Oracle® Multimedia

OraDAV Driver Guide

11*g* Release 2 (11.2)

E10782-01

December 2009

Oracle Multimedia OraDAV driver enables WebDAV access to media content in an Oracle database.

ORACLE®

Oracle Multimedia OraDAV Driver Guide, 11*g* Release 2 (11.2)

E10782-01

Primary Author: Sue Pelski

Contributing Author: Chuck Murray

Contributors: Rob Abbott, Janet Blowney, Ling Cheng, Dave Diamond, Bob Hanckel, John Janosik, Sharon Juhasz, Dong Lin, Sue Mavris, Dan Mullen, Mike Rubino

Contents

Index

List of Examples

List of Tables

Preface

This guide provides information about Oracle Multimedia OraDAV Driver.

In Oracle Database 11g Release 1 (11.1), the name Oracle *inter*Media was changed to Oracle Multimedia. The feature remains the same, only the name has changed. References to Oracle *inter*Media were replaced with Oracle Multimedia, however some references to Oracle *inter*Media or *inter*Media might still appear in graphical user interfaces, code examples, and related documents in the Documentation Library for Oracle Database 11*g*.

Audience

This guide is primarily for people who install and configure software in an Oracle environment. You should be familiar with Oracle Database concepts, including basic installation and configuration. In addition, you should be somewhat familiar with Oracle Application Server or Oracle Fusion Middleware concepts.

This guide is also for application developers and database administrators who are interested in storing, retrieving, and manipulating audio, image, video, and heterogeneous media data in a database. Before using this guide, familiarize yourself with the concepts presented in *Oracle Multimedia User's Guide* and *Oracle Multimedia Reference*.

Documentation Accessibility

Our goal is to make Oracle products, services, and supporting documentation accessible to all users, including users that are disabled. To that end, our documentation includes features that make information available to users of assistive technology. This documentation is available in HTML format, and contains markup to facilitate access by the disabled community. Accessibility standards will continue to evolve over time, and Oracle is actively engaged with other market-leading technology vendors to address technical obstacles so that our documentation can be accessible to all of our customers. For more information, visit the Oracle Accessibility Program Web site at `http://www.oracle.com/accessibility/`.

Accessibility of Code Examples in Documentation

Screen readers may not always correctly read the code examples in this document. The conventions for writing code require that closing braces should appear on an otherwise empty line; however, some screen readers may not always read a line of text that consists solely of a bracket or brace.

Accessibility of Links to External Web Sites in Documentation

This documentation may contain links to Web sites of other companies or organizations that Oracle does not own or control. Oracle neither evaluates nor makes any representations regarding the accessibility of these Web sites.

Deaf/Hard of Hearing Access to Oracle Support Services

To reach Oracle Support Services, use a telecommunications relay service (TRS) to call Oracle Support at 1.800.223.1711. An Oracle Support Services engineer will handle technical issues and provide customer support according to the Oracle service request process. Information about TRS is available at `http://www.fcc.gov/cgb/consumerfacts/trs.html`, and a list of phone numbers is available at `http://www.fcc.gov/cgb/dro/trsphonebk.html`.

Related Documents

Note: For details about installation and configuration, and for information added after the release of this guide, refer to the online `README.txt` file included in this kit.

For more information about Oracle Multimedia, see the following documents in the Oracle Database Online Documentation Library:

- *Oracle Multimedia Reference*
- *Oracle Multimedia User's Guide*

For more information about using Oracle Multimedia and OraDAV in a development environment, see these Oracle resources:

- Oracle Database Online Documentation Library
- Oracle Application Server Online Documentation Library
- Oracle Fusion Middleware Online Documentation Library

Conventions

The following text conventions are used in this document:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
italic	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
`monospace`	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

What's New in Oracle Multimedia OraDAV Driver?

This document summarizes any new features introduced in the current release.

New Features for Oracle Database 11*g* Release 11.2

Oracle Database 11*g* Release 2 (11.2) includes *no* new features for *Oracle Multimedia OraDAV Driver Guide*.

1

Overview of Oracle Multimedia OraDAV Driver

Oracle Multimedia (formerly Oracle *inter*Media) OraDAV driver enables WebDAV access to media content in an Oracle database.

WebDAV is a protocol extension to HTTP 1.1 that supports distributed authoring and versioning. WebDAV enables the Internet to become a transparent read and write medium, where content can be checked out, edited, and checked in to a URL address. The mod_dav module is an implementation of the WebDAV specification. The standard implementation of the mod_dav module supports read and write access to files.

OraDAV is the Oracle implementation of WebDAV. OraDAV refers to the capabilities available through the mod_oradav module, which is an extended implementation of the mod_dav module. The mod_oradav module can read and write to local files or to an Oracle database. The Oracle database must include an OraDAV driver (a stored procedure package) that the mod_oradav module calls to map WebDAV activity to database activity. Essentially, the mod_oradav module enables WebDAV clients to connect to an Oracle database, read and write content, and query and lock documents in various schemas. The mod_oradav module is integrated with the Oracle HTTP Server, which is the Web server component of Oracle Application Server 10*g* and Oracle Fusion Middleware.

Each OraDAV driver manages documents in a repository in an Oracle database by providing support for the following WebDAV functions over the Internet:

- Reading and writing documents
- Locking and unlocking documents
- Managing hierarchies of information, including the following:
  - Creating them
  - Populating them
  - Deleting them
- Retrieving properties associated with documents
- Associating properties with specific documents

Oracle Multimedia OraDAV driver consists of a set of PL/SQL packages installed in the ORDSYS schema. This driver manages a specific repository, which is called an Oracle Multimedia container (or a **container**) in an Oracle database. This container includes media tables to store media content by default. This container also includes auxiliary tables that enable the driver to provide WebDAV functions on the media content. The Oracle Multimedia container can be created in any user schema.

2

OraDAV Container System Tables and Views

When you create a container, a set of OraDAV system tables and views is created to provide the infrastructure for the container. The options you choose when creating containers affect the content of the OraDAV system tables and views (see Section 2.1).

The following SQL procedures are provided for administrators to create and delete containers:

- `orddavcc.sql`
- `orddavdc.sql`

The `orddavcc.sql` procedure creates a container that is used by OraDAV to maintain information and perform operations. The name of each table and view in that container begins with the value of the <Container Name> option.

The `orddavdc.sql` procedure lets you delete a container from the SQL command line.

This chapter includes the following sections:

2.1 Options for Creating Oracle Multimedia Containers

When you create Oracle Multimedia containers, you use the following options.

> **Note:** The logic to create a container backs out of the creation of DDL objects if it cannot complete the entire operation.

Container Name

This option is a prefix of 20 characters or less. The value you enter in this option serves as a prefix for all the tables, views, indexes, sequences, triggers, and tablespaces that are created for the container. Choose a container name whose prefix does not cause namespace clashes with other data definition language (DDL) objects that were created in the schema. The default value is `oradav`.

Container Size

This option is the size of the storage you want to allocate for the content. The value you enter in this option is an integer that represents megabytes of storage. Choose a number that will not exhaust the disk space on the system where Oracle is running. When the container is created, two tablespaces are generated. Twenty percent of the storage allocation is used to create a tablespace that holds the OraDAV auxiliary tables,

including all of the DDL objects created for the container. The remaining eighty percent of the storage allocation is used to create another tablespace that is dedicated to the storage of media. The default value is `1000`. The minimum value is `100`.

> **Note:** The size of the container dictates how long the `orddavcc.sql` script is to run. If you are creating a large amount of storage, expect a long delay before the script returns control and reports that it has completed.

NoExecute

This option is supported if you only want to generate a script for creating the container, without executing any DDL. You can execute this script in SQL*Plus later. Use this option after familiarizing yourself with the system tables created, and when you want to tune the physical storage characteristics of the container tables. The default value is `n`.

LogFileDirectory

This option works in conjunction with the LogFile option. It prompts you for a file directory to which the Oracle database has been configured to write using the UTL_FILE utility. Logging is done by the server on the server system, rather than by SQL*Plus or the client. There is no default value.

LogFile

This option works in conjunction with the LogFileDirectory option. It enables you to enter a name for the log file when the LogFileDirectory option is set. The default value is `container_install.sql`.

This option is a file that captures all DDL generated while creating a container. This is useful for re-creating the container or tuning storage attributes. This option uses UTL_FILE directives.

Trace Output

This option enables tracing. Tracing is an alterative way of letting you see the DDL that is generated. Tracing output is raw DDL that is generated, and that is less readable than the log file. Tracing output is a good way to see what was built, or to debug problems with the DDL that is executed.

Entering a value of `y` causes DDL to be generated and sent to serveroutput. In addition, you must enter the following command in SQL*Plus:

```
SET SERVEROUTPUT ON
```

The default value is `n`.

Add index.html

This option lets you verify the successful creation of a container using a Web browser. Entering a value of `y` causes a small `index.html` file to be added to the container that is being created. This `index.html` file can later be deleted or overwritten with any DAV client. The default value is `y`.

Use Oracle Multimedia objects

This option lets you specify the storage method for Oracle Multimedia media files. Entering a value of `y` causes the media files to be stored in three separate tables of

ORDImage, ORDAudio, and ORDVideo objects. Otherwise, all files are stored as BLOBs. The default value is `Y`.

Tablespace datafile directory

This option lets you specify the server directory where you want the tablespace data files for the container to be created. Enter the desired server directory (including the trailing slash, if appropriate). If no server directory is specified, the default location for your database is used.

2.2 OraDAV System Tables and Views

Table 2–1 lists each of the OraDAV system tables and views, where <*xxx*> represents the value of the <Container Name> option.

Table 2–1 Container System Tables and Views

Table or View	Description
<*xxx*>$ACE	Access control entry
<*xxx*>$ASL	Advanced searching and locating
<*xxx*>$AUDIO	Content storage for ORDAudio files, if the value of the <Use Oracle Multimedia objects> option is `Y`
<*xxx*>$BLOB	Content storage for files other than ORDAudio, ORDVideo, or ORDImage files, if the value of the <Use Oracle Multimedia objects> option is `Y`. Or, content storage for all media files, if the value of the <Use Oracle Multimedia objects> option is `N`
<*xxx*>$CONTAIN	Container identifier
<*xxx*>$CONTAINER	Information used internally to identify, maintain, and track the container
<*xxx*>$CONTENT	Lists content to storage mapping
<*xxx*>$IMAGE	Content storage for ORDImage files, if the value of the <Use Oracle Multimedia objects> option is `Y`
<*xxx*>$LOCKS	DAV locks
<*xxx*>$MIME	MIME mapping and content types Modifications to this table are permitted. For more information about this table, see Section 2.2.1.
<*xxx*>$PATH	Hierarchy of path names
<*xxx*>$PRINCIPAL	User names and authorization information
<*xxx*>$PROP	Property information for resources Modifications to this table are permitted. For more information about this table, see Section 2.2.2.
<*xxx*>$RESOURCE	Convenience view that selects certain columns from xxx$PATH and xxx$ASL based on an equijoin of their DOC_ID columns
<*xxx*>$STORE	Table for registering storage areas used in partitioning files by MIME type, if the value of the <Use Oracle Multimedia objects> option is `Y`
<*xxx*>$TBLOB	Temporary table used for generated content
<*xxx*>$VIDEO	Content storage for ORDVideo files, if the value of the <Use Oracle Multimedia objects> option is `Y`

Note: With the exception of <Container Name>$MIME and <Container Name>$PROP, these tables and views are *read-only*.

2.2.1 <Container Name>$MIME Table

The <Container Name>$MIME table contains information about MIME mapping and content types. Modifications to this table are permitted.

Table 2–2 describes the columns in the <Container Name>$MIME table.

Table 2–2 Columns in the <Container Name>$MIME Table

Column Name	Data Type	Explanation
EXTENSION	VARCHAR2(30)	A file extension typically used by files of a specified content type. The period must be included in the extension. For example: `.jpg`
PREFERRED_EXT	VARCHAR2(1)	The value is `T` if EXTENSION is the file extension to be used by OraDAV when generating files of the associated content type; the value is `F` if EXTENSION is not to be used in this case. This value is most important when a content type is associated with multiple extensions. For example, if `image/jpeg` is associated with the `.jpg`, `.jpeg`, and `.jpe` extensions, you must specify one with PREFERRED_EXT as `T`.
CONTENTTYPE	VARCHAR2(80)	MIME type typically associated with a file with the extension. For a given CONTENTTYPE, there must be one (and only one) PREFERRED_EXT set to `T`. (See the explanation for the PREFERRED_EXT column name.)
ANNOLEVEL	VARCHAR2(1)	Degree to which data for the content type is to be annotated: `D` (detailed) or `S` (summary). Ignored if the ANNOROUTINE column is null.
ANNOROUTINE	VARCHAR2(108)	Annotation routine: the name of the PL/SQL procedure to be called to perform the annotation. The name of the annotation routine should be qualified with the schema and package name, if necessary. The proper privileges and access rights must have been granted to enable the database schema in which the container was created to execute the annotation routine.
STORAGE_AREA	VARCHAR2(30)	Storage identifier, corresponding to a STORAGE_AREA column value in the xxx$STORE table (described in Table 2–1). For example, if the STORAGE_AREA value for `image/jpeg` is `IMAGE`, then all files of this CONTENTTYPE, at file creation time, are inserted and stored in the storage area with storage identifier `IMAGE`.

The ANNOLEVEL and ANNOROUTINE values are not used unless the ORAAnnotate parameter is set to TRUE in the <Location> directive in the Oracle HTTP Server OraDAV configuration file. The ORAAnnotate parameter controls the automatic extraction and storage of properties from resources. If this parameter is set

to TRUE, an OraDAV-enabled Oracle HTTP Server performs the following additional actions whenever a resource is added to a collection:

- It selects the following from the <Container Name>$MIME table in the container: the name of the PL/SQL procedure to call to perform the annotation, and annotation level (either summary or detailed).
- It invokes the annotation routine, passing the following parameters: an XML string (described in Section 3.4) representing the context parameter, the OraDAV document ID for the resource being annotated, the BLOB handle for the resource's data, and the annotation level (`S` for summary, `D` for detailed). The annotation routine is expected to parse attributes from the resource's data and store them as properties in the container's <Container Name>$PROP table (described in Section 2.2.2). If the annotation level is `S`, all the properties are concatenated into one XML string and stored as a single property in the <Container Name>$PROP table. If the annotation level is `D`, each property is stored as its own row in the <Container Name>$PROP table.

The following example displays a row from the <Container Name>$MIME table. (The output is slightly reformatted for readability.)

```
SQL> SELECT * FROM test$mime WHERE extension='.jpg';

EXTENSION PREFERRED_EXT CONTENTTYPE ANNOLEVEL  ANNOROUTINE   STORAGE_AREA

--------------------------------------------------------------------------------

 .jpg        T            image/jpeg   D                       IMAGE
```

You can modify the <Container Name>$MIME table to perform any of the following tasks:

- Add rows to support additional content types.
- Change the storage areas used for newly added objects for certain content types. (Existing objects remain in their current storage areas.)
- Add or change the annotation routine to be called for a particular content type.

2.2.2 <Container Name>$PROP Table

The <Container Name>$PROP table contains information about resource properties. Modifications to this table are permitted.

Table 2–3 describes the columns in the <Container Name>$PROP table.

Table 2–3 Columns in the <Container Name>$PROP Table

Column Name	Data Type	Explanation
DOC_ID	NUMBER(38)	Internal document identifier. DOC_ID values are unique and never reused.
NAMESPACE	VARCHAR2(538)	XML namespace associated with this property. Used to avoid ambiguity if there are duplicate TAG values.
TAG	VARCHAR2(100)	Name assigned to the property. Used with NAMESPACE to identify the property.
LANGUAGE	VARCHAR2(100)	Language in which the property is written.
DESCRIPTION	VARCHAR2(4000)	Additional information (if any) about the property.

Table 2–3 (Cont.) Columns in the <Container Name>$PROP Table

Column Name	Data Type	Explanation
DEADORLIVE	VARCHAR2(1)	A value of `D` represents a **dead** (static) property. A value of `L` represents a **live** (dynamic) property. Dead properties are not derived directly from the document; they are created and maintained by the client. Live properties are derived directly from the document's content; they are maintained by the server, and most cannot be modified by the client.
READONLY	VARCHAR2(1)	Indicates whether the property is read-only: `T` (True) or `F` (False).
PROPV_TYPE	VARCHAR2(30)	Data type for the property: VARCHAR2, DATE, NUMBER, or CLOB.
VAL_VARCHAR2	VARCHAR2(4000)	If PROPV_TYPE is VARCHAR2, the value for the property.
VAL_CLOB	CLOB	If PROPV_TYPE is CLOB, the value for the property.
VAL_DATE	DATE	If PROPV_TYPE is DATE, the value for the property.
VAL_NUMBER	NUMBER	If PROPV_TYPE is NUMBER, the value for the property.

It is appropriate for writers of annotation routines to make insertions into the <Container Name>$PROP table (but not into other container tables, with the possible exception of the <Container Name>$MIME table).

Note: If a document is being overwritten, all existing live properties are deleted before an annotation routine is called.

3

OraDAV Programming Interface

OraDAV provides PL/SQL methods (including procedures and functions) that let you **expose** Oracle Multimedia objects. Exposing objects makes them visible as files in a container for access through a Web browser. When you expose an object, it is not actually copied into the container; instead, only a reference to the content is inserted into the container. Oracle Multimedia OraDAV driver does not support DAV methods that modify exposed objects.

These PL/SQL methods are in a package named DAV_PUBLIC under the ORDSYS schema. By using these methods, you can expose content from existing tables that contain columns of type ORDImage, ORDAudio, ORDVideo, ORDDoc, or BLOB.

These tables can be in different schemas from the one containing your OraDAV container, as long as the container's schema is granted the proper access to the tables. (See Chapter 2 for information about containers.)

> **Note:** Do not use methods in the DAV_PUBLIC package on any of the container system tables and views described in Chapter 2. Use these methods only for objects stored in tables other than the container system tables and views.

This chapter includes the following sections:

3.1 Options for Exposing Content

You can expose a single object at a time, or all objects in a column:

- The Expose_Resource_By_Rowid procedure exposes the single object in a specified column in the row with the associated ROWID.
- The Expose_OracleMultimedia_Column and Expose_Blob_Column procedures expose all objects in a specified column.

The following sections describe examples of both options for exposing objects. See Chapter 4 for reference information about these procedures.

The examples in this chapter as well as those in Chapter 4 assume that you work for a real estate firm that wants to make pictures and descriptions of houses for sale available at its Web site. Your database and system setup includes the following:

- A container named `mywebsite` has been created in schema SCOTT (see the creating container example in the `README.txt` file).
- User REALTOR has a table called HOUSES.
- The HOUSES table has a column called HOUSE_ID that uniquely identifies each row.
- The HOUSES table has a column called HOUSE_PIX, which is of type ORDImage.
- The HOUSES table has a column called HOUSE_DOC, which is of type BLOB.
- The ORDImage.setProperties( ) method has been invoked such that the contentLength and mimeType attributes of the ORDImage objects are set.
- User REALTOR has granted SELECT access on HOUSES to SCOTT.
- An OraDAV-enabled Oracle HTTP Server is running on host `mywebserver`.
- That Oracle HTTP Server has been configured with an OraDAV-enabled location called `/oradav`.

To run these examples, connect to the database as SCOTT.

3.1.1 Example: Exposing a Single Object

Example 3–1 extracts information to uniquely identify the desired object and exposes it, associating it with a URL.

Example 3–1 Exposing a Single Object

```
DECLARE
   ldocid INTEGER;
   loblen INTEGER;
   lrowid UROWID;
BEGIN
SELECT rowid,DBMS_LOB.GETLENGTH(HOUSE_DOC)
           INTO lrowid, loblen
           FROM REALTOR.HOUSES
           WHERE HOUSE_ID = 1;
ORDSYS.DAV_PUBLIC.expose_resource_by_rowid (
   ORDSYS.DAV_PUBLIC.generate_ctx('mywebsite'),
   'REALTOR',
   'HOUSES',
   'HOUSE_DOC',
   'BLOB',
   lrowid,
   '/external_collection',
   'house',
   '.html',
   'text/html',
   loblen,
   SYSDATE,
   ORDSYS.DAV_PUBLIC.DUPE_MODE_FAIL,
   ldocid);
COMMIT;
END;
/
```

Upon successful execution of the code in this example, the following URL returns the Web page from REATOR.HOUSES where the HOUSE_ID is `1` (assuming that `httpd.conf` contains `<Location /oradav>`):

```
http://mywebserver/oradav/external_collection/house.html
```

3.1.2 Example: Exposing All Objects in a Column

If you want to expose all objects in a multimedia column and if it seems too tedious to expose the objects one at a time, you can expose all of the objects in the column by using the Expose_OracleMultimedia_Column procedure (for ORDImage, ORDAudio, ORDVideo, or ORDDoc data) or the Expose_Blob_Column procedure (for BLOB data).

Example 3–2 exposes all the images in the HOUSE_PIX column.

Example 3–2 Exposing All Objects in a Column

```
DECLARE
BEGIN
ORDSYS.DAV_PUBLIC.expose_OracleMultimedia_column(
   ORDSYS.DAV_PUBLIC.generate_ctx('mywebsite'),
   'REALTOR',
   'HOUSES',
   'HOUSE_PIX',
   'HOUSE_ID',
   NULL,
   NULL,
   NULL,
   1,
   ORDSYS.DAV_PUBLIC.DUPE_MODE_FAIL);
COMMIT;
END;
/
```

Upon successful execution of the code in this example, all house pictures in REALTOR.HOUSES can be returned by URLs that all start with `http://mywebserver/oradav/REALTOR/HOUSES/HOUSE_PIX`. The remainder of each URL is the value of the HOUSE_ID column for that house. For example, the following URL returns the image from REALTOR.HOUSES where the HOUSE_ID is `1`:

```
http://mywebserver/oradav/REALTOR/HOUSES/HOUSE_PIX/1.jpg
```

3.2 DAV Methods on Exposed Data

The OraDAV implementation allows some DAV protocol methods on exposed data and disallows other methods, as shown in Table 3–1.

Table 3–1 DAV Methods on Exposed Data

Method	Allowed?
GET	Yes
COPY	Yes, as long as the destination is not an exposed collection
PROPFIND	Yes (but you can restrict massive collections by using <Limit> in the `httpd.conf` file)
OPTIONS	Yes
DELETE	Yes (The reference to the data is removed.)

Table 3–1 (Cont.) DAV Methods on Exposed Data

Method	Allowed?
MKCOL	No
PROPPATCH	Yes
PUT	No
LOCK	No
MOVE	No
POST	No
UNLOCK	Yes (but it always fails because LOCK is not allowed)

If you are using a client on exposed data and a command or operation fails, it may be because the application's command or operation maps to (is implemented using) a DAV method that OraDAV does allow to be used on exposed data.

For more information about DAV methods, see RFC2518 (*HTTP Extensions for Distributed Authoring -- WEBDAV*).

3.3 Duplicate File Behavior Options

Procedures that expose a resource or a column have a p_dupe_behavior parameter that controls the behavior when a call to the procedure would result in a duplicate file being added to the collection. For example, if the procedure attempts to expose a file named `emp100id.jpg` and a file named `emp100id.jpg` is already exposed at the specified location (path), the value of the p_dupe_behavior parameter determines how this condition is handled.

Table 3–2 lists the currently supported values for the dupe_behavior parameter.

Table 3–2 dupe_behavior Parameter Values

Value	Explanation
DUPE_MODE_FAIL	Any duplicate file names result in an exception. Note that if the p_add_triggers parameter is set to TRUE, the trigger could also generate an exception from a simple SQL INSERT or UPDATE statement. Your application must be prepared to handle these exceptions.

See the descriptions for the p_dupe_behavior parameter as well as the usage notes for the Expose_Blob_Column, Expose_OracleMultimedia_Column, and Expose_Resource_By_Rowid procedures.

3.4 Context Parameter

A context parameter is required for enabling annotation and exposing or unexposing data. This parameter is a string in XML format. Because almost every OraDAV routine operates on a container, the context parameter must, at minimum, specify the container name, as shown in Example 3–3.

Example 3–3 Minimal Context Parameter

```
<ORADAV>
  <DAVPARAM>
    <ORACONTAINERNAME>sales</ORACONTAINERNAME>
```

```
  </DAVPARAM>
</ORADAV>
```

For many procedure calls, a minimal context parameter string such as the one shown in this example is sufficient. You can use the Generate_Ctx function to generate a minimal context string.

4

OraDAV Driver API Reference

This chapter contains reference information for the methods (procedures and functions) in the ORDSYS.DAV_PUBLIC PL/SQL package. These methods are presented in alphabetical order.

Before you use the methods in this package, be sure you understand the concepts and guidelines in Chapter 3, which describes the OraDAV programming interface.

Table 4–1 lists the methods. The rest of this chapter presents detailed reference information for each method.

Table 4–1 OraDAV Methods

Procedure or Function	Description
Cversion	Function that returns the container version number (also called the physical version number).
Expose_Blob_Column	Procedure that makes objects in a column of type BLOB visible by exposing the content of each object as a file with an associated URL.
Expose_OracleMultimedia_Column	Procedure that makes objects in a column of type ORDImage, ORDAudio, ORDVideo, or ORDDoc visible by exposing the content of each object as a file with an associated URL.
Expose_Resource_By_Rowid	Procedure that makes a single object visible (associated with a specified ROWID) by exposing its content as a file with an associated URL.
Generate_Ctx	Function that returns an XML string with minimal context information for the specified container.
Unexpose_Column	Procedure that removes the visibility of objects in a column by reversing the effect of the Expose_Blob_Column or Expose_OracleMultimedia_Column procedure.
Unexpose_Resource_By_Rowid	Procedure that removes the visibility of a single object (associated with a specified ROWID) by reversing the effect of the Expose_Resource_By_Rowid procedure.
Version	Function that returns the release (version) number of the DAV_PUBLIC package.

Cversion

Format

ORDSYS.DAV_PUBLIC.Cversion() RETURN VARCHAR2;

Description

Returns the container (or physical) version number.

Parameters

None.

Usage Notes

This function returns a number describing the container (physical) version that the Oracle Multimedia OraDAV driver supports.

Contrast this function with the Version function, which returns the package version number.

Examples

The following example returns the container (physical) version number.

```
SELECT ORDSYS.DAV_PUBLIC.Cversion FROM DUAL;

  CVERSION
----------
       1.5
```

Expose_Blob_Column

Format

```
ORDSYS.DAV_PUBLIC.Expose_Blob_Column(
    p_ctx              IN  VARCHAR2,
    p_schema_name      IN  VARCHAR2,
    p_table_name       IN  VARCHAR2,
    p_media_column_name IN  VARCHAR2,
    p_date_column_name  IN  VARCHAR2,
    p_mime_column_name  IN  VARCHAR2,
    p_default_mimetype  IN  VARCHAR2,
    p_key_column_name   IN  VARCHAR2,
    p_key_prefix       IN  VARCHAR2,
    p_key_suffix       IN  VARCHAR2,
    p_parent_path      IN  VARCHAR2,
    p_add_triggers     IN  NUMBER(38),
   p_dupe_behavior     IN  NUMBER(38));
```

Description

Makes objects in a column of type BLOB visible by exposing the content of each object as a file with an associated URL.

Parameters

p_ctx
Context string for the container on which to operate. (See Section 3.4 for detailed information about the context.)

p_schema_name
Name of the schema containing the table with the column to be exposed.

p_table_name
Name of the table or view containing the column to be exposed. Must not be one of the container system tables or views described in Section 2.2.

p_media_column_name
Name of the BLOB column to be exposed.

p_date_column_name
Name of the column of type DATE that contains the last-modified date for the data in the BLOB column. If specified as `null`, SYSDATE (the current system date and time) is used.

p_mime_column_name
Name of the column in the p_table_name parameter that identifies the MIME type. If specified as `null`, the p_default_mimetype parameter is used.

p_default_mimetype
The MIME type to be used if the value of the p_mime_column_name parameter is `null` or if the data selected from the column is null.

p_key_column_name
Name of the column in the p_table_name parameter that has unique values to be used as the base file names and the final part of each URL.

p_key_prefix
Prefix to be used in the base file names and the final part of each URL. For example, if the value of the p_key_prefix parameter is `emp`, the value of the p_key_suffix parameter is `id`, and a row includes a base column value of `100`, the resulting file name is `emp100id.jpg` (if the MIME type is `image/jpeg`).

p_key_suffix
Suffix to be used in the base file names and the final part of each URL. (The suffix is not a file extension such as `.jpg` or `.wav`.) For example, if the value of the p_key_prefix parameter is `emp`, the value of the key_suffix parameter is`id`, and a row includes a base column value of `100`, the resulting file name is `emp100id.jpg` (if the MIME type is `image/jpeg`).

p_parent_path
Path of the folder to which the column is to be mapped. If specified as `null`, the default is a path in the following format:

```
/<p_schema_name>/<p_table_name>/<p_media_column_name>/
```

p_add_triggers
A value that determines whether triggers are added. If the value of this parameter is `0`, no triggers are added; otherwise, if the value is `1`, INSERT, UPDATE, and DELETE triggers are added to the p_table_name parameter so that the contents of the WebDAV client are automatically changed to reflect the actions on the underlying table. The default value is `0`.

p_dupe_behavior
Action to be taken if a call to the procedure would result in a duplicate file being added to the collection. For more information and a list of acceptable values, see Section 3.3.

Usage Notes

Before using this procedure, be sure you understand the concepts, guidelines, and examples in Section 3.1.

The following guidelines apply to the folder `parent_path`:

- If the path already exists, it must point to an empty folder that is not locked. For example, if you specify the path `/external_data/my_blobs`, and the subfolder `my_blobs` already exists in the folder `external_data`, the subfolder `my_blobs` must not contain any files or folders and it must not be locked.
- If the path does not exist, its parent folder must exist and the parent folder must not be locked. Using the preceding example, if the subfolder `my_blobs` does not exist, the folder `external_data` must exist and it must not be locked.

For information about DAV methods that are allowed and disallowed on exposed data, see Table 3–1.

To remove the visibility of a column that had been exposed by this procedure, use the Unexpose_Column procedure.

Examples

The following example exposes all the images in the HOUSE_DOC column. This example uses the same set of assumptions as those that are used in Section 3.1.

```
DECLARE
BEGIN
ORDSYS.DAV_PUBLIC.Expose_Blob_Column(
   ORDSYS.DAV_PUBLIC.Generate_Ctx('mywebsite'),
   'REALTOR',
   'HOUSES',
   'HOUSE_DOC',
   NULL,
   NULL,
   'text/html',
   'HOUSE_ID',
   NULL,
   NULL,
   NULL,
   0,
   ORDSYS.DAV_PUBLIC.DUPE_MODE_FAIL);
COMMIT;
END;
/
```

Upon successful execution of the code in this example, all house descriptions in REALTOR.HOUSES can be returned by URLs that start with `http://mywebserver/oradav/REALTOR/HOUSES/HOUSE_DOC/`. The remainder of each URL is the value of the HOUSE_ID column for that house. For example, the following URL returns the description from REALTOR.HOUSES where the value of HOUSE_ID is `1`:

```
http://mywebserver/oradav/REALTOR/HOUSES/HOUSE_DOC/1.htm
```

Expose_OracleMultimedia_Column

Format

```
ORDSYS.DAV_PUBLIC.Expose_OracleMultimedia_Column(
    p_ctx              IN  VARCHAR2,
    p_schema_name      IN  VARCHAR2,
    p_table_name       IN  VARCHAR2,
    p_media_column_name IN  VARCHAR2,
    p_key_column_name  IN  VARCHAR2,
    p_key_prefix       IN  VARCHAR2,
    p_key_suffix       IN  VARCHAR2,
    p_parent_path      IN  VARCHAR2,
    p_add_triggers     IN  NUMBER(38),
    p_dupe_behavior    IN  NUMBER(38));
```

Description

Makes objects in a column of type ORDImage, ORDAudio, ORDVideo, or ORDDoc visible by exposing the content of each object as a file with an associated URL.

Parameters

p_ctx
Context string for the container on which to operate. (See Section 3.4 for detailed information about the context.)

p_schema_name
Name of the schema containing the table with the column to be exposed.

p_table_name
Name of the table or view containing the column to be exposed. Must not be one of the container system tables or views described in Section 2.2.

p_media_column_name
Name of the column of type ORDImage, ORDAudio, ORDVideo, or ORDDoc to be exposed.

p_key_column_name
Name of the column in the p_table_name parameter that has unique values to be used as the base file names and the final part of each URL.

p_key_prefix
Prefix to be used in the base file names and the final part of each URL. For example, if the value of the p_key_prefix parameter is `emp`, the value of the p_key_suffix parameter is `id`, and a row includes a base column value of `100`, the resulting file name is `emp100id.jpg` (if the MIME type is `image/jpeg`).

p_key_suffix
Suffix to be used in the base file names and the final part of each URL. (The suffix is not a file extension such as `.jpg` or `.wav`.) For example, if the value of the p_key_prefix parameter is `emp`, the value of the p_key_suffix parameter is `id`, and a row includes a base column value of `100`, the resulting file name is `emp100id.jpg` (if the MIME type is `image/jpeg`).

p_parent_path
Path of the folder to which the column is to be mapped. If specified as `null`, the default is a path in the following format:

```
/<p_schema_name>/<p_table_name>/<p_media_column_name>/
```

p_add_triggers
A value that determines whether triggers are added. If the value of this parameter is `0`, no triggers are added; otherwise, if the value is `1`, INSERT, UPDATE, and DELETE triggers are added to the p_table_name parameter so that the contents of the WebDAV client are automatically changed to reflect the actions on the underlying table. The default value is `0`.

p_dupe_behavior
Action to be taken if a call to the procedure would result in a duplicate file being added to the collection. For more information and a list of acceptable values, see Section 3.3.

Usage Notes

Before using this procedure, be sure you understand the concepts, guidelines, and examples in Section 3.1.

For information about DAV methods that are allowed and disallowed on exposed data, see Table 3–1.

To remove the visibility of a column that had been exposed by this procedure, use the Unexpose_Column procedure.

Examples

The following example exposes all the images in the HOUSE_PIX column. This example uses the same set of assumptions as those that are used in Section 3.1.

```
DECLARE
BEGIN
ORDSYS.DAV_PUBLIC.expose_OracleMultimedia_column(
   ORDSYS.DAV_PUBLIC.Generate_Ctx('mywebsite'),
   'REALTOR',
   'HOUSES',
   'HOUSE_PIX',
   'HOUSE_ID',
   NULL,
   NULL,
   NULL,
   1,
   ORDSYS.DAV_PUBLIC.DUPE_MODE_FAIL);
COMMIT;
END;
/
```

Upon successful execution of the code in this example, all house pictures in REALTOR.HOUSES can be returned by URLs that start with

`http://mywebserver/oradav/REALTOR/HOUSES/HOUSE_PIX`. The remainder of each URL is the value of the HOUSE_ID column for that house. For example, the following URL returns the image from REALTOR.HOUSES where the value of HOUSE_ID is `1`:

```
http://mywebserver/oradav/REALTOR/HOUSES/HOUSE_PIX/1.jpg
```

Expose_Resource_By_Rowid

Format

```
ORDSYS.DAV_PUBLIC.Expose_Resource_By_Rowid(
    p_ctx           IN  VARCHAR2,
    p_schema_name   IN  VARCHAR2,
    p_table_name    IN  VARCHAR2,
    p_value_colname IN  VARCHAR2,
    p_value_coltype IN  VARCHAR2,
    p_prowid        IN  UROWID,
    p_parent_path   IN  VARCHAR2,
    p_base_file_name IN VARCHAR2,
    p_extension     IN  VARCHAR2,
    p_mimetype      IN  VARCHAR2,
    p_content_length IN INTEGER,
    p_creation_date IN  DATE,
    p_dupe_behavior IN  NUMBER(38),
    p_docid         OUT NUMBER(38));
```

Description

Makes a single object (associated with a specified ROWID) visible by exposing its content as a file with an associated URL.

Parameters

p_ctx
Context string for the container on which to operate. (See Section 3.4 for detailed information about the context.)

p_schema_name
Name of the schema containing the table with the object to be exposed.

p_table_name
Name of the table or view containing the object to be exposed. Must not be one of the container system tables or views described in Section 2.2.

p_value_colname
Name of the column containing the object to be exposed.

p_value_coltype
Data type of the p_value_colname parameter (for example, BLOB or ORDImage).

p_prowid
The ROWID of the row in the p_table_name parameter to be exposed.

p_parent_path
Path of the folder to which the object is to be mapped. If specified as `null`, the default is a path in the following format:

```
/< p_schema_name>/< p_table_name>/< p_media_column_name>/
```

p_base_file_name
Name to be used for the base file in the final part of the URL. For example, if the value of the p_base_file_name parameter is `123_main_street` and the extension is `.jpg`, the resulting file name is `123_main_street.jpg`.

p_extension
String to be used as the file extension in the final part of the URL. For example, if the value of the p_base_file_name parameter is `123_main_street` and the extension is `.jpg`, the resulting file name is `123_main_street.jpg`.

p_mimetype
MIME type of the object (for example: `image/jpeg`). If specified as `null`, the MIME type associated with the extension is used. If no MIME type is associated with the extension, `application/unknown` is used.

p_content_length
Length, in bytes, of the content to be exposed.

p_creation_date
Creation date of the file. If specified as null, SYSDATE (the current system date and time) is used.

p_dupe_behavior
Action to be taken if a call to the procedure would result in a duplicate file being added to the collection. For more information and a list of acceptable values, see Section 3.3.

p_docid
The internal document ID generated by OraDAV.

Usage Notes

Before using this procedure, be sure you understand the concepts, guidelines, and examples in Section 3.1.

For information about DAV methods that are allowed and disallowed on exposed data, see Table 3–1.

To remove the visibility of a resource that had been exposed by this procedure, use the Unexpose_Resource_By_Rowid procedure.

Examples

The following example exposes the object in the HOUSE_DOC column, where the value of HOUSE_ID is `1`. This example uses the same set of assumptions as those that are used in Section 3.1.

```
DECLARE
   ldocid INTEGER;
   loblen INTEGER;
   lrowid UROWID;
BEGIN
SELECT rowid,DBMS_LOB.GETLENGTH(HOUSE_DOC)
           INTO lrowid, loblen
```

```
          FROM REALTOR.HOUSES
          WHERE HOUSE_ID = 1;
ORDSYS.DAV_PUBLIC.expose_resource_by_rowid (
   ORDSYS.DAV_PUBLIC.generate_ctx('mywebsite'),
   'REALTOR',
   'HOUSES',
   'HOUSE_DOC',
   'BLOB',
   lrowid,
   '/external_collection',
   'house',
   '.html',
   'text/html',
   loblen,
   SYSDATE,
   ORDSYS.DAV_PUBLIC.DUPE_MODE_FAIL,
   ldocid);
COMMIT;
END;
/
```

Upon successful execution of the code in this example, the following URL returns the Web page from REATOR.HOUSES where the value of HOUSE_ID is `1`:

```
http://mywebserver/oradav/external_collection/house.html
```

Generate_Ctx

Format

```
ORDSYS.DAV_PUBLIC.Generate_Ctx(
    p_container_name  IN VARCHAR2,
    ) RETURN VARCHAR2;
```

Description

Returns an XML string with minimal context information for the specified container.

Parameters

p_container_name
Name of the container for which to generate minimal context information.

Usage Notes

This function can be used to generate a minimal context string for input to procedures that expose data.

For more information about container context, including examples of the XML string, see Section 3.4.

Examples

The following example returns the container context information. (In the actual output, the entire XML tag is on one line, with no space or line break.)

```
SELECT ORDSYS.DAV_PUBLIC.Generate_Ctx('mywebsite') FROM DUAL;

ORDSYS.DAV_PUBLIC.GENERATE_CTX('MYWEBSITE')
--------------------------------------------------------------------------------

<ORADAV><DAVPARAM><ORACONTAINERNAME>mywebsite</ORACONTAINERNAME><ORAEXCEPTION>
    RAISE</ORAEXCEPTION></DAVPARAM></ORADAV>
```

Unexpose_Column

Format

```
ORDSYS.DAV_PUBLIC.Unexpose_Column(
    p_ctx              IN  VARCHAR2,
    p_schema_name      IN  VARCHAR2,
    p_table_name       IN  VARCHAR2,
    p_media_column_name IN  VARCHAR2);
```

Description

Removes the visibility of objects in a column by reversing the effect of the Expose_Blob_Column or Expose_OracleMultimedia_Column procedure.

Parameters

p_ctx
Context string for the container on which to operate. (See Section 3.4 for detailed information about the context.)

p_schema_name
Name of the schema containing the table with the column to be unexposed.

p_table_name
Name of the table or view containing the column to be unexposed. Must not be one of the container system tables or views described in Section 2.2.

p_media_column_name
Name of the column containing the objects to be unexposed.

Usage Notes

Before using this procedure, be sure you understand the concepts, guidelines, and examples in Section 3.1.

Examples

The following example unexposes the images in the HOUSE_PIX column. This example uses the same set of assumptions as those that are used in Section 3.1.

```
DECLARE
BEGIN
ORDSYS.DAV_PUBLIC.Unexpose_Column(
   ORDSYS.DAV_PUBLIC.Generate_Ctx('mywebsite'),
   'REALTOR',
   'HOUSES',
   'HOUSE_PIX');

COMMIT;
END;
/
```

Upon successful execution of the code in this example, the images from REALTOR.HOUSES that were previously exposed are no longer available through the URL that was assigned.

Unexpose_Resource_By_Rowid

Format

```
ORDSYS.DAV_PUBLIC.Unexpose_Resource_By_Rowid(
    p_ctx          IN  VARCHAR2,
    p_schema_name  IN  VARCHAR2,
    p_table_name   IN  VARCHAR2,
    p_column_name  IN  VARCHAR2,
    p_rowid        IN  UROWID);
```

Description

Removes the visibility of a single object (associated with a specified ROWID) by reversing the effect of the Expose_Resource_By_Rowid procedure.

Parameters

p_ctx
Context string for the container on which to operate. (See Section 3.4 for detailed information about the context.)

p_schema_name
Name of the schema containing the table with the column to be unexposed.

p_table_name
Name of the table or view containing the object to be unexposed. Must not be one of the container system tables or views described in Section 2.2.

p_column_name
Name of the column containing the object to be unexposed.

p_rowid
The ROWID of the row in the p_table_name parameter containing the object to be unexposed.

Usage Notes

Before using this procedure, be sure you understand the concepts, guidelines, and examples in Section 3.1.

Examples

The following example unexposes the description in the HOUSE_DOC column, where the value of HOUSE_ID value is `1`. This example uses the same set of assumptions as those that are used in Section 3.1.

```
DECLARE
   v_rowid UROWID;
BEGIN
SELECT ROWID, INTO v_rowid FROM REALTOR.HOUSES
   WHERE HOUSE_ID = '1';

ORDSYS.DAV_PUBLIC.Unexpose_Resource_By_Rowid(
```

```
   ORDSYS.DAV_PUBLIC.Generate_Ctx('mywebsite'),
   'REALTOR',
   'HOUSES',
   'HOUSE_DOC',
   v_rowid);

COMMIT;
END;
/
```

Upon successful execution of the code in this example, the description from REALTOR.HOUSES with HOUSE_ID = '1' that was previously exposed is no longer available through the URL that was assigned.

Version

Format

ORDSYS.DAV_PUBLIC.Version() RETURN VARCHAR2;

Description

Returns the release (version) number of the DAV_PUBLIC package.

Parameters

None.

Usage Notes

Contrast this function with the Cversion function, which returns the container (physical) version number.

Examples

The following example returns the DAV_PUBLIC package version number.

```
SELECT ORDSYS.DAV_PUBLIC.Version FROM DUAL;

VERSION
--------------------------------------------------------------------------------

1.0.3.3.0-0019
```

Index

R

S

T

U

V

W