```
... dm_sysobject_r2.i_folder_id IN
('0c01068980000106', '0b01068980004509',
'0c001068980000107') ...
```

In some cases, this causes performance degradation compared with the translation used in the earlier release. The earlier translation of the FOLDER clause is:

```
... dm_sysobject_r2.i_folder_id = '0c01068980000106'

or dm_sysobject_r2.i_folder_id = '0b01068980004509' or dm_sysobject_r2.i_folder_id =
'0c001068980000107' ...
```

To get the earlier translation, use the CONVERT_FOLDER_LIST_TO_OR DQL hint. For example:

```
... FOLDER ('/Temp', '/Temp/test_folder_1', '/System')
ENABLE(CONVERT_FOLDER_LIST_TO_OR) ...
```

A.3 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.

A.3.1 SQL Server, the hint, and subqueries

If you use this hint on query against a SQL Server database and the query includes a subquery, the hint is not applied to the subquery.

A.4 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.

Note: Not supported for PostgreSQL.

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)
```

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 Documentum CM 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.

A.5 FTDQL and NOFTDQL

The FTDQL hint supports the FTDQL functionality. If you include this hint in query, Documentum CM Server attempts to execute the query as an FTDQL query. If the remaining syntax in the query conforms to the required syntax for an FTDQL query, the query is executed as an FTDQL query. If the syntax does not conform to FTDQL query rules, an error is returned.

If you include the NOFTDQL hint, the query is run as a standard SELECT query whether the syntax satisfies the FTDQL query rules or not.

A standard query queries both the fulltext index and the database. Such a query typically contains a SEARCH clause and a WHERE clause. If the WHERE clause is not compliant with the rules of FTDQL or the query contains an explicit ENABLE(NOFTDQL), Documentum CM Server executes the SEARCH clause against the fulltext index and the WHERE clause against the database and then returns the intersection of the results.

A.6 FT_CONTAIN_FRAGMENT

The FT_CONTAIN_FRAGMENT hint expands the range of matches for full-text searches that include a search clause of the form LIKE '%*string*%' clause, such as:

```
subject LIKE '%work%'
```

When the query includes the DQL hint FT_CONTAIN_FRAGMENT, this clause returns all objects whose subject contains “work” in any position, including where the string appears as part of a larger string. For example, it returns any object whose subject contains “networking” or “workers”. However, if the query does not include this hint (the default behavior), a full-text search that uses this clause returns only those objects whose subject contains “work” (or a variant form of the word if grammatical normalization is enabled) as a separate word. Limiting the results to separate word matches improves the performance of the query. From the 6.0 release, the default behavior was changed. In previous releases, the default was to return objects as though FT_CONTAIN_FRAGMENT was specified.

The FT_CONTAIN_FRAGMENT hint affects only full-text searches. Database searches return partial string matches regardless of this hint. The hint only applies to search clauses that include a percent sign both before and after the literal string; The behavior for `LIKE '%work'` and `LIKE 'work%'` are unaffected.

A.7 GENERATE_SQL_ONLY

The GENERATE_SQL_ONLY hint returns the SQL statement for a particular DQL statement (Select statement only) without executing the statement on the database. This hint enables you to verify the SQL statement for a DQL statement without querying the data on the database.

A.8 GROUP_LIST_LIMIT N

The GROUP_LIST_LIMIT hint may improve query performance when a user is a member of a large number of groups. For example, suppose a user who is a member of 500 groups executes the following query:

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

When this query is translated to SQL, a subquery is needed to perform the ACL checking because the default group list limit for queries is 250. The performance is slower due to the need for the subquery. You can use the DQL hint to override the default value for group list limit. For example:

```
SELECT r_object_id FROM dm_document
WHERE object_name LIKE '%dm%'
ENABLE(GROUP_LIST_LIMIT 600)
```

Using this hint overrides the default value and the DM_GROUP_LIST_LIMIT environment variable, if that is set, as well as flushes and rebuilds the group list cache in the new size. For users whose group membership number is below the specified limit, generated SQL statements display group names inline without subqueries.

A.9 HIDE_SHARED_PARENT DQL hint

The HIDE_SHARED_PARENT DQL hint directs Documentum CM Server to return only the rows in the query results that are not shared parents. In order to accomplish this, the server will add two additional attributes, r_object_id and r_object_type to the SQL statement select list, if not already there (only applies to dm_sysobject or any of its subtypes). The server will run the SQL query against the database and then for each qualified row it will get the value of r_object_type, fetch the type object and check to see if it is a shareable type. For non-shareable types it will just return the row, but for shareable types it will do the following:

- Look in the sysobject cache for the r_object_id object, or issue an SQL statement if the object is not in the cache, and examine the i_sharing_type.
- Return the row if i_sharing_type is empty, or skip the row if it is not empty (indicating that the row is from a shared parent).

The results of a query that uses the HIDE_SHARED_PARENT DQL hint will not contain any shared parents. A side effect of this hint is that queries will show shared parents without child objects, but the shared parents will disappear from the results when a child LWSO is attached. Webtop uses this hint by default.

This hint does affect performance, since there are the additional checks required to determine whether the result contains shared parents.

A.10 IN and EXISTS

IN and EXISTS are mutually exclusive hints that you can use in a standard WHERE clause, in a repeating property predicate that contains a subquery. These hints may not be used in a WHERE clause in an FTDQL query.

The syntax is:

```
WHERE ANY [IN|EXISTS] property_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 Documentum CM Server translates the query, it includes one or the other automatically. Which hint Documentum CM Server chooses to include is determined by the indexes present in the repository, the properties 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 property predicate that uses a subquery is performing badly, run the query and examine the generated SQL statement to determine which hint Documentum CM Server is adding to the generated SQL and note the performance time. Then, rewrite the DQL query to include the hint that the server is not 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.

A.11 OPTIMIZE_ON_BASE_TABLE

The OPTIMIZE_ON_BASE_TABLE hint can optimize queries that only reference properties contained within one type in the type hierarchy.

DQL statements that do not use this hint query against the view of the table for the type (the _sp or _rp view). When the type hierarchy is deep, or the number of instances are large, a query run against the view can be slow. In contrast, if all the attributes in the query are contained in the table for the type, a query run only against that table can be much faster. Since the properties are contained in that table, there is no need for the joins required to query attributes in the tables of the supertype. You must evaluate this hint for your own environment before you decide to use it in production.

The OPTIMIZE_ON_BASE_TABLE hint has the following effects for these cases:

- Case 1: The type in the FROM clause does not have any supertype. For this case, the hint has no effect, and the _sp and/or _rp views are used.
- Case 2: The referenced properties in the select list, WHERE clause, ORDER BY clause, or GROUP BY clause, are from more than one type. For this case, the hint has no effect, and the _sp and/or _rp views are used.
- Case 3: The type is not a subtype of dm_sysobject. A) If all properties are from the type specified in the FROM clause, then the _s and _r tables of the type are joined (or only the _s table if all attributes are single-valued). B) If all properties are from a supertype of the type specified in the FROM clause, then the relevant _s and/or _r base table is used, plus the _s table of the referenced type is joined with the _s table of the type in the FROM clause.
- Case 4: The type is a subtype of dm_sysobject. A) If all properties are from the type specified in the FROM clause, then the hint has the same effect as for Case 3A, but if an ACL predicate is needed (for non-superusers), or a version clause is needed (when DELETED is not specified), then the dm_sysobject_s table is also joined for the query. B) If all properties are from the supertype of the type specified in the FROM clause the effect is the same as for Case 3B, but if an ACL predicate or version clause is needed, then the dm_sysobject_s table is also joined for the query.

Here is an example of *Case 3A*, all properties from the specified type:

```
SELECT group_class, groups_names
FROM dm_audittrail_group
enable(OPTIMIZE_ON_BASE_TABLE)
```

And this listing is the resulting SQL:

```
SELECT all dm_audittrail_group.group_class, dm_repeating.groups_names
FROM dm_audittrail_group_s  dm_audittrail_group,
dm_audittrail_group_r dm_repeating
WHERE dm_repeating.r_object_id=dm_audittrail_group.r_object_id
```

Here is an example of *Case 3B*, all properties from the supertype of the specified type:

```
SELECT event_name, object_name
FROM dm_audittrail_group
enable(OPTIMIZE_ON_BASE_TABLE)
```

And this listing is the resulting SQL:

```
SELECT all dm_audittrail_group.event_name, dm_audittrail_group.object_name
FROM    (select dm_opt_base.* from dm_audittrail_s
dm_opt_base, dm_audittrail_group_s dm_opt_from
where dm_opt_base.r_object_id = dm_opt_from.r_object_id)
dm_audittrail_group
```

A.12 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.

Note: Not supported for PostgreSQL.

On SQL Server, you can include an integer value (as N) to define how many rows you want returned quickly. On Oracle, the number is ignored.

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 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.

A.13 RETURN_RANGE

The RETURN_RANGE hint specifies which rows are returned by a query sorted by the returned values of specified properties. This hint is provided as a general way to paginate the results of a query. The syntax of this hint is:

```
RETURN_RANGE starting_row ending_row [optimize_top_row] 'sorting_clause'
```

Table A-1: RETURN_RANGE argument descriptions

Variable	Description
starting_row	Specifies the starting row number to return from the qualified rows.
ending_row	Specifies the ending row number to return from the qualified rows.
optimize_top_row	Specifies the number of top rows to be optimized by the database optimizer. This parameter is optional. If not specified, then there is no corresponding hint generated for the converted SQL statement.
sorting_clause	Specifies the attribute and its sorting sequence used to determine the range. It defines the sequence of the qualified results.

The syntax of the sorting clause is:

```
'attribute_name[ASC|DESC][,attribute_name[ASC|DESC]...]'
```

Table A-2: ***sorting_clause*** **argument descriptions**

Variable	Description
attribute_name	Specifies which attribute is used to sort the qualified rows.
ASC	Sorts in ascending order. This is the default if no order is specified.
DESC	Sorts in descending order.

The RETURN_RANGE hint can appear in the outermost query and in subqueries. When in the outermost query, RETURN_RANGE honors the object mode, like the RETURN_TOP hint. In other words, if the ROW_BASED hint is used, then RETURN_RANGE returns the top rows as specified, otherwise, it returns the specified top rows or objects based on the server.ini settings. If the RETURN_RANGE hint appears in a subquery, it will only perform on the row level.

A.14 RETURN_TOP N

The RETURN_TOP N hint, by default, limits the number of rows returned by a query. Since an object with repeating properties may consist of multiple rows, this hint may return fewer than N objects.

In the 6.5 SP2 release, a server.ini flag was added, return_top_results_row_based, true by default, that controls whether RETURN_TOP returns the number of rows or the number of objects specified. Setting this flag to false (return_top_results_row_based = F), causes the returned results to be limited by the number of objects.

A.14.1 More details about RETURN_TOP N

DQL is an object based query language - and we need to understand whether RETURN_TOP 10 means to return 10 rows, or 10 objects. Databases are not aware of OpenText Documentum CM objects. 10 rows does not always equal 10 objects. If a DQL query contains repeating attributes, each database row returned does not translate into a separate OpenText Documentum CM object. If we make RETURN_TOP 10 mean top 10 objects, we must limit results at the Documentum CM Server level. If we mean RETURN_TOP 10 to mean top 10 rows, we can limit at the database level (for more efficiency). A query containing only single valued attributes will behave the same for object or row-based results.

A.14.2 Database-specific implementations

The following sections describe how RETURN_TOP is handled for individual databases.

A.14.2.1 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, if 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
```

A.14.2.1.1 Subqueries and the hint

If the query includes a subquery, the hint is not applied to the subquery.

A.14.2.2 Oracle

If you include RETURN_TOP in a query running against Oracle, the returned results are not limited at the database level, but by Documentum CM Server itself. Consequently, using RETURN_TOP on Oracle results in no database performance gains, but may reduce network traffic.

Additionally, in Oracle implementations prior to the 6.5 release, RETURN_TOP *N* returned the number of objects specified, not the number of rows. Use the server.ini flag, return_top_results_row_based, in releases 6.5 SP2 and later to cause this same behavior.

A.14.2.3 PosgreSQL

Including RETURN_TOP adds the Postgres LIMIT hint to the generated SQL statement. For example, if you issue the following DQL statement:

```
SELECT user_name FROM dm_user ENABLE (RETURN_TOP 3)
```

The generated SQL for SQL Server is:

```
select user_name from dm_user_s LIMIT 3
```

If a limit count is given, no more than that many rows is returned (but possibly less, if the query itself yields less rows) and is used to limit the size of a result set.

A.14.3 Effects 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 fulltext index level. If the searched documents are not all public, then the limits are imposed when Documentum CM Server performs the security checking on the returned results.

A.14.4 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.

A.15 ROW_BASED

The ROW_BASED hint changes both the way query results are returned and the syntax rules for the query itself.

This hint may not be used in FTDQL queries or queries that reference a lightweight object type in the FROM clause.

A.15.1 Effects on returned results

The ROW_BASED hint forces Documentum CM Server to returns query results in a row format, as opposed to an object-based format. The difference is most readily apparent if the query selects values from a repeating property. In an object-based format, the server returns all selected repeating values for a particular object in one query result object. In a row-based format, the server returns each selected repeating property value in a separate query result object.

For example, by default the following query returns results in an object-based format:

```
SELECT "r_object_id","title","authors" FROM "dm_document"
WHERE "subject"='new_book_proposal'
```

That query returns the authors values as a list of authors in one query result object for each returned object. “Example of object-based query results” on page 383, shows how the results are returned. Each row in the table represents one query result object and each column is one property in the query result objects.

Table A-3: Example of object-based query results

r_object_id	title	authors
090000015973a2fc	Our Life and Times	Jennie Doe Carol Jones Hortense Smith
0900000123ac12f6	Life of an Amoeba	James Does Jules Doe

There is one query result object (one row) for each object returned by the query. Now, add the ROW_BASED hint to the query:

```
SELECT "r_object_id","title","authors" FROM "dm_document"
WHERE "subject"='new_book_proposal'
ENABLE(ROW_BASED)
```

"Example of row-based query results" on page 384, illustrates how Documentum CM Server returns the results for that query.

Table A-4: Example of row-based query results

r_object_id	title	authors
090000015973a2fc	Our Life and Times	Jennie Doe
090000015973a2fc	Our Life and Times	Carol Jones
090000015973a2fc	Our Life and Times	Hortense Smith
0900000123ac12f6	Life of an Amoeba	James Doe
0900000123ac12f6	Life of an Amoeba	Jules Doe

There is one query result object (one row) for each repeating property value returned. The returned repeating property values for each object are not aggregated into one result object.

A.15.2 Effects on query syntax rules

In addition to the changes the hint causes in the returned results, including the hint also changes some of the syntax rules for a query. The hint affects:

- Query syntax rules when a repeating property is a selected property

  "Repeating properties" on page 130, describes how the hint affects query syntax rules when a repeating property is selected.

- The use of repeating properties in expressions in WHERE clauses

  "Using repeating properties in qualifications" on page 156, describes how the hint affects the use of repeating properties in WHERE clause qualifications.

- The behavior of the asterisk as a selected value

  "The asterisk (*) as a selected value" on page 140, describes how the hint affects the returned values for an asterisk.

A.16 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.

Note: Not supported for PostgreSQL.

On SQL Server, using default result sets allows the RDBMS server to perform fewer logical read operations. "Comparison of logical reads with and without default result sets" on page 385 shows some test examples:

Table A-5: 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='0900014c 8000210b'	1	36	5
SELECT object_name FROM dm_document WHERE r_creator_name='user 2'	100	1600	700
SELECT object_name FROM dm_document WHERE r_creator_name='user 1'	2000	22500	500

However, the reduction in I/O may be offset by higher CPU costs. Additionally, because using default result sets requires Documentum CM 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, Documentum CM 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.

A.17 TRY_FTDQL_FIRST

The TRY_FTDQL_FIRST hint is useful if a query is timing out or exceeds a resource limit in the full-text engine. When it is included in a query, the query is first executed as an FTDQL query, and if a timeout or resource exceeded error occurs, the query is then retried as a standard query.

A.18 UNCOMMITTED_READ

Use the UNCOMMITTED_READ hint in read only queries, to ensure that the query returns quickly even if another session is holding locks on the tables queried by the read only query.

This hint is useful only on SQL Server.

A.19 DM_LEFT_OUTER_JOIN_FOR_ACL

When this DQL hint is added and if the user is not a super user, Documentum CM Server generates an SQL query that contains a LEFT OUTER JOIN on the dm_acl_s table. This operation improves the performance of DQL queries issued by a non-super user by optimizing the user permit calculation of resolved objects. This hint is effective when the user is a part of many groups. The following DQL statement is an example of using this hint:

```
SELECT r_object_id FROM dm_document WHERE object_name = 'test' ENABLE
(DM_LEFT_OUTER_JOIN_FOR_ACL)
```

The query returns r_object_id of all documents with object name ‘test’.

This hint overrides the computer variable DM_LEFT_OUTER_JOIN_FOR_ACL.

A.20 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)
```

Documentum CM 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.

A.21 Passthrough hints

Passthrough hints are hints that are passed to the RDBMS server. They are not handled by Documentum CM Server.

Note: Not supported for PostgreSQL.

SQL Server has 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 Oracle, include passthrough hints only at the end of the SELECT statement.

A.21.1 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, and SQL_SERVER. 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'),
SQL_SERVER('LOOP JOIN','FAST 1')
```

A.21.2 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 Documentum CM 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, Documentum CM Server runs with the last SQL trace option turned on. You can turn on full SQL tracing using an IDfSession.setServerTraceLevel race method or the SET_OPTIONS administrative function. The SET_OPTIONS administration method is described in “SET_OPTIONS” on page 340.

Appendix B. Database footprint reduction of dmr_content objects

From the 6.6 release, support for true NULL values was added to Documentum CM Server, for databases that support it. Users can create and alter their own types to take advantage of the space savings possible with this feature, but most internal Documentum CM Server types do not use it. The one exception is dmr_content.

In order to use true NULL storage with dmr_content, you must set the server.ini parameter, *spaceoptimize* to dmr_content in a new repository:

```
spaceoptimize=dmr_content
```

Currently, dmr_content is the only internal type that you can specify. For Oracle installations, all the properties of dmr_content will be set to SPACEOPTIMIZE. For SQL Server and PostgreSQL, only the character and string properties and the ID properties will be set to SPACEOPTIMIZE.

For single-valued properties, the total number of bytes saved per content object is 87 bytes for Oracle and 74 bytes for SQL Server.

For repeating-valued properties, the space saving is not so easy to predict. If all the repeating properties have one value, the default value, there is no space saving since a last value of default is always stored, not as a NULL, but as the default value. However, if there are N values for each repeating property, each with default value, then the space saving would be about (N-1)*(propertysize).

The details of the space saving are listed in the table, "Dmr_content object space savings" on page 389.

Table B-1: Dmr_content object space savings

Property	Type	Bytes Saved	Single/Repeating
rendition	integer	4	single
other_ticket	integer	4	single
resolution	integer	4	single
x_range	integer	4	single
y_range	integer	4	single
z_range	integer	4	single
encoding	char(10)	2 on SQL Server, 1 on others	single
loss	integer	4	single
transform_path	char(32)	2 on SQL Server, 1 on others	single

Property	Type	Bytes Saved	Single/Repeating
set_client	char(64)	2 on SQL Server, 1 on others	single
set_file	chare(255)	2 on SQL Server, 1 on others	single
set_time	time	8 on SQL Server, 9 on Oracle	single
is_offline	Boolean	2	single
i_contents	char(4000)	2 on SQL Server, 1 on others	single
is_archived	Boolean	2	single
index_format	ID	31 on SQL Server, 16 on Oracle	single
index_parent	ID	31 on SQL Server, 16 on Oracle	single
i_rendition	integer	4	repeating
i_px	integer	4	repeating
i_py	integer	4	repeating
i_encoding	char(10)	2 on SQL Server, 1 on others, for each entry	repeating
page_modifier	char(16)	2 on SQL Server, 1 on others, for each entry	repeating
r_content_hash	chare(256)	2 on SQL Server, 1 on others	single
i_parked_state	integer	4	single
other_file_size	double	4	single

Appendix C. IDQL and IAPI

This appendix describes the IDQL and IAPI utilities. The utility is useful if you want to test short scripts or perform short, small tasks in the repository. The IAPI utility allows you to execute Docbasic scripts.

The appendix includes the following topics:

- "Using IDQL" on page 391
- "Using IAPI" on page 395

C.1 Using IDQL

The IDQL utility is an interactive tool that lets you enter ad hoc DQL queries against a repository. IDQL is also a useful as a tool for testing and other tasks that support an application or installation because it allows you to run scripts and batch files.

IDQL is included and installed with Documentum CM Server. It is found in %DM_HOME%\bin ($DM_HOME/bin). Copy the utility's executable from the bin directory to a directory you can access or copy it to your local machine. Depending on your operating system, there may be multiple IDQL utilities. For example, on a 64-bit Windows installation, you will find both a 32-bit and 64-bit version of the utility, `idql32` and `idql64`.

If you copy it to your local machine, you must also copy the client library (DMCL) to your local machine if it is not already there.

C.1.1 Starting IDQL

You can start the utility from the operating system prompt or an icon. To start the utility from the operating system prompt, use the following syntax:

On Windows (use idql32 for the 32-bit version, idql64 for the 64-bit version):

```
idql32 [-n] [-wcolumnwidth]  [-Uusername|-ENV_CONNECT_USER_NAME] [-Pos_password|-
ENV_CONNECT_PASSWORD][-Ddomain]  repository|-ENV_CONNECT_DOCBASE_NAME [-Rinput_file] [-X]
[-E][-Ltrace_level][-zZ][-Winteger][-Ksecure_mode]
```

On Linux:

```
idql [-n] [-wcolumnwidth]
[-Uusername|-ENV_CONNECT_USER_NAME]
[-Pos_password|-ENV_CONNECT_PASSWORD][-Ddomain]
repository|-ENV_CONNECT_DOCBASE_NAME [-Rinput_file]
[-X][-E][-Ltrace_level][-zZ][-Winteger][-Ksecure_mode]
```

The order of the arguments on the command line is not important.

If you created an icon for the utility, double-click the icon and enter your name and password. To invoke the utility with additional command line arguments, use the icon's Properties to add the arguments to the command line before you double-click the icon.

"IDQL command line arguments" on page 392, describes the command line arguments.

Table C-1: IDQL command line arguments

Argument	Description
-n	Removes numbering and the prompt symbol from input files. This argument is used primarily for scripts.
-w*columnwidth*	Sets the screen width for output. The default width is 80 characters.
-U*username* or -ENV_CONNECT_USER_NAME	Identifies the user who is starting the session. You can include the user name on the command line with the -U flag, or you can use the -ENV_CONNECT_USER_NAME argument, which directs the system to look for the user name in the environment variable DM_USER_NAME.
-P*os_password* or -ENV_CONNECT_PASSWORD	Identifies the user's operating system password. You can include the password on the command line with the -P flag, or you can use the -ENV_CONNECT_PASSWORD argument, which directs the system to look for the password in the environment variable DM_PASSWORD. If the user does not specify a password, the utility prompts for the password. If the user enters the -P flag without the password, IDQL uses the default password, NULL. Passwords are case sensitive.
-D*domain*	Identifies the user's domain.
repository or -ENV_CONNECT_DOCBASE_ NAME	Identifies the repository. You can include the repository name on the command line, or you can use the -ENV_CONNECT_DOCBASE_NAME argument, which directs the system to look for the repository name in the environment variable DM_DOCBASE_NAME.
-Rinputfile	Specifies the name of an input file containing DQL queries. This is an optional argument.
-X	Allows prompt for domain.
-E	Turns on echo.

Argument	Description
-L*trace_level*	Turns tracing on for the session. The *OpenText Documentum Content Management - Server Administration and Configuration Guide (EDCCS250400-AGD)* contains the information about the trace level security values.
-zZ	Specifies Windows NT Unified Logon. This option overrides any user name and password specified on the command line.
-W*integer* or -w*integer*	Sets the maximum column width when displaying results. *integer* is interpreted as number of bytes. The default is 2000 bytes. Any value which exceeds the maximum row width is truncated when displayed.
-K*secure_mode*	Defines the type of connection you want to establish. Valid values are: • secure • native • try_secure_first • try_native_first The default is native.

After you start IDQL, the utility returns with its prompt. The IDQL utility has a numerical prompt that begins with 1 and increments each time you press the carriage return or Enter key to move to a new line on your display.

C.1.2 IDQL commands

IDQL recognizes the commands listed in “IDQL commands” on page 393.

Table C-2: IDQL commands

Command	Description
go	Executes a DQL statement and clears the query buffer.
clear	Clears the query buffer.
quit	Exits the utility.

Command	**Description**

describe	Provides a description of a specified object type or registered table. The syntax formats for this command are:
	`describe [type] ` *`type_name`*
	`describe table`*`table_name`*

@scriptfile	Executes the specified script.
shutdown	Issues a Shutdown method that shuts down the Documentum CM Server gracefully. You must be a superuser or system administrator to use this command. When shutdown is complete, the utility exits.
trace	Turns on tracing for the current session. By default, tracing is set to level 5.
-W*integer* or -w*integer*	Sets the maximum column width when displaying results. integer is interpreted in bytes. The default is 2000 bytes. Setting this affects the results display for all statements issued after this command.

Each command must be entered on a separate line. A command cannot appear on the same line as a query, nor can two or more commands appear on one line.

C.1.3 Entering queries

Enter a query by typing it at the IDQL prompt. The following IDQL session shows two queries typed at the prompt:

```
1> BEGIN TRAN
2> go
3> SELECT *
4> FROM Engineering
5> WHERE Engineer LIKE 'Smith, %'
6> go
```

The query processor interprets semicolons (;) as statement terminators and ignores everything that follows the semicolon on the line.

When you enter a query, IDQL places the query in a buffer. You can then execute the query or clear the buffer.

The go command executes a query. Each go command can execute only one query. You cannot execute multiple queries with one go command.

There are two ways to execute queries in a batch using IDQL:

- Use the input file option in the IDQL command line.

- Use the *@scriptfile* command.

The input file is a file that contains queries. You must place a `go` command after each query in the file, and you must terminate the file with a `quit` command.

The results of each query are returned to standard output by default.

The *@scriptfile*command executes a specified script, which can include DQL queries. Like an input file, queries in the script must be separated by `go` commands. You are not required to use a `quit` command to terminate a script executed with *@scriptfile*.

C.1.4 Clearing the buffer

The following excerpt enters a query and then clears it from the buffer:

```
1> SELECT *
2> FROM Engineering
3> WHERE Engineer LIKE 'Smith, %'
4> clear
```

C.1.5 Entering comments

To enter a comment in an IDQL session, start the comment line with \\ or -- .

C.1.6 Stopping IDQL

To close an IDQL session, enter the `quit` command at the IDQL prompt.

C.2 Using IAPI

The IAPI utility is a tool that you can use to execute Docbasic scripts. This utility is included and installed with Documentum CM Server. It is found in %DM_HOME%\bin ($DM_HOME/bin). On a 64-bit Windows installation, you will find a 64-bit version of the utility, `iapi64`.

C.2.1 Starting IAPI

You can start the utility from the operating system prompt or an icon. To start the utility from the operating system prompt, use the following syntax:

On Windows (use iapi64 for the 64-bit version):

```
iapi [-Uusername|-ENV_CONNECT_USER_NAME]
[-Pos_password|-ENV_CONNECT_PASSWORD] [-Llogfile] [-X][-E]
[-V-][-Q] repository|-ENV_CONNECT_DOCBASE_NAME [-Rinputfile]
[-Ftrace_level][-Sinit_level][-Iescape_character][-zZ]
[-Winteger][-Ksecure_mode]
```

On Linux:

```
iapi [-Uusername|-ENV_CONNECT_USER_NAME]
[-Pos_password|-ENV_CONNECT_PASSWORD] [-Llogfile] [-X][-E]
[-V-][-Q] repository|-ENV_CONNECT_DOCBASE_NAME [-Rinputfile]
```

```
[-Ftrace_level][-Sinit_level][-Iescape_character][-zZ]
[-Winteger][-Ksecure_mode]
```

The order of the arguments on the command line is not important.

If you created an icon for the utility, double-click the icon and enter your name and password. To invoke the utility with additional command line arguments, use the icon's Properties to add the arguments to the command line before you double-click the icon.

"IAPI command line arguments" on page 396, describes the command line arguments.

Table C-3: IAPI command line arguments

Argument	Description
-U*username* or -ENV_CONNECT_USER_NAME	Identifies the user who is starting the session. You can include the user name on the command line with the -U flag or you can use the -ENV_CONNECT_USER_NAME argument, which directs the system to look for the user name in the environment variable DM_USER_NAME.
-P*os_password* or -ENV_CONNECT_PASSWORD	Identifies the user's operating system password. You can include the password on the command line with the -P flag, or you can use the -ENV_CONNECT_PASSWORD argument, which directs the system to look for the password in the environment variable DM_PASSWORD. If the user does not specify a password, the utility prompts for the password.
-L*logfile*	Directs the server to start a log file for the session. You must include the file name with the argument. You can specify a full path.
-X	Allows prompt for domain.
-E	Turns on echo.
-V-	Turns verbose mode off. The utility runs in verbose mode by default. If you turn verbose mode off, the system does not display the IAPI prompt, error messages, or any responses to your commands (return values or status messages). You can also change modes from the IAPI prompt (refer to"IAPI commands" on page 398).

Argument	Description
-Q	Directs the utility to provide a quick connection test. The utility attempts to connect and exits with a status of zero if successful or non-zero if not. It provides error messages if the attempt is not successful.
repository or -ENV_CONNECT_DOCBASE_ NAME	Identifies the repository. You can include the repository name on the command line, or you can use the -ENV_CONNECT_DOCBASE_NAME argument, which directs the system to look for the repository name in the environment variable DM_DOCBASE_NAME.
-R*inputfile*	Identifies an input file containing API methods.
-F*trace_level*	Turns tracing on for the session. The *OpenText Documentum Content Management - Server Administration and Configuration Guide (EDCCS250400-AGD)* contains the information about the trace level security values.
-S*init_level*	Defines the connection level established when you start the API session. Valid init levels and their meanings are: *api*, which opens an API session. User must issue an explicit Connect to start a session. *connect*, meaning undetermined *standard*, which opens a session
-I*escape_character*	Defines an escape character for certain symbols such as a forward slash (/).
-zZ	Specifies Windows NT Unified Logon. This option overrides any user name and password specified on the command line.
-W*integer* or -w*integer*	Sets the maximum column width when displaying results returned from a query entered with a ? command. *integer* is interpreted as number of bytes. The default is 2000 bytes. Any value which exceeds the maximum row width is truncated when displayed.

Argument	Description
-K*secure_mode*	Defines the type of connection you want to establish. Valid values are: • secure • native • try_secure_first • try_native_first The default is try_native_first.

When you issue the `iapi` command, the system returns a status message that tells you it is connecting you to the specified database. This is followed by a message giving you the session ID for your session and then the utility's prompt (API>).

C.2.2 IAPI commands

The IAPI commands affect how the IAPI utility functions. These commands do not affect the repository. The utility accepts the commands listed in "IAPI commands" on page 398.

Table C-4: IAPI commands

Command	Description
$*logfile*	Outputs to the specified file
@*scriptfile*	Reads input from the specified file
%*scriptfile*	Outputs a script
-V[+|-]	Turns verbose mode on (+) or off (-). By default, verbose mode is on. If you turn verbose mode off, the system does not display the IAPI prompt, error messages, or any responses to your commands (return values or status messages).
-W*integer* or -w*integer*	Sets the maximum column width when displaying results returned from a query entered with a ? command. integer is interpreted in bytes. The default is 2000 bytes. Setting this affects the results display for all methods issued after this command.
?	Allows you to enter a query or collection ID and formats the output as a table.

The ? command allows you to enter a DQL SELECT statement in IAPI. When used with a SELECT statement, the command is the equivalent of a Readquery method. The syntax is:

```
API>?,c,select_query_string
```

For example:

```
API>?,c,select r_object_id, object_name from dm_document
```

The results are formatted in a table-like block. For example:

```
r_object_idobject_name
----------------------
090015c38000013Bfoo.txt
090015c3800002e4spam.html
090015c3800001c2fah.txt
```

You can also use the ? command to retrieve a collection's content in one block. For example, suppose you issue a readquery method, which returns a collection:

```
API>readquery,c,select r_object_id, object_name from dm_document
...
API>q0
```

Instead of using a Next method to retrieve each row of the collection, you can use ? to retrieve them all at once:

```
API>?,c,q0
...
r_object_idobject_name
----------------------
090015c38000013Bfoo.txt
090015c3800002e4spam.html
090015c3800001c2fah.txt
```

C.2.3 Entering method calls

Call methods directly from the IAPI prompt by typing the method name and its arguments as a continuous string, with commas separating the parts. For example, the following command creates a new folder:

```
API> create,s0,dm_folder
```

If any parameter value contains a comma, you must enclose that parameter in single quotes. For example, the following command creates a new document and assigns it a content file:

```
API> create,s0,dm_document
API> setfile,s0,last,'/home/janed/temp.doc,1',text
```

The name of the content file is in single quotes because it contains a comma.

When the method completes, the utility displays the return value. For example, the Create method returns the object ID of the newly created object. So if you issue the Create command shown in the previous example, the utility displays the object ID of the new folder:

```
API> create,s0,dm_folder
. . .
0b6385F20000561B
```

When you execute a method that assigns a value to an attribute, you do not specify the value as part of the command syntax. Instead, IAPI prompts you for that value. For example, the following exchange sets the name of the folder object created in the previous example:

```
API> set,s0,last,object_name
SET> cake_recipes
. . .
OK
```

The keyword `last` refers to the last created, fetched, checked in/out, or retrieved object ID (in this case, the folder ID).

C.2.4 Entering comments

Enter a comment in an IAPI session by preceding it with the pound sign (#). This is useful when you want to use scripts with IAPI.

C.2.5 Shutting down and restarting Documentum CM Server

The following command shuts down the Documentum CM Server:

```
API> apply,c,NULL,SHUTDOWN_WORKFLOW_AGENT,<TIMEOUT>,<|>,<timeout_value>
```

The *`TIME_OUT`* parameter is optional. If the parameter is not provided, then the Documentum CM Server will take the value specified in server_config > system_shutdown_timeout as the timeout value for shutdown.

If the *`TIME_OUT`* parameter is specified, the Documentum CM Server is shutdown depending on the values.

- > 0: The workflow agent will wait for the specified value to do a graceful shutdown and when the timeout value is reached, it will shutdown ungracefully.
- = 0: The workflow agent will shutdown ungracefully.
- < 0: The workflow agent will do a graceful shutdown.

The following command restarts the Documentum CM Server:

```
API> apply,c,NULL,START_WORKFLOW_AGENT
```

C.2.6 Quitting an IAPI session

To exit from IAPI, enter quit at the IAPI prompt:

```
IAPI> quit
```

Appendix D. Implementing Java evaluation of Docbasic expressions

This appendix describes how to implement Java evaluation of Docbasic expressions defined for check constraints or value assistance in the data dictionary.

D.1 Docbasic expression handling by Documentum CM Server

This section describes how Documentum CM Server describes how Docbasic expressions defined for check constraints or value assistance in the data dictionary are stored in the repository

Figure D-1, illustrates the architecture.

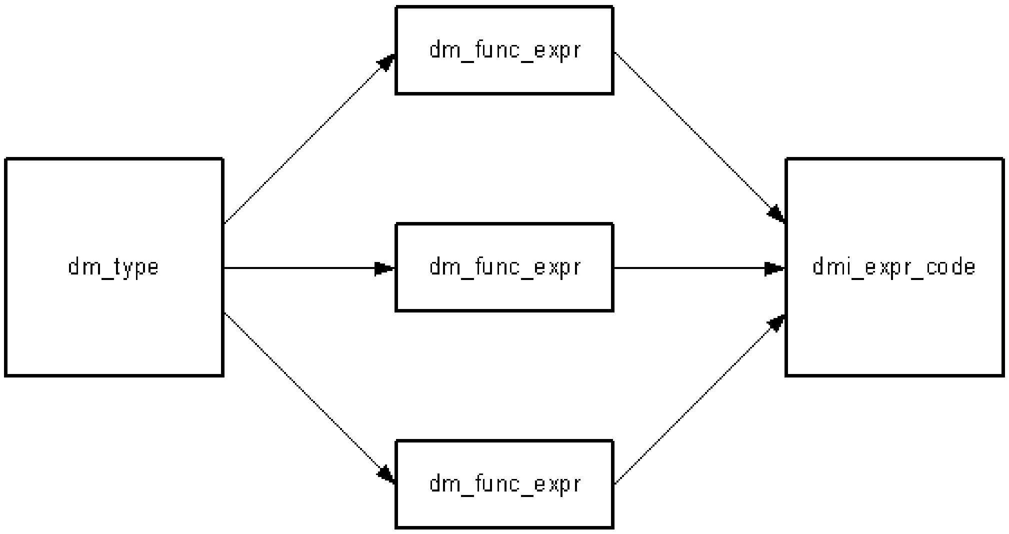

Figure D-1: Repository storage of Docbasic expressions for object types

For each object type that has Docbasic expressions defined as check constraints or in conditional value assistance for one or more properties, Documentum CM Server creates one expr code object and a number of func expr objects. Each func expr objects records one Docbasic expression. The expr code object contains the compiled code and pcode that implements the expressions recorded in the func expr objects. The routine_id property in the func expr objects points to the expr code object.

D.2 How Foundation Java API 6.0 and later handles the expressions

Foundation Java API 6.0 and later generates the Java code equivalent to the Docbasic expressions defined for check constraints or value assistance at runtime. The result is cached in memory for the life of the Foundation Java API process.

D.3 Migrating the expressions for pre-6.0 clients

Prior to 6.0, Foundation Java API required that the expressions be manually migrated to Java if you wished to have the expressions evaluated as Java code. If you are running pre-6.0 clients, use the information in this section to migrate the Docbasic expressions to Java code. This section describes how that implementation was stored in the repository and the methods used to migrate the expressions.

Manually migrating a Docbasic expression for Foundation Java API 6.0 and later clients is not useful. Foundation Java API 6.0 and later automatically generates Java code at runtime for Docbasic expressions and will ignore the results of manual migration.

D.3.1 Repository storage of migrated expressions

Migrating an expression manually creates additional repository objects that are associated with the objects created by Documentum CM Server for the expression.

If you migrate an expression to Java, the migration method compiles the expression into Java code. If the expression is the first expression compiled into Java for the object type, the migration method also creates a dmc_jar object and a dmc_validation_module object. The compiled code is stored with the dmc_jar object. The jar object is related to the validation module object through a relationship named dmc_module_to_jar.

The validation module object points back to the associated expr code object through a property and is related to the func expr objects using dmc_validation_relation objects. Validation relation objects are a subtype of dm_relation. The name of the relationship represented by validation relation objects is dmc_expr_to_module. Both the validation module and validation relation objects have properties used to manage the Java evaluation. Figure D-2, illustrates this architecture.

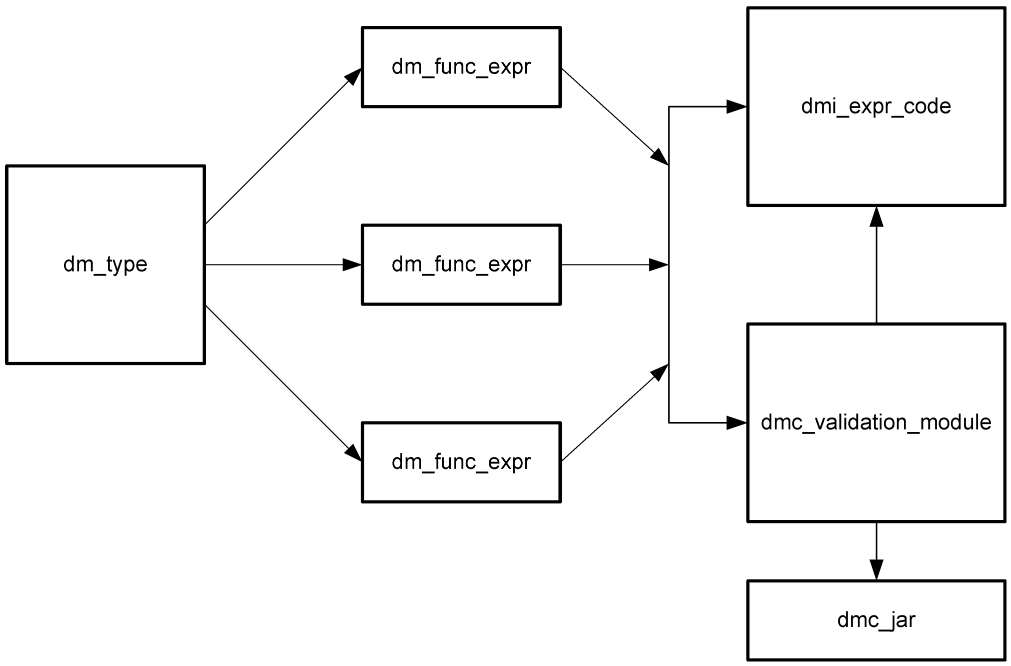

Figure D-2: Repository storage of manually migrated Docbasic expressions

If additional Docbasic expressions are added later for an object type, the migration method updates the validation module and jar objects.

Migrating expressions to Java is currently a manual operation. OpenText Documentum CM provides two migration methods, described in "Migrating Docbasic expressions to Java" on page 405. You can migrate expressions already defined in the data dictionary. You can migrate new expressions after the expressions are defined using an ALTER TYPE or CREATE TYPE statement.

D.3.2 Migrating Docbasic expressions to Java

There are two methods that create compiled Java code for Docbasic expressions stored with expr code objects. These methods are:

- dmc_MigrateDbExprsToJava

  Use this method if you want to migrate expressions defined for multiple object types.

- dmc_MigrateDbExprsToJavaForType

  Use this method if you want to migrate expressions added to a single object type (or the type's properties) using an ALTER TYPE or if you want to migrate expressions defined while creating a new type using CREATE TYPE.

Arguments for both methods let you select which Docbasic expressions to migrate to Java code. You can choose to compile only new Docbasic expressions, or those that failed compilation previously, or all expressions within the scope of the method.

The methods are executed using either a Foundation Java API script or using a DO_METHOD administration method. The DO_METHOD syntax is:

```
dmAPIGet("apply,session,NULL,DO_METHOD,METHOD,S,migration_method_name,SAVE_RESULTS,B,T,AR
GUMENTS,S,argument_list
```

migration_method_name is either dmc_MigrateDbExprsToJava or dmc_MigrateDbExprsToJavaForType.

Use a space to separate arguments in *argument_list*."Arguments for the Docbasic expression migration methods" on page 406, lists the valid arguments for *argument_list*. .

Table D-1: Arguments for the Docbasic expression migration methods

Argument	Description
-docbase *docbase_name*	Name of the repository that contains the expressions you want to migrate. This must a 5.3 (or later) repository.
-user *user_name*	Name of a user who has at least Sysadmin privileges in the specified repository.
-ticket *login_ticket*	String containing a login ticket. **Note:** The *OpenText Documentum Content Management - Server Administration and Configuration Guide (EDCCS250400-AGD)* contains more information about login tickets and generating those tickets.

Argument	Description
-select *selection_choice*	Identifies which expressions you wish to migrate. *selection_choice* is one of: • *new_only*, includes only new expressions, that is, those for which migration has not been previously attempted • *new_and_failed*, includes all new expressions and those for which migration previously failed • *new_and_disabled*, includes all new expressions and those that do not have a corresponding enabled Java version • *all*, includes all expressions Any existing Java code is replaced if currently migrated expressions are recompiled. For dmc_MigrateDbExprsToJava, expressions for all object types are considered for inclusion. For dmc_MigrateDbExprsToJavaForType, only expressions defined for the object type identified in the *type* argument are considered for inclusion.
-maxerrs *value*	Maximum number of errors that can occur before code generation stops. By default, there is no maximum. Setting this argument is recommended only if there are large numbers of expressions to compile.
-listExprsOnly *value*	Whether to actually perform the migration or only create a file that records the expressions selected for migration. Setting value to true directs the method to only create the file. Setting the value to false directs the method to migrate the expressions. If you set this argument to true, you must include the SAVE_RESULTS argument in the DO_M,ETHOD command line.
-type *type_name*	Name of the object type. Include this argument only if you are executing the dmc_MigrateDbExprsToJavaForType method. Use the type's internal name; for example: dm_document.

It is strongly recommended that you include the SAVE_RESULTS argument on the DO_METHOD command line regardless of the -listExprsOnly setting. The content of the generated results document varies depending on the circumstances of the method's execution:

- If -listExprsOnly is set to true, the results document contains a list of all expressions selected for migration.
- If -listExprsOnly is false and non-fatal errors occur during migration, the results document contains the errors.
- If -listExprsOnly is false and fatal errors occur during migration, the results document contains the exception stack trace embedded in the HTML error message.
- If -listExprsOnly is false and no errors occur during migration, the results document contains only the text “There were no errors”.

Both methods return 0 unless a fatal error occurred during execution. If a fatal error occurs, the methods return 1. Fatal errors are reported using the standard method server mechanism.

D.4 Disabling or re-enabling Java evaluation

It is possible to disable Java evaluation of a particular Docbasic expression. If you disable Java evaluation of an expression, the Docbasic library is used to evaluate the expression, rather than the compiled Java code.

To disable Java evaluation, execute the dmc_SetJavaExprEnabled method. This method has an argument that turns evaluation of a specified expression on or off. The method is executed by a Foundation Java API script or a DO_METHOD administration method. Here is the syntax for the DO_METHOD call:

```
apply,session,NULL,DO_METHOD,METHOD,S,dmc_SetJavaExprEnabled,SAVE_RESULTS,B,T,ARGUMENTS,S
,argument_list
```

“dmc_SetJavaExprEnabled arguments” on page 408, lists the arguments for dmc_SetJavaExprEnabled. All arguments for this method are required. Use a space to separate arguments in *argument_list*.

Table D-2: dmc_SetJavaExprEnabled arguments

Argument	Description
-docbase *repository_name*	Name of the repository that contains the Java expression implementation you want to disable or re-enable.
-user *user_name*	Name of the user performing the operation. Use the user's login name. The user must have at least Sysadmin privileges in the repository.
-ticket *login_ticket*	A string containing a login ticket generated for the specified user.

Argument	Description
-func_expr_id *func_expr_obj_id*	Object ID of the dm_func_expr object representing the Docbasic expression.
-enable *value*	Set this to true to enable the Java evaluation of the Docbasic expression identified in the -func_expr_id argument. Or, set it to false, to disable Java evaluation of that expression.

D.5 Docbasic expression components support

This section lists and briefly describes the Docbasic components, such as operators, functions, and constants, that are supported for Java migration.

D.5.1 Operators

"Docbasic operators supported by Java evaluation" on page 409, lists the Docbasic operators for which Java evaluation is supported.

Table D-3: Docbasic operators supported by Java evaluation

Operator	Operation performed
&	String concatenation
*	Multiplication
+	Addition or concatenation
–	Subtraction
/	Floating point division
<	Comparison (less than)
<=	Comparison (less than or equal to)
<>	Comparison (not equal to)
=	Comparison (equal to)
>	Comparison (greater than)
>=	Comparison (greater than or equal to)
\	Integer division
^	Exponentiation
And	Logical or binary conjunction
Eqv	Logical or binary equivalence (not Xor)
Imp	Logical or binary implication

Operator	Operation performed
Is	Compares two operands and returns true if they refer to the same object; otherwise, returns false. Because objects are not supported, using this operator always results in a type mismatch error at runtime.
Like	Compares two strings and returns true if the expression matches the given pattern; otherwise, returns false.
Mod	Returns the remainder of the *expression_1*/ *expression_2* as whole number
Not	Returns either a logical or binary negation of an expression
Or	Logical or binary disjunction on two expressions
Xor	Exclusive Or operation

D.5.2 Supported functions for Java evaluation

"Docbasic functions supported for Java evaluation" on page 410, lists the Docbasic functions for which Java evaluation is supported.

Table D-4: Docbasic functions supported for Java evaluation

Function	Description
abs(expr)	Returns the absolute value of expr
Asc(text$)	Returns the integer containing the numeric code for the first character of the text$ (0–255)
AscB or AscW	Returns the byte or wide-character equivalents of Asc
Atn(number)	Returns the inverse tangent of number
CBool(expr)	Converts the expr to a Boolean
CDate(expr) or CVDate(expr)	Converts the expr to a Date **Note:** If expr is a string, it is expected that the date is in canonical form unless a specific date format string was passed to the validation method.
CDbl(expr)	Converts expr to a Double

Function	Description
Choose(index, expr1, expr2,...exprN)	Returns the expression whose position in the list of expr arguments matches the value specified index. If none is found, returns null.
Chr(code) or Chr$(code)	Returns the character whose value is code The Java implementation returns a String variant in both functions.
CInt(expr)	Converts expr to a Docbasic integer. The Java implementation converts expr to a Java short.
CLng(expr)	Converts expr to a Docbasic long The Java implementation converts expr to a Java int.
Cos(angle)	Returns the cosine of angle where angle is assumed to be expressed in radians
CSng(expr)	Converts expr to a Docbasic Single The Java implementation converts expr to a Java float.
CStr(expr)	Converts expr to a String
CVar(expr)	Converts expr to a Variant
Date	Returns the current date as a Date Variant
Date$	Returns the current date as a String The Java implementation will always be in canonical form or in the format passed to the validation method.
DateAdd(interval$, increment&, date)	Increments the date field identified in interval$ of the given date by the amount specified in interval&
DateDiff(interval$, date1, date2)	Returns a Date variant representing the number of given time intervals between date1 and date2
DatePart(interval$, date)	Returns an Integer representing a specific part of a date/time expression
DateSerial(year, nonth, day)	Returns a Date variant representing the specified date
DateValue(dateString$)	Returns a Date variant representing the date specified by dateString$
Day(date)	Returns the day of the month represented by date

Function	Description
Exp(val)	Returns the value of *e* raised to the power of val
Fix(number)	Returns the integer part of number
Format or Format$(expr, userFormat)	Returns a string formatted to user specification (supports both named and unnamed formats)
Hex[$](number)	Returns a String containing the hexadecimal equivalent of number
Hour(time)	Returns the hour of the day encoded in time
If(condition, TrueExpression, FalseExpression)	Returns TrueExpression if condition is true; otherwise, returns FalseExpression
InStr([start,] search, find [,compare])	Returns the first character position of string “find” within string “search”
InStrB([start,] search, find [,compare])	Returns the first character position of string “find” within string “search”
Int(number)	Returns the integer part of number
IsDate(expr)	Returns true if expr can be legally converted to a date; otherwise, returns false
IsNull(expr)	Returns true if expr is a Variant variable that contains no valid data; otherwise, returns false
IsNumeric(expr)	Returns true if expr can be converted to a number; otherwise, returns false
Item$(text$, first, last [,delimiters$])	Returns all items between first and last within the specified formatted text list
ItemCount(text$ [,delimiters$])	Returns an integer containing the number of items in the specified delimited list
LBound(arrayVar [,dimension])	Returns an integer containing the lower bound of the specified dimension of the specified array variable
LCase[$](text)	Returns the lowercase equivalent of text
Left[$](text, nChars) or LeftB[$](text, nChars)	Returns the leftmost nChars characters (Left and Left$) or bytes (LeftB and LeftB$) from text **Note:** Use LeftB with caution. Multibyte characters can be split in the middle, resulting in the construction of illegal strings.
Len(expr or lenB(expr)	Returns the number of characters in expr or the number of bytes required to store expr

Function	Description
Line$(text$, first [,last])	Returns a string containing a single line or a group of lines between first and last
LineCount(text$)	Returns an integer representing the number of lines in text$
Log(number)	Returns a double representing the natural logarithm of number
LTrim[$](text)	Returns text with the leading spaces removed
Mid[$](text, start [,length]) or MidB[$](text, start [,length])	Returns a substring of the specified text, beginning with start for length number of characters or bytes **Note:** Use MidB with caution. Multibyte characters can be split in the middle, resulting in the construction of illegal strings.
Minute(time)	Returns the minute of the day encoded in the time argument
Month(date)	Returns the month of the date
Now	Returns a date variant representing the current date and time
Oct(number) or Oct$(number)	Returns a string containing the octal equivalent of the specified number
Random(min,max)	Returns a long value greater than or equal to min and less than or equal to max
Right[$](text,nChars) or RightB[$](text,nChars)	Returns the rightmost nChars characters or bytes from the text. RightB and RightB$ are used to return byte data from strings containing byte data. **Note:** Use RightB with caution. Multibyte characters can be split in the middle, resulting in the construction of illegal strings.
Rnd[(number)]	Returns a random Single number between 0 and 1
RTrim[$](text)	Returns text with the trailing spaces removed
Second(time)	Returns the second of the day encoded in the time argument
Sgn(number)	Returns an integer indicating whether a number is less than, greater than, or equal to 0

Function	Description
Sin(angle)	Returns a double value specifying the sine of angle
Space[$](NumSpaces)	Returns a string containing the specified number of spaces
Sqr(number)	Returns a double representing the square root of number
Str[$](number)	Returns a string representation of number
StrComp(string1, string2 [,compare])	Returns an integer indicating the result of comparing string1 and string2
String[$](number, [CharCode \| text$])	Returns a string of length number consisting of a repetition of the specified filler character
Swith(condition1, expr1 [,condition2,expr2... [condition7,expr7]])	Returns the expression corresponding to the first true condition.
Tan(angle)	Returns a double representing the tangent of angle
Time/Time$	Returns the system time as a string or as a Date variant In the Java implementation, the time is in canonical form.
Timer	Returns a Single representing the number of seconds that have elapsed since midnight
TimeSerial(hour, minute, second)	Returns a Date variant representing the given time with a date of zero
TimeValue(time_string$)	Returns a date variant representing the time contained in time_string$ The date must be in canonical form or in a format that conforms to the format string passed to the validation method.
Trim[$](text)	Returns a copy of text with leading and trailing spaces removed
TypeName(varname)	Returns the type name of the specified variable
UBound(arrayVar [,dimension])	Returns an integer containing the upper bound of the specified dimension of the specified array variable
UCase[$](text)	Returns the uppercase equivalent of the specified string
Val(numberString)	Converts a given string expression to a number

Function	Description
Vartype(variable)	Returns an integer representing the type of data in variable
Weekday(date)	Returns an integer value representing the day of the week given by date Sunday is 1, Monday is 2, and so forth.
Word$(text$, first [,last])	Returns a string containing a single word or sequence of words between first and last
WordCount(text$)	Returns an integer representing the number of words in text$
Year(date)	Returns the year of the date encoded in the date argument The value returned is between 100 and 9999, inclusive.

D.5.3 Unsupported functions for Java evaluation

The functions listed in "Docbasic functions not supported for Java evaluation" on page 415 are not supported for Java evaluation. For detailed information about these functions, refer to the Docbasic documentation.

Table D-5: Docbasic functions not supported for Java evaluation

Names of unsupported functions			
CCur(expr)	EOF	GetSession	Pmt(Rate, NPer, Pv, Fv, Due)
Clipboard$	Erl	GetSetting([appname], section, key[, default])	PPmt(rate, Per, Nper, Pv, Fv, Due)
Command or Command$	Err	Input, Input$, InputB, and InputB$ (...)	PrinterGetOrientation
CreateObject(ClassID)	Error and Error$ (errNum)	InputBox and InputBox$	PrintFile(filename$)
CurDir and CurDir$ (drive)	External(pathOrObjID)	IPmt(Rate, Per, Nper, Pv, Fv, Due)	Pv(reate, NPer, Pmt, Fv, Due)
CVErr	FileAttr(fileNum, attr)	IRR(ValueArray(), Guess)	Rate(NPer, Pmt, Pv, Fv, Due, Guess)
DDB(Cosot, Salvage, Life, Period)	FileDateTime(filename)	IsError(expr)	ReadIni$(section$, item$[, filename$])
DDEInitiate(application$, topic$)	FileExists(filename)	IsMissing(variable)	SaveFilename$[([title$[, extensions$]])]

Names of unsupported functions			
DDERequest[$](channel, DataItem$)	FileLen(filename)	Loc(filenumber)	Seek(filenumber)
Dir[$][(filespec$[,properties])]	FileParse(filename, op)	Lof(filenumber)	SetAppInfo(section$, entry$, string$, filename$)
DiskFree&([drive$])	FileType(filename)	MacID(text$)	Shell(command$[, WindowStyle])
dmAPIExec	FreeFile	MIRR(valArray, rate, reinvestRate)	ShellSync(command)
dmAPIGet	Fv(Rate, Nper, Pmt, Pv, Due)	MsgBox(msg)	Spc(numspaces)
dmAPISet	GetAppInfo(section$, entry$, filename$)	NPer(rate, Pmt, Pv, Fv, Due)	SYD(cost, salvage, life, period)
dmExit	GetAttr(filename$[,class$])	Npv(rate, valArray)	Tab(column)
Environ or Environ$ (varName | varNum)	GetOption(name$ | id)	OpenFilename$([[([title$[, extensions $]])]	TypeOf<objectVar> Is ObjectType

D.5.4 Supported constants

"Docbasic constants supported for Java evaluation" on page 416, lists the Docbasic constants that are supported for Java evaluation.

Table D-6: Docbasic constants supported for Java evaluation

Constant	Description
ebBoolean	Integer representing Boolean datatype
ebCurrency	Integer representing Currency type
ebDataObject	Integer representing a data object
ebDate	Integer representing a Date type
ebEmpty	Integer representing an Empty variant
ebError	Integer representing an Error variant
ebInteger	Integer representing the Integer type
ebLong	Integer representing the Long type
ebNull	Integer representing Null variant type
ebObject	Integer representing the OLE automation object type
ebSingle	Integer representing the Single type

Constant	Description
ebString	Integer representing the String type
ebVariant	Integer representing the Variant type
Empty	Constant representing an initialized variant
False	Boolean constant
Nothing	Null Object reference
Null	Explicit Docbasic value distinguishable from Empty or Nothing. (Refer to the Docbasic documentation for complete information.)
Pi	Value of Pi
True	Boolean constant

D.5.5 Unsupported constants

"Constants not supported for Java evaluation" on page 417, lists the constants that are not supported for Java evaluation.

Table D-7: Constants not supported for Java evaluation

Constant names		
ebCancel	ebNone	ebSystem
ebCritical	ebNormal	ebSystemModal
ebHidden	ebReadOnly	ebVolume

D.5.6 Implicit objects

The implicit objects provided by Docbasic, such as the implicit object named Basic, are not supported for use in expressions that are compiled for Java evaluation.

Appendix E. 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 "DQL statements" on page 43.

E.1 DQL statements

This section contains the formal syntax for each of the DQL statements.

E.1.1 Abort

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

E.1.2 Alter Aspect

```
ALTER ASPECT aspect_name ADD
[(property_def {,property_def})]
[[NO]OPTIMIZEFETCH]
```

```
ALTER ASPECT aspect_name ADD_FTINDEX ALL
```

```
ALTER ASPECT aspect_name DROP_FTINDEX ALL
```

```
ALTER ASPECT aspect_name ADD_FTINDEX property_list
```

```
ALTER ASPECT aspect_name DROP_FTINDEX property_list
```

E.1.3 Alter Group

```
ALTER GROUP group_name ADD members
```

```
ALTER GROUP group_name DROP members
```

```
ALTER GROUP group_name SET ADDRESS email_address
```

E.1.4 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 (property_modifier_clause)[PUBLISH]
```

```
ALTER TYPE type_name
ADD property_def {,property_def}[PUBLISH]
```

```
ALTER TYPE type_name
DROP property_name {,property_name}[PUBLISH]
```

```
ALTER TYPE type_name ALLOW ASPECTS
```

```
ALTER TYPE type_name
ADD|SET|REMOVE DEFAULT ASPECTS aspect_list
```

```
ALTER TYPE type_name ENABLE PARTITION
```

```
ALTER TYPE type_name SHAREABLE [PUBLISH]
```

E.1.5 Begin Tran

```
BEGIN TRAN[SACTION]
```

E.1.6 Change...Object

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

E.1.7 Commit

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

E.1.8 Create Group

```
CREATE GROUP group_name [WITH] [ADDRESS mail_address]
[MEMBERS member_list]
```

E.1.9 Create...Object

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

E.1.10 Create Type

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

To create a partitionable type:

```
CREATE PARTITIONABLE TYPE type_name
[(property_def {,property_def})]
[WITH] SUPERTYPE NULL
[type_modifier_list] [PUBLISH]
```

To create a shareable type:

```
CREATE SHAREABLE TYPE type_name
[(property_def {,property_def})]
[WITH] SUPERTYPE parent_type [PUBLISH]
```

To create a lightweight type:

```
CREATE LIGHTWEIGHT TYPE type_name
[(property_def {,property_def})]
SHARES shareable_type
[AUTO MATERIALIZATION |
MATERIALIZATION ON REQUEST |
DISALLOW MATERIALIZATION]
[FULLTEXT SUPPORT [
 NONE |
 LITE ADD ALL
 LITE ADD property_list |
 BASE ADD ALL |
 BASE ADD property_list
]
[PUBLISH]
```

E.1.11 Delete

```
DELETE FROM table_name WHERE qualification
```

E.1.12 Delete...Object

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

E.1.13 Drop Group

```
DROP GROUP group_name
```

E.1.14 Drop Type

```
DROP TYPE type_name
```

E.1.15 Execute

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

E.1.16 Grant

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

E.1.17 Insert

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

E.1.18 Register

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

E.1.19 Revoke

```
REVOKE privilege {,privilege} FROM users
```

E.1.20 Select

The syntax for a standard query is:

```
SELECT [FOR base_permit_level][ALL|DISTINCT] value [AS name] {,value [AS name]}
FROM [PUBLIC] source_list[WITHIN PARTITION (partition_id{,partition_id})
| IN DOCUMENT clause
| IN ASSEMBLYclause]
[SEARCH [FIRST|LAST]fulltext_search_condition[IN FTINDEX index_name{,index_name}]
[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 property {,property} [ASC|DESC]]
```

and the IN ASSEMBLY *clause* is:

```
IN ASSEMBLY object_id [VERSION version_label]
[NODE component_id][DESCEND]
```

The syntax for an FTDQL SELECT statement is:

```
SELECT [FOR base_permit_level][ALL|DISTINCT] value [AS name] {,value [AS name]}
FROM [PUBLIC] source_list[SEARCH fulltext_search_condition[IN FTINDEX
index_name{,index_name}]
[WHERE qualification]
[ORDER BY SCORE]
[ENABLE (hint_list)]
```

“Summary of FTDQL query rules” on page 423, lists the constraints imposed on the clauses in an FTDQL query.

Table E-1: Summary of FTDQL query rules

Applicable to	Rule
FTDQL queries in general	An FTDQL query must contain one of the following: • SEARCH clause • One of: SCORE, SUMMARY, or TEXT as a selected value • ENABLE(FTDQL) The query may not contain: • An IN DOCUMENT or IN ASSEMBLY clause • The FIRST or LAST keyword in a SEARCH clause • A UNION, GROUP BY, or HAVING clause
selected values list	The selected values list must satisfy the following rules: • Only the following keywords are acceptable: CONTENTID, ISCURRENT, OBJTYPE, SCORE, SUMMARY, SYSOBJ_ID, TEXT, and THUMBNAIL_URL • Literals or expressions of any type may not be included. • An asterisk (*) is not allowed.
AS clause	The AS clause is acceptable with no restrictions.
FROM clause	The following rules apply to the FROM clause: • You may include only one object type reference, and that reference must be to dm_sysobject or a SysObject subtype. • You may not include references to a registered table.
SEARCH clause	The SEARCH clause may not reference the FIRST or LAST keyword. There are no restrictions on the full-text search condition. However, note that the SEARCH DOCUMENT CONTAINS syntax is the preferred way to specify a SEARCH clause.

Applicable to	Rule
WHERE clause	A WHERE clause in an FTDQL query is subject to the following rules: • Any repeating properties referenced in the clause must be of type string or ID. • Only the DQL UPPER and LOWER functions are allowed. The functions SUBSTR, MFILE_URL, and all aggregate functions are not acceptable. • The following predicates are not allowed: – BETWEEN – NOT LIKE – NOT FOLDER – NOT NULL – [NOT] CABINET – ONLY – TYPE – <> ” (empty string) • The IN and EXISTS keywords are not allowed. • Any valid form of the FOLDER predicate (except NOT FOLDER) may be used. The DESCEND option is also allowed. • The following rules apply to the LIKE predicate: – The LIKE predicate may be used with pattern matching characters, but not against an ID attribute, only a string attribute. – The LIKE predicate can be used with an ESCAPE clause, but it is ignored. • The keywords TODAY, YESTERDAY, TOMORROW may not be used in the DATE function. • The following rules apply to all expressions in the WHERE clause: – Expressions may not contain the ISREPLICA or USER keywords. – Expressions may use any comparison operator. – Expressions may use an arithmetic operator, but they may not be used to form a compound expression.

Applicable to	Rule

	– Expressions that compare one property to another are not allowed. For example, subject=title is invalid.
	– Expressions in the following format are not allowed:
	`property_name operator('literal' operator 'literal)`
	– Expressions that force index correspondence between repeating properties are not allowed. Such expressions AND together expressions that reference repeating properties in the format:
	`predicate(repeating_attr_expr AND repeating_attr_expr)`
	For more information about forcing index correspondence, refer to "Forcing index correspondence in query results" on page 367.
	• The following additional rules apply to expressions in the format *first_expression operator second_expression*:
	– The *first_expression* is limited to one of the following:
	`property name`
	`upper(property name)`
	`lower(property name)`
	– The *second_expression* is limited to one of the following:
	`literal value`
	`upper(literal value)`
	`lower(literal value)`
	`the DATE() function`
	– The operator may be any valid operator.

ORDER BY clause	If included, this clause must be: `ORDER BY SCORE` SCORE must be referenced by name, not position, in the selected values list, also.
ENABLE clause	There are no restrictions on this clause.

E.1.21 Unregister

```
UNREGISTER [TABLE] [owner_name.]table_name
```

E.1.22 Update

```
UPDATE table_name SET column_assignmentsWHERE qualification
```

E.1.23 Update...Object

```
UPDATE [PUBLIC]type_name [(ALL)][correlation_var]
[WITHIN PARTITION partition_id {,partition_id}]
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]
```

E.2 DQL reserved words

If you using DQL reserved words as object or property names, enclose name in double quotes when using in a DQL query.

Table E-2: DQL reserved words

Initial letter	Reserved words
A	ABORT, ACL, ADD, ADD_FTINDEX, ADDRESS, ALL, ALLOW, ALTER, AND, ANY, APPEND, APPLICATION, AS, ASC, ASSEMBLIES, ASSEMBLY, ASSISTANCE, ATTR, AUTO, and AVG.
B	BAG, BEGIN, BETWEEN, BOOL, BOOLEAN, BROWSE, BUSINESS, and BY.
C	CABINET, CACHING, CHANGE, CHARACTER, CHARACTERS, CHAR, CHECK, COLLECTION, COMMENT, COMMIT, COMPLETE, COMPONENTS, COMPOSITE, COMPUTED, CONTAIN_ID, CONTAINS, CONTENT_FORMAT, CONTENT_ID, COUNT, CREATE, and CURRENT.
D	DATE, DATEADD, DATEDIFF, DATEFLOOR, DATETOSTRING, DAY, DEFAULT, DELETE, DELETED, DEPENDENCY, DEPTH, DESC, DESCEND, DISABLE, DISALLOW, DISPLAY, DISTINCT, DI_SESSION_DD_LOCALE, DOCBASIC, DOCUMENT, DOUBLE, DROP, and DROP_FTINDEX.
E	ELSE, ELSEIF, ENABLE, ENFORCE, ESCAPE, ESTIMATE, EXEC, EXECUTE, and EXISTS.
F	FALSE, FIRST, FLOAT, FOLDER, FOR, FOREIGN, FROM, FTINDEX, FT_OPTIMIZER, FULLTEXT, and FUNCTION.
G	GRANT and GROUP.

Initial letter	Reserved words
H	HAVING and HITS.
I	ID, IF, IN, INSERT, INT, INTEGER, INTERVAL, INTO, IS, ISCURRENT, ISPUBLIC, and ISREPLICA.
J	JOIN
K	KEY
L	LANGUAGE, LAST, LATEST, LEFT, LIGHTWEIGHT, LIKE, LINK, LIST, LITE, and LOWER.
M	MAPPING, MATERIALIZE, MATERIALIZATION, MAX, MCONTENTID, MEMBERS, MFILE_URL, MHITS, MIN, MODIFY, MONTH, MOVE, and MSCORE.
N	NODE, NODESORT, NONE, NOT, NOTE, NOW, NULL, NULLDATE, NULLID, NULLINT, and NULLSTRING.
O	OF, OBJECT, OBJECTS, ON, ONLY, OR, ORDER, OUTER, and OWNER.
P	PAGE_NO, PARENT, PARTITION, PATH, PERMIT, POLICY, POSITION, PRIMARY, PRIVATE, PRIVILEGES, PROPERTY, and PUBLIC. **Note:** The POSITION keyword, previously supported in SELECT queries against the full-text index, is no longer supported.
Q	QRY and QUALIFIABLE
R	RDBMS, READ, REFERENCES, REGISTER, RELATE, REMOVE, REPEATING, REPLACEIF, REPORT, REQUEST, and REVOKE.
S	SCORE, SEARCH, SELECT, SEPARATOR, SERVER, SET, SETFILE, SHAREABLE, SHARES, SMALLINT, SOME, STATE, STORAGE, STRING, SUBSTR, SUBSTRING, SUM, SUMMARY, SUPERTYPE, SUPERUSER, SUPPORT, SYNONYM, SYSADMIN, SYSOBJ_ID, and SYSTEM.
T	TABLE, TAG, TEXT, TIME, TINYINT, TO, TODAY, TOMORROW, TOPIC, TRAN, TRANSACTION, TRUE, TRUNCATE, and TYPE.
U	UNION, UNIQUE, UNLINK, UNREGISTER, USER, USING, UPDATE, and UPPER.
V	VALUE, VALUES, VERSION, VERIFY, and VIOLATION.
W	WHERE, WITH, WITHIN, WITHOUT, WORLD, WRITE, and WEEK.
Y	YEAR and YESTERDAY.

Appendix F. DQL 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" on page 120.

F.1 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 aggregate functions in a query. Some of these examples also demonstrate how to query repeating properties.

F.1.1 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 repository:

```
SELECT “user_name” FROM “dm_user”
```

This next example returns the names of the users and their user privileges within a repository:

```
SELECT “user_name”,“user_privileges” FROM “dm_user”
```

F.1.2 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 properties:

```
SELECT “name” FROM “dm_type”
WHERE “super_name” != ‘dm_sysobject’
AND “attr_count” > 15
```

The following example returns the types that were created one day before the current date together with the date and time that the types were last modified:

```
SELECT “name”, “r_modify_date” FROM “dm_type”
where “r_creation_date” >= DATE(YESTERDAY)
```

F.1.2.1 Searching repeating properties in a WHERE Clause

To search a repeating property for a specific value, you use the ANY predicate in the WHERE clause qualification unless you have included the ROW_BASED hint in the query. When you use this predicate, or if the query includes ROW_BASED, if any of the values in the specified repeating property meets the search criteria, the server returns the object.

The following example searches the authors property 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 property 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 property and returns any document that has a key word that begins with hap:

```
SELECT “object_name” FROM “dm_document”
WHERE ANY “keywords” LIKE ‘hap%’
```

F.1.3 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" on page 19.

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 repository:

```
SELECT MIN(DISTINCT “r_creation_date”),MAX(DISTINCT “r_creation_date”)
FROM “dm_document”
```

This final example returns the average number of properties in OpenText Documentum CM types:

```
SELECT AVG(DISTINCT “attr_count”) FROM “dm_type”
```

F.1.3.1 Using the GROUP BY clause

The GROUP BY clause provides a way to sort the returned objects on some property 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 property_name,aggregate_function FROM type_name
GROUP BY property_name
```

The property named in the target list must be the same as the property 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”
```

F.1.3.2 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
```

F.1.4 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 property [ASC|DESC] {,property [ASC|DESC]}
```

The *num* identifies one of the selected values by its position in the selected values list. The *property* 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
```

F.1.5 Using the asterisk (*) in queries

DQL lets you use the asterisk in the target list. If the query does not include the ROW_BASED hint and the asterisk is the selected value, the FROM clause may specify only a type name or a registered table—it cannot specify both. If the query includes the ROW_BASED hint, the FROM clause may include either or both object types and registered tables. The values returned by the asterisk depend on what is specified in the FROM clause and whether the ROW_BASED hint is included. For examples and complete details, refer to “The asterisk (*) as a selected value” on page 140.

F.2 Searching cabinets and folders

In OpenText Documentum CM, 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. There is a limit of 25,000 contained folders within the specified folder when you include DESCEND in the predicate.

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)
```

F.3 Querying registered tables

A registered table is a table from your underlying RDBMS that has been registered with the repository. 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 “DQL statements“ on page 43.

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)
```

F.4 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 F-1:

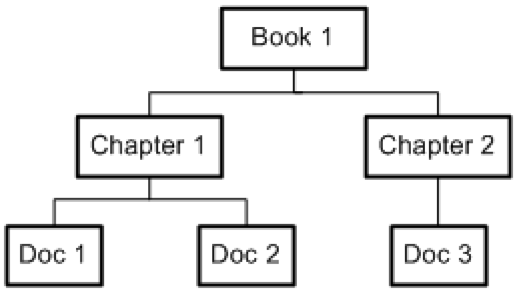

Figure F-1: Virtual document model

F.4.1 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’)
```