Open Document Management API

Version 2.0

Revision History

Version 1.0 - with small BHC-325 at the end of the document

Brad Clements, SoftSolutions. This is the original version.

Version 1.0 - with small DM1\285 at the end of the document

Mike Gardiner, Novell. This contains additional information for registering ODMA under Windows 95 and NT. It also adds a new table for document ID constants.

Version 1.0a

Rod Schiffman, Novell. Additional information on the ODMA spec background, usage of calls, suggestions on the use of document ID character sets and the addition of ODM_E_INUSE as a valid return code in ODMActivate.

Version 1.5

Jerry (Gerald) Willett, PC DOCS, Inc. Added the Query specification. This included the Query Syntax and Query Examples sections and the ODMQuery* functions and the IODMQuery COM interface. Also added a clarification to the expected behavior of an application when calling ODMSaveAs and an empty string is returned for lpszNewDocId.

Version 2.0

Bob St.Jean, Digital Equipment Corp. Added additional attributes and functions to improve ODMA support for popular Document Management System (DMS) features. These are mostly defined in a new IODMDocMan2 interface. Defined several items to help provide better out-of-the-box integration between desktop applications and DMSs. Also added clarifications to the existing specification and arranged the functions alphabetically.

Overview

The original impetus for the Open Document Management API (ODMA) was the recognition that there was no standard method for a client application to integrate with a Document Management System (DMS). Each DMS vendor wrote separate integration code for each of the major client applications they supported. Applications that did not have integrations written for them by the DMS vendors would have to write and support a separate integration for each DMS that was supported. This required a complete matrix of integrations, each with its own set of bugs, limitations and reliability issues. It seemed obvious that a high level standard for connecting applications and document management systems was a natural fit.

A small group of application and DMS vendors started working together to create an API that would allow applications and document management systems to inter-operate through a single high level API. This implied the creation of a standard, however, the creation of a standard has many pitfalls. Probably the biggest problem is that a lot of work can be put into the creation of the standard, and nobody uses it.

The industry is filled with examples of standards that were obsolete by the time they made it through the standardization process. By the time some standards make it through the standardization process, they are so large and unwieldy they are almost impossible to implement and maintain. Company politics and hidden agendas of the participants can play as big a role in the adoption of a standard as trying to solve the problem in the first place. The industry is also full of proprietary API's that claim to be standards, but are not. The initial group of vendors that met and formed the ODMA consortium wanted to avoid as many of these problems as possible.

The working rules of the ODMA consortium are fairly simple.

1. If the standard does not solve a problem it will not be used.
2. If the creation of the standard takes a long period of time, it does not solve the problem.
3. If the standard is difficult to implement, it does not solve the problem.
4. The standard must be vendor independent.
5. The standard must not try to solve all vendors' problems, or it will be big, complex and take a long time to implement. This violates rules 1, 2 and 3 above.
6. It is the customers that lose if there is not a straightforward way to integrate applications that create documents and applications that manage documents.
7. Easy integration between applications and document management systems will grow the industry and increase sales for the entire marketplace.

It is difficult to express the importance the initial members of the consortium placed on wanting to create a useful API that is vendor and platform independent while still simple to implement. They recognized that they could solve 80 percent of the problem easily and were willing to live with having to solve another 10 percent over time and probably never being able to solve the final 10 percent.

The Open Document Management API (ODMA) is a standardized, high-level interface between desktop applications and document management systems (DMSs). Its purposes are:

1. To make DMS services available to users of desktop applications in a seamless manner so that these services appear to the user like extensions of the applications.

2. To reduce the application vendors' burden of having to deal with multiple DMS vendors. By writing to ODMA, an application vendor has potentially integrated his application with all supporting DMSs.

3. To reduce the DMS vendors' burden of integrating with multiple applications. By supporting ODMA a DMS vendor has potentially integrated with all applications that have written to ODMA.

4. To reduce effort and complexity needed to install and maintain DMSs.

ODMA specifies a set of interfaces that applications can use to initiate actions within a DMS. The API is intended to be relatively easy for application vendors to incorporate into updates of existing applications. It should not require major restructuring of an application to integrate it with ODMA. Note that this version of ODMA does not specify how DMSs may initiate actions within the applications.

The ODMA API is platform-independent. The associated data type definitions and binding information are platform-specific. Currently, most of the work has been done in Windows. It makes this document look Windows specific, but over time, the platform specific entries for other platforms will be added as they are defined.

Document IDs

Many of the ODMA functions accept or return a Document ID parameter. A document ID is a persistent, portable identifier for a document. It can be stored and used in a later session, and it can be passed across platforms via email or other processes.

A Document ID is a case insensitive, null-terminated string of printable characters. Although a document ID is case insensitive, an application should never change the case of a document ID. The format of a document ID is

::ODMA\DMS_ID\DM_SPECIFIC_INFO

The DMS_ID portion of a document ID will identify which DMS provided the ID. This information is primarily for the use of ODMA itself; applications using ODMA should not need to know which DMS provided a particular ID. The ODMA group members will coordinate these IDs to ensure their uniqueness. The maximum length of the DMS_ID portion of the document ID is specified by the constant **ODM_DMSID_MAX.** The DM_SPECIFIC_INFO portion of the ID will vary depending on which DMS built the ID. The total length of the document ID including the terminating Null character cannot exceed **ODM_DOCID_MAX** bytes.

ODMA-aware applications should be able to handle a document ID anywhere they handle an externally-generated document filename. For example, if the application allows a document filename to be passed as a command line argument then it should allow a document ID to be passed in the same way. If the application allows document filenames to be used in DDE commands then it should also support the use of document IDs in the same commands.

Although the technical definition of a document ID is a *case insensitive, null-terminated strings of printable characters*, there are some general rules that are more likely to make a DMS and ODMA application work better together. ODMA was designed so that it would be easy to add to an application without major modifications in code or structure. If a DMS passes a document ID that breaks fundamental rules of normal file and path names it will probably run into problems if it is passed in on a command line. Special characters like ^, [,], |, *, -, >, < and ? are processed by the UNIX shell even before they are seen by the application. It is possible to pass these characters by using special escape sequences, but that places a burden on the DMS vendor to process the document ID before giving it to the ODMA application. Some operating systems require the application to handle the reverse process of interpreting and removing the escape characters. The application may be able to support the escape removal on the command line, but not if the document ID with escape characters is returned in a procedure call. In most cases, it is easier to generate a document ID that contains a fairly simple set of characters. The following table suggests characters it may be wise to avoid for different platforms.

Platform	**Characters to avoid**

Platform	**Characters to avoid**
Windows 3.x	" ‘ < > * ? | and the space character
Windows 95	" ‘ < > * ? |
Other platforms to be defined	

Constants

The following table lists the constants that are defined in the odma.h header.

CONSTANTS	**Win 3.x**	**Win32**	**Mac**	**UNIX**	**Other**
ODM_DOCID_MAX Maximum length of a document ID including the null-terminator.	80	255	255	255	255
ODM_FILENAME_MAX Maximum length of a path/filename returned by ODMA including the terminating Null character.	128	255	255	1024	255
ODM_API_VERSION The version of the ODMA API to which this header file corresponds. See **ODMRegisterApp**.	200	200	200	200	200
ODM_DMSID_MAX Maximum length of a DMS ID including the null-terminator.	9	9	9	9	9
ODM_APPID_MAX Maximum length of an Application ID including the null-terminator.	16	16	16	16	16
ODM_FORMAT_MAX Maximum length of a content format string including the null-terminator.	81	81	81	81	81
ODM_QUERYID_MAX Maximum length of a query ID string including the null-terminator.	255	255	255	255	255

Error Handling

Nearly all of the ODMA functions use the return value to indicate to the calling application whether the function succeeded, failed because the user canceled the operation, or failed for other reasons. The DMS is responsible for displaying informational error messages to the user where appropriate except when the **ODM_SILENT** flag is specified. The DMS must take care to return the appropriate error indication because applications may act differently depending on whether an ODMA call was canceled by the user or failed for other reasons. The calling application generally should not display error messages when an error value is returned from ODMA unless the **ODM_SILENT** flag was specified.

Connections and the ODMA Connection Manager

The ODMA connection manager is a small software module that sits between applications using ODMA and document management systems implementing ODMA. It manages the connections between these components and routes ODMA calls to the appropriate provider. A freely redistributable copy of the ODMA connection manager will be provided to any vendor wishing to implement or make use of the ODMA API. This is a place where it would be possible to provide mapping code that would allow ODMA to have truly platform independent document IDs, however, it currently only manages connections and does not touch document IDs.

Document Format Names

When new documents are registered with a DMS via ODMA and when an existing document's format is changed by an application, the application passes a document format name to ODMA. Document format names are null-terminated strings defining the format of a document's content. These strings have a maximum length defined by the **ODM_FORMAT_MAX** constant.

There are several places in ODMA where a content format name for a document can be specified or requested. The **ODMNewDoc**, **ODMSaveAs** and **ODMSaveAsEx** functions each have a *lpszFormat* parameter where a format can be specified. The **ODMSaveAs** also has a callback, which uses a format string. The **ODM_CONTENTFORMAT** attribute can be requested or set using the **ODMGetDocInfo** and **ODMSetDocInfo** functions.

The ODMA 1.0 specification did not attempt to standardize the format names used by DMS and application vendors. This was a significant limitation in some cases. The DMS would not know in advance what type of file was being created by the application. Also an application that saved an existing file back to the DMS using a different format would have no standard format name to give to the DMS. This model relied on both the DMS and the application being able to auto-detect all file formats.

ODMA 2.0 defines guidelines to standardize format names. This has become necessary due to vendors' desire to have applications and DMSs more tightly integrated and because ODMA 2.0 introduces some new functions which require standardization in order to work. The new **ODMGetAlternateContent** and **ODMSetAlternateContent** functions require the DMS and application to have a standard way to identify file content. A new document attribute called **ODM_ALTERNATE_RENDERINGS** allows an application to find out if a DMS can provide a document in other formats and make a choice of which to use.

ODMA has defined the following guidelines to standardize how a DMS or application might specify a document's content format:

1. MIME Content Type. If a file type does not have a MIME content type, then a file extension must be used.
2. File Extension (a.k.a. File Type). If a file extension is used the first character must be a dot. The file extension should be the one normally used to identify the document on the platform where the client application is running. Some platforms allow the file extension to contain more than three characters. ODMA does not define a maximum file extension length.
3. File Extension plus other identifying information. This other information may be useful in cases where a single file extension is used for different variations or versions of the file type. In this format the first character must be a dot, followed by the file extension, plus a single forward slash, plus a string containing the additional information.

Examples:

```
application/msword
.doc
.doc/MS Word 95 Document
.doc/MS Word 97 Document

image/tiff
.TIF
.TIFF
.TIF/Tagged Image File Format
.TIF/Scanned TIFF
```

A DMS or application can use the Windows Registry or its own mapping capability to convert a file extension to a MIME code or vice versa. Note that some file extensions are used by more than one application, some file types have multiple valid file extensions, and that some MIME content types are used to define more than one file type within the same application. A DMS or application may have use whatever format name is most precise.

DMS or Native File System Dialogs - Which to Display First

ODMA imposes no rules on applications requiring them to always show the DMS dialogs first in response to filing commands such as File Open and File Save As. ODMA applications are encouraged to provide a user preference setting so the user can designate if the application's native file system dialog or the DMS's dialog is displayed first. Ideally the application may provide a push-button or some other control in its native file system dialog box to allow the user to quickly select the DMS dialog box.

If the application cannot provide one or more of the options described above then they have no choice but to always display the DMS dialog box in response to the File Open and File Save As commands. Each ODMA compliant DMS must provide a way for the user to select the native file system from dialogs displayed while processing the **ODMSelectDoc**, **ODMSelectDocEx**, **ODMNewDoc**, **ODMSaveAs** and **ODMSaveAsEx** functions. The DMS returns the **ODM_E_APPSELECT** status if the user elects to use the native file system. When this status is returned the application can display their dialog box. If the user wishes to return to the DMS dialog box they will have to cancel and re-do the file command.

Character Sets

All strings passed to or returned from ODMA functions should be null-terminated series of octets, in the standard character set of the system on where ODMA is being used. So for example, 8859-1 would be used on English Windows, Shift-JIS would be used on Japanese Windows, etc. The term "null-terminated" as used in this specification means terminated by the character set's natural Null character. For most character sets this means a single byte with the value 0x0.

If an application obtains an ODMA document ID on one platform and later uses it on another platform, the application is responsible for translating the ID to the character set being used on the second platform before using it there.

Application Interfaces

An ODMA-aware application can choose to communicate with the ODMA Connection Manager either through a traditional, function-oriented API or through Component Object Model (COM) interfaces. Prototypes and constants used for the function-oriented API are included in the odma.h header file.

Prototypes, constants, and interface definitions for the COM interface are included in the odmacom.h header file.

After calling **ODMRegisterApp,** applications can obtain one or more COM interfaces to ODMA through the **ODMQueryInterface** function. The **IODMDocMan** interface provides an alternate entry point to most of the ODMA functions documented below. Other functions are defined in the **IODMDocMan2** and **IODMQuery** interfaces. These interfaces and their interface IDs are defined in the odmacom.h header file.

Note that **IODMDocMan::QueryInterface** will only query the default DMS for the calling application. The application must call **ODMQueryInterface** in order to query other DMSs.

Query Syntax

The query syntax described here is used in the **ODMQueryExecute** function.

```
<query>             :: select <returned_list> <search_criteria>

<returned_list>     :: <returned_item>[, <returned_list>]

<returned_item>     :: ODM_DOCID
                    :: ODM_NAME

<search_criteria>   :: <search_clause>
                    :: <where_clause>
                    :: <search_clause> <where_clause>
                    :: <where_clause> <search_clause>

<search_clause>     :: search document contains <expression>

<expressioin>       :: (<expression>)
                    :: [not] (<expression>)
                    :: <term> [<operator> <expression>]

<term>              :: ‘<word>’
                    :: [not] ‘<word>’

<operator>          :: and
                    :: or
```

<word> is a user supplied word. If there is a single quote (’) inside this word, it needs to be represented with two consecutive single quotes (’’).

```
<where_clause>      :: where <condition_list>

<condition_list>    :: <condition> [<operator> <condition_list>]

<condition>         :: <attribute> <op> ‘<attribute_value>’

<attribute>         :: ODM_FORMAT
                    :: ODM_NAME
                    :: ODM_AUTHOR
                    :: ODM_TYPE
                    :: ODM_DMS_DEFINED:<dms_attribute>
                    :: * Any other document attribute listed in Appendix A

<op>                :: =
                    :: !=
                    :: <>
                    :: >
                    :: <
                    :: <=
                    :: >=
```

<dms_attribute> is a DMS specific attribute name.

<attribute_value> is a user supplied value for ODM_NAME, etc. If there is a single quote (’) inside any of these names, it must be represented with two consecutive single quotes (’’).

Query Examples

```
Example 1:   select ODM_DOCID, ODM_NAME
             where ODM_TYPE = ‘Memo’
             search document contains ‘Internet’ and not ‘Intranet’

Example 2:   select ODM_DOCID, ODM_NAME
             where ODM_AUTHOR = ‘Mark Twain’ and ODM_TYPE = ‘Story’

Example 3:   select ODM_DOCID, ODM_NAME
             search document contains ‘Mary’’s Lamb’
             where ODM_NAME = ‘Mary Has a Little Lamb’ and
             ODM_TYPE = ‘Song’
```

```
Example 4:  select ODM_DOCID, ODM_NAME
            search document contains (‘rabbit’ or ‘hare’) and not ‘bunny’
            where ODM_NAME = ‘Wild Life’ and ODM_AUTHOR = ‘John Doe’
            and ODM_TYPE = ‘Letter’
```

ODMA API

The following section describes each function in the ODMA API that a typical application would use. The functions are listed in alphabetical order.

ODMActivate

ODMSTATUS ODMActivate (ODMHANDLE handle, WORD action,
LPSTR lpszDocId)

This function causes the DMS to perform actions that do not require cooperation from the calling application. Control is returned to the calling application after the specified action has been completed, except where noted. A DMS is not required to support all of these actions.

Parameters:

handle - in - A handle obtained by a previous **ODMRegisterApp** call.

action - in - One of the following action codes:

ODM_NONE - No specific action is requested. The DMS should simply make itself visible and let the user select the action to be performed.

ODM_DELETE - The DMS should delete the specified document. Note that most DMSs will not allow a deletion to occur if the document is currently in use.

ODM_SHOWATTRIBUTES - The DMS should display the specified document's profile or attributes.

ODM_EDITATTRIBUTES - The DMS should display the specified document's profile or attributes, and the user should be put in edit mode. Note that some DMSs will not allow a document's attributes to be edited while it is in use.

ODM_VIEWDOC - The DMS should display the specified document in a viewer window. The DMS may return control to the calling application before displaying the document.

ODM_OPENDOC - The DMS should open the specified document in its native application. The DMS may return control to the calling application before displaying the document. This function is intended for use by applications other than the document's native application (e-mail, workflow, annotation, etc.). Applications should use **ODMOpenDoc** to access their own documents. If this function fails, the calling application may wish to retry using the **ODM_VIEWDOC** action code.

ODM_NEWDOC - The DMS should allow the user to create and save a new document. Optionally the caller can specify a template document in *lpszDocId*. If *lpszDocId* is Null the DMS should allow the user to choose the file format of the document to be created and the document template. In most cases it is expected that the DMS will need to launch an application to create the document. The DMS may return before satisfying this call, in which case the calling application will not get notification when the new document has been created. The user is free to cancel this function.

ODM_CHECKOUT - The DMS should check-out/reserve the document for the current user. The DMS can display whatever UI it might use for document check-out. This action allows the user to explicitly reserve a document in the DMS in a way that is persistent across ODMA or DMS sessions. **ODMOpenDoc** should be used to get the content file. See **ODMCloseDoc** and **ODMCloseDocEx** for recommendations on how to handle closing a document that was explicitly reserved before it was opened.

ODM_CANCELCHECKOUT - The DMS should cancel a previous checkout/reserve on the document, if it has been checked-out by the current user. The DMS can display whatever UI it might use for canceling a document check-out.

ODM_CHECKIN - The DMS should check-in/unreserve the document if it's checked-out by the current user. The DMS can display whatever UI it might use for document check-in. **ODMSaveDoc** or **ODMSaveDocEx** should already have been used, if necessary, to save the content to the DMS.

ODM_SHOWHISTORY - The DMS should display the specified document's history (i.e. revisions, events, activities, etc.).

lpszDocId - in - A document ID specifying the document on which to perform the requested action. This parameter may be Null if the *action* is **ODM_NONE** or **ODM_NEWDOC**. If *action* is **ODM_NEWDOC** the application can specify the document ID of a template document.

Return value:

ODM_SUCCESS if successful.

ODM_E_DOCID if the document ID is invalid or refers to a document that no longer exists.

ODM_E_INUSE if the document is currently in use or checked-out by another user on actions where this would preclude the operation from completing correctly.

ODM_E_ACCESS if the user doesn't have appropriate access rights to perform the requested action (i.e. check-out or check-in the document).

ODM_E_OFFLINE if the DMS cannot currently access the document because the user client is off-line.

ODM_E_ARCHIVED if the DMS cannot currently supply the document content because it has been archived.

ODM_E_CANCEL if the action was canceled by the user.

ODM_E_NOSUPPORT if *action* is not supported by the DMS.

ODM_E_ITEM if *action* is invalid or not supported by the DMS.

ODM_E_FAIL if the action could not be completed by the DMS.

ODM_W_NOACTION if *action* is **ODM_CHECKOUT** and the document is already checked-out/reserved by the current user. This status can also be returned if *action* is either **ODM_CANCELCHECKOUT** or **ODM_CHECKIN** and the document is not currently checked-out/reserved by any user.

ODM_E_HANDLE if *handle* was invalid.

ODMCloseDoc

ODMSTATUS ODMCloseDoc(ODMHANDLE handle, LPSTR lpszDocId,
DWORD activeTime, DWORD pagesPrinted, LPVOID sessionData,
WORD dataLen)

An application that has opened a document by calling **ODMOpenDoc** must call **ODMCloseDoc** when it is finished using the document. The application should not call this function until after it has closed the document, because the DMS may move the document or make it inaccessible as a result of this call. Note that this function will not cause the document to be saved into the DMS's persistent repository unless **ODMSaveDoc** has been called previously.

It is possible for a user to explicitly check-out/reserve a document using either **ODMActivate** or a command provided by the DMS. If a document was already reserved by the user before **ODMOpenDoc** was called, then when **ODMCloseDoc** is called it is recommended that the DMS keep the document reserved. The DMS should only check-in/unreserve the document in **ODMCloseDoc** if it first displays a dialog box confirming this with the user.

Parameters:

handle - in - A handle obtained by a previous **ODMRegisterApp** call.

lpszDocId - in - A null-terminated document ID. This is typically obtained by a call to **ODMSelectDoc** or **ODMNewDoc**. This document must have been previously opened by a call to **ODMOpenDoc**.

activeTime - in - If the application tracks time spent editing the document then it should pass the number of seconds here. Otherwise it should pass 0xFFFFFFFF.

pagesPrinted - in - If the application tracks the number of pages printed from this document during the current editing session; it should pass this number here. Otherwise it should pass 0xFFFFFFFF.

sessionData - in - The application may pass other information regarding the current editing session in this parameter. For example, an application might pass the number of keystrokes that were entered. The calling application is free to determine the format of this data, so DMSs that rely on this information will have to coordinate with each application supported. Null should be passed if the application has no meaningful information to pass through this parameter.

dataLen - in - The length of the data passed in the *sessionData* parameter. Ignored if *sessionData* is Null.

Return value:

ODM_SUCCESS if successful.
ODM_E_NOOPEN if the document was not previously opened.
ODM_E_FILELOCKED if the DMS could not complete the function because the temporary file provided in **ODMOpenDoc** is still opened by the calling application.
ODM_E_FAIL if the DMS is unable to close the document for any other reason.
ODM_E_HANDLE if *handle* was invalid.

ODMCloseDocEx

ODMSTATUS ODMCloseDocEx(ODMHANDLE handle, LPSTR lpszDocId, LPDWORD pdwFlags, DWORD activeTime, DWORD pagesPrinted, LPVOID sessionData, WORD dataLen)

ODMCloseDocEx is the same as **ODMCloseDoc** except it has a *pwdFlags* parameter. There is an **ODM_SILENT** flag to facilitate unattended document processing. Some DMSs display a user interface when closing a document. This flag allows the calling application to suppress this UI.

An application that has opened a document by calling **ODMOpenDoc** must call **ODMCloseDoc** or **ODMCloseDocEx** when it is finished using the document. The application should not call this function until after it has closed the document, because the DMS may move the document or make it inaccessible as a result of this call. Note that this function will not cause the document to be saved into the DMS's persistent repository unless **ODMSaveDoc** or **ODMSaveDocEx** has been called previously.

It is possible for a user to explicitly check-out/reserve a document using either **ODMActivate** or a command provided by the DMS. If a document was already reserved by the user before **ODMOpenDoc** was called, then when **ODMCloseDocEx** is called it is recommended that the DMS keep the document reserved. The DMS should only check-in/unreserve the document in **ODMCloseDocEx** if the **ODM_SILENT** flag is not set and it first displays a dialog box confirming this with the user.

Parameters:

handle - in - A handle obtained by a previous **ODMRegisterApp** call.

lpszDocId - in - A null-terminated document ID. This is typically obtained by a call to **ODMSelectDoc**, **ODMSelectDocEx** or **ODMNewDoc**. This document must have been previously opened by a call to **ODMOpenDoc**.

pdwFlags - in/out - A pointer to a variable containing flags used on both input and output. On input, 0 or the following flag:

ODM_SILENT - The DMS should not require user interaction while satisfying the call. If the call cannot be satisfied without user interaction then an error should be returned.

ODMA 2.0 does not define any output flags at this time.

activeTime - in - If the application tracks time spent editing the document then it should pass the number of seconds here. Otherwise it should pass 0xFFFFFFFF.

pagesPrinted - in - If the application tracks the number of pages printed from this document during the current editing session; it should pass this number here. Otherwise it should pass 0xFFFFFFFF.

sessionData - in - The application may pass other information regarding the current editing session in this parameter. For example, an application might pass the number of keystrokes that were entered. The calling application is free to determine the format of this data, so DMSs that rely on this information will have to coordinate with each application supported. Null should be passed if the application has no meaningful information to pass through this parameter.

dataLen - in - The length of the data passed in the *sessionData* parameter. Ignored if *sessionData* is Null.

Return value:

ODM_SUCCESS if successful.
ODM_E_NOOPEN if the document was not previously opened.
ODM_E_USERINT if the **ODM_SILENT** flag was specified and the DMS could not satisfy the call without user interaction.
ODM_E_FILELOCKED if the DMS could not complete the function because the temporary file provided in **ODMOpenDoc** is still opened by the calling application.
ODM_E_FAIL if the DMS is unable to close the document for any other reason.
ODM_E_HANDLE if *handle* was invalid.

ODMGetAlternateContent

ODMSTATUS ODMGetAlternateContent(ODMHANDLE handle, LPSTR lpszDocId, LPDWORD pdwFlags, LPSTR lpszFormat, LPSTR lpszDocLocation)

This function causes the DMS to return an alternate content file for the specified document. The format of the alternate content file should be one of the formats returned by the **ODM_ALTERNATE_RENDERINGS** item in a previous call to **ODMGetDocInfo**. The application is responsible for deleting the alternate content file when it is finished using it.

Parameters:

handle - in - A handle obtained by a previous **ODMRegisterApp** call.

lpszDocId - in - A null-terminated document ID. This is typically obtained by a call to **ODMSelectDoc**, **ODMSelectDocEx** or **ODMNewDoc**. The specified document may or may not be open when this function is called. This document ID refers to the main document for which an alternate content file is being requested.

pdwFlags - in/out - A pointer to a variable containing flags used on both input and output. On input, zero (0) or the following flag:

ODM_SILENT - The DMS should not require user interaction while satisfying the call. If the call cannot be satisfied without user interaction then an error should be returned.

ODMA 2.0 does not define any output flags at this time.

lpszFormat - in - A null-terminated string containing the format name of the alternate format requested. The Document Format Names section has information on how file formats are identified in ODMA. Typically the calling application would have first obtained the alternate renderings the DMS has for the specified document then specified one in this parameter. This is done by requesting the **ODM_ALTERNATE_RENDERINGS** attribute in **ODMGetDocInfo**. However, the application is free to simply specify the MIME Content Type or File Extension for the format it needs. If the DMS cannot return the requested format, it will return the **ODM_E_NOSUPPORT** status.

lpszDocLocation - in/out - A pointer to a buffer of at least **ODM_FILENAME_MAX** bytes in length, into which the DMS will write a null-terminated string containing a full path and file name of the alternate content file. If an error occurs then the contents of the buffer will be undefined. It is the responsibility of the calling application, not the DMS, to delete this file when it is finished with it.

Optionally the calling application can specify a file extension in this parameter for the DMS to use in the file specification it returns. If specified, the file extension should be preceded by a period. The DMS may choose to ignore this information and return a file specification containing a different file extension.

Return value:

ODM_SUCCESS if successful.
ODM_E_ACCESS if the user does not have access rights to the requested alternate content file or the main document.
ODM_E_INUSE if the user is currently unable to access the alternate content file because the main document is checked-out by another user.
ODM_E_DOCID if the document ID is invalid or refers to a document that no longer exists.
ODM_E_OFFLINE if the DMS cannot currently access the document because the user client is off-line.
ODM_E_ARCHIVED if the DMS cannot currently supply the document content because it has been archived.
ODM_E_USERINT if the **ODM_SILENT** flag was specified and the DMS could not make the specified document available without user interaction.
ODM_E_INVARG if the value in *flags* is invalid.
ODM_E_REQARG if a required parameter isn't specified (i.e. *lpszFormat* or *lpszDocLocation*).
ODM_E_NOSUPPORT if the DMS does not support the function or it cannot return the requested alternate content format for the specified document.
ODM_E_FAIL if the DMS is unable to make the document accessible for any other reason.
ODM_E_HANDLE if *handle* was invalid.

ODMGetDMS

WORD ODMGetDMS(LPCSTR lpszAppId, LPSTR lpszDmsId)

This function provides an application a programmatic way to get its default DMS. The DMS ID of the current DMS for the application will be returned. This can be either the default DMS from the registry or a DMS previously set by the application using **ODMSetDMS**.

Parameters:

> *lpszAppId* - in - A pointer to a null-terminated string containing an Application ID.
>
> *lpszDmsId* - out - A pointer to a buffer to receive the DMS ID of the default DMS. This buffer must be at least **ODM_DMSID_MAX** bytes in length. If an error occurs the value in this buffer is undefined.

Returns value:

> **ODM_SUCCESS** if successful.
> **ODM_E_NODMS** if there is no default DMS registered.
> **ODM_E_REQARG** if either parameter is not specified.

ODMGetDMSCount

WORD ODMGetDMSCount( )

This is an informational function. It will count the number of DMSs currently registered on the system. This information can be used to determine the minimum size for the buffer in a call to **ODMGetDMSList**.

Parameters: None.

Return value: The number of DMSs currently registered on the system.

ODMGetDMSInfo

ODMSTATUS ODMGetDMSInfo(ODMHANDLE handle, LPSTR lpszDmsId, LPWORD pwVerNo, LPDWORD pdwExtensions)

This function returns information to the application about the currently active DMS. The application if free to only request the information it needs.

Parameters:

handle - in - A handle obtained by a previous **ODMRegisterApp** call.

lpszDmsId - out - If specified by the caller, this should be a pointer to a buffer of at least **ODM_DMSID_MAX** bytes in length. A null-terminated ID identifying the DMS is returned here. This is the same ID embedded in document IDs returned by this DMS.

pwVerNo - out - If specified by the caller, this should be a pointer to a variable to receive a version number. The version of the ODMA API supported by this DMS is returned here.

pdwExtensions - out - If specified by the caller, this should be a pointer to a variable to receive extension information. Indications of extensions to the base ODMA API that are supported by this DMS are returned here. If a DMS does not support any currently defined extensions, then 0 will be returned. The DMS will return flag values for all the extensions it supports. The currently defined ODMA extensions are:

ODM_EXT_WORKFLOW - The DMS supports the ODMA Workflow Extensions.
ODM_EXT_QUERY - The DMS supports the Query Extensions defined in ODMA 1.5.

Return value:

ODM_SUCCESS if successful.
ODM_E_HANDLE if *handle* was invalid.

ODMGetDMSList

WORD ODMGetDMSList(LPSTR buffer, WORD buffer_size)

This function gets a list of the DMSs currently registered on the system.

Parameters:

> *buffer* - out - A pointer to a buffer which will receive a list of DMSs. The format of the list is a collection of Null terminated strings. The list is terminated by an empty string (i.e. "<id#1>\0<id#2>\0\0").
>
> *buffer_size* - in - Size of *buffer*. It must be at least **ODMGetDMSCount()** * **ODM_DMSID_MAX** + 1 bytes in length.

Return value: The number of DMSs returned in *buffer.*

ODMGetDocInfo

ODMSTATUS ODMGetDocInfo(ODMHANDLE handle, LPSTR lpszDocId, WORD item, LPSTR lpszData, WORD dataLen)

An application can use this function to obtain information about a document from the DMS. It is recommended that the DMS not display any user interface while processing this function.

Parameters:

handle - in - A handle obtained by a previous **ODMRegisterApp** call.

lpszDocId - in - A null-terminated document ID. This is typically obtained by a call to **ODMSelectDoc**, **ODMSelectDocEx** or **ODMNewDoc**. The specified document may or may not be open when **ODMGetDocInfo** is called.

item - in - A single Item Id.

Refer to Appendix A for a complete list of Item Ids for document attributes which can be specified in this parameter.

ODM_DMS_DEFINED - Can be used for DMS specific attributes not explicitly defined by ODMA. The *lpszData* parameter contains a DMS-specific indication of the data to be returned. Note that an application must know which DMS it is talking to and must understand the data indications supported by the DMS in order to use this item id.

lpszData - in/out - On input, ignored if *item* is anything other than **ODM_DMS_DEFINED**. If *item* is **ODM_DMS_DEFINED** then *lpszData* contains an indication of the data to be returned. It is recommended that the DMS be able to recognize a null-terminated string containing one of its known attribute names. On output, the requested data is returned in the buffer pointed to by *lpszData*. The data is always null-terminated.

dataLen - in - The length of the output buffer pointed to by *lpszData*. If the data to be returned is longer than *dataLen* it will be truncated. The DMS must return **ODM_E_TRUNCATED** when the requested data cannot be safely truncated; for example when an attribute containing a document ID or URL is returned. In these cases the application should recall the function supplying a larger buffer.

Return value:

ODM_SUCCESS if successful.
ODM_E_DOCID if the document ID is invalid or refers to a document that no longer exists.
ODM_E_OFFLINE if the DMS cannot currently access the document because the user client is off-line.
ODM_E_TRUNCATED if the application supplied buffer is too small to hold data that cannot be safely truncated. A DMS cannot truncate an **ODM_URL** attribute or any attribute containing an ODMA document ID.
ODM_E_ITEM if *item* is invalid or unknown to this DMS.
ODM_E_NOSUPPORT if the DMS does not support the function or does not support the specified document attribute.
ODM_E_HANDLE if *handle* was invalid.

ODMGetDocRelation

ODMSTATUS ODMGetDocRelation(ODMHANDLE handle, LPSTR lpszDocId, LPDWORD pdwFlags, LPSTR lpszLinkedId, LPSTR lpszFormat, LPSTR lpszPreviousId)

An application can use this function to retrieve pointers to document versions linked to a particular document ID. This function would be used typically as part of retrieving a compound document. It could also be used when saving an updated component document, for the user to verify which compound documents will be impacted by any changes.

Parameters:

handle - in - A handle obtained by a previous **ODMRegisterApp** call.

lpszDocId - in - A null-terminated document ID. This is typically obtained by a call to **ODMSelectDoc** or **ODMNewDoc**. The specified document may or may not be open when **ODMGetDocRelation** is called.

pdwFlags - in/out - A pointer to a variable containing flags used on both input and output. On input, one or more of the following flags:

ODM_REL_PARENT - Return the *lpszLinkedId* that is a parent of *lpszDocId*.

ODM_REL_CHILD - Return the *lpszLinkedId* that is a child of *lpszDocId*.

ODM_SILENT - The DMS should not require user interaction while satisfying the call. If the call cannot be satisfied without user interaction then an error should be returned.

Upon return one or more of the following flags may be set by the DMS unless an error occurred:

ODM_REL_NOTORDERED - The default is that the DMS maintains child links as an ordered list, and that it will use *lpszPreviousId* to select the appropriate *lpszLinkedId* to return. If it is returning them in no particular, logical, meaningful sequence, the DMS will return this value to warn the application. A DMS could maintain both ordered and unordered lists, an example of the former being a book with separate chapters, while the latter might be a collection such as attached to a workflow.

ODM_REL_FIXED - The relationship is frozen between the particular document versions represented by *lpszDocId* and *lpszLinkedId*. This may also be used to signify that the application and/or the user want to retain control over updating the link, rather than have the DMS do it.

ODM_REL_RELEASED - The link should be maintained to the Current or Released version of the child document. Released is available in some DMS to distinguish a version that has been approved for release. If the DMS does not distinguish between Released and Latest, then Latest is used. The value is called Released and not Current, to avoid confusion with the version that is already (currently) linked.

ODM_REL_LATEST - The link should be maintained to the latest version of the child document.

lpszLinkedId - out - A pointer to a buffer where the DMS will return the ID of the next document version that is linked to *lpszDocId*. This buffer must be at least **ODM_DOCID_MAX** bytes in length. If successful then a null-terminated document ID will be returned here. Otherwise the contents of the buffer will be undefined.

lpszFormat - in/out - This is an optional parameter. If specified it must be a pointer to a buffer of at least **ODM_FORMAT_MAX** characters. This buffer contains a null-terminated string naming the file format of the rendition to be retrieved when the link is followed. Refer to the Document Format Names section for information on how file formats are identified in ODMA. This parameter allows the caller to identify the best format for the intended use of the main document. The requested format may or may not be the one established in **ODMSetDocRelation** and may or may not be the primary editing format. For instance, a graphic may be saved in several resolutions, one for on-line use and another for printing.

On input, this is a hint to the DMS of the preferred format of the child to be retrieved. If the application does not provide a format it should set the buffer in *lpszFormat* to a zero length string. In this case the DMS should use the format established in **ODMSetDocRelation** or the primary format of the document.

On output, the DMS should write the format name of the rendition returned. This may be the closest rendition format the DMS could find. If the DMS does not record rendition formats when relating documents, then it should return a zero length string in this buffer.

If this parameter is Null, then the DMS should use the format established in **ODMSetDocRelation** or the primary format of the child document (the one used in **ODMOpenDoc**).

lpszPreviousId - in - A null-terminated document ID. This is typically returned as *lpszLinkedId* on a previous call to **ODMGetDocRelation**. By making repeated calls to **ODMGetDocRelation**, the application obtains a logically ordered list of document ID's that make up a compound document. To get the first document in a list, provide a zero length, null-terminated document ID. Null should be passed if the application has no meaningful information to pass through this parameter.

Return value:

ODM_SUCCESS if successful.
ODM_E_NORELATION if the specified document has no related parent or child.
ODM_E_NOMOREDATA if the end of the list has been passed.
ODM_E_DOCID if a document ID is invalid or refers to a document that no longer exists.
ODM_E_USERINT if the **ODM_SILENT** flag was specified and the DMS could not satisfy the call without user interaction.
ODM_E_NOSUPPORT if the DMS does not support the function.
ODM_E_INVARG if the value specified in *pdwFlags* was invalid. On input either **ODM_REL_PARENT** or **ODM_REL_CHILD** must be set.
ODM_E_REQARG if a required parameter (i.e. *lpszLinkedId*) is Null.
ODM_E_HANDLE if *handle* was invalid.
ODM_E_FAIL if the specified data was invalid or the DMS was unable to accept it for other reasons.

ODMGetLeadMoniker

ODMSTATUS ODMGetLeadMoniker(ODMHANDLE handle,
LPSTR lpszDocId, LPMONIKER FAR *ppMoniker)

Applications that are OLE 2 servers typically form composite monikers for their OLE links by combining a file moniker representing the document with one or more item monikers representing a particular section of the document. This approach often does not work in environments where Document Management Systems are in use because the filename that the application sees is usually just temporary. This function lets the application obtain a leading moniker from the DMS that can be used in place of the file moniker.

This function will only be available on platforms supporting OLE 2. This function may not be supported by some DMSs; those DMSs will return **ODM_E_FAIL**. In this case the application should go ahead and use the file moniker as though ODMA were not present. Note that this function is prototyped in odmacom.h instead of odma.h, so that non-OLE-aware applications do not have to #include the OLE header files.

Parameters:

handle - in - A handle obtained by a previous **ODMRegisterApp** call.

lpszDocId - in - An ODMA document ID.

ppMoniker - out - A leading moniker for the specified document ID will be returned here if successful. Otherwise Null will be returned here.

Return value:

ODM_SUCCESS if successful.
ODM_E_FAIL if the DMS that created the specified document ID does not support OLE moniker building.
ODM_E_DOCID if the document ID is invalid or refers to a document that no longer exists.
ODM_E_HANDLE if *handle* is invalid.

ODMNewDoc

ODMSTATUS ODMNewDoc(ODMHANDLE handle, LPSTR lpszDocId, DWORD dwFlags, LPSTR lpszFormat, LPSTR lpszDocLocation)

This function causes the DMS to create a new document profile and return the document ID for the new document to the calling application. The DMS may create a temporary or pending document profile; it may not have actually created a document in its data store. If the DMS displays a dialog box for this function it should be task/application modal.

The document ID returned by the DMS in **ODMNewDoc** provides a document context for the application. This document ID must be used in subsequent ODMA function calls which are needed to complete the process of saving a new document into a DMS. If the user decides to cancel the operation or save the document to the native file system, then the application should call **ODMCloseDoc** or **ODMCloseDocEx** and throw the document ID away. The document ID returned by **ODMNewDoc** may be temporary so the application must be prepared for the possibility that the DMS will later override it in a call to **ODMSaveAs**, **ODMSaveAsEx** or **ODMSaveDoc** or **ODMSaveDocEx**.

The calling sequence for creating a new document, which isn’t based on another existing document, is as follows: **ODMNewDoc**, **ODMSaveAs**, **ODMOpenDoc** then **ODMSaveDoc**. Finally the document must be closed with **ODMCloseDoc**. The above sequence will likely involve user interaction with the DMS displaying one or more dialog boxes.

It is possible for an application to create a new document without any user interaction if the DMS supports it. To accomplish this the calling application should set the **ODM_SILENT** flag and use the following call sequence: **ODMNewDoc**, {**ODMSetDocInfo**}, **ODMSaveAsEx**, **ODMOpenDoc**, **ODMSaveDocEx** and **ODMCloseDocEx**.

Optionally, if the application wishes to pre-set some document attributes in the Save As dialog box it can call **ODMSetDocInfo** using the document ID from **ODMNewDoc**. This should be done before calling **ODMSaveAs** or **ODMSaveAsEx**. Example attributes the DMS might accept and perhaps show in its Save As dialog are **ODM_NAME**, **ODM_AUTHOR**, **ODM_KEYWORDS** and others. After **ODMSaveAs** is called, the application might wish to call **ODMGetDocInfo** to see if the user changed the value of any document attribute it is tracking.

Parameters:

handle - in - A handle obtained by a previous **ODMRegisterApp** call.

lpszDocId - out - A pointer to a buffer where the DMS will return the ID of the new document. This buffer must be at least **ODM_DOCID_MAX** bytes in length. If successful then a null-terminated document ID will be returned here. Otherwise the contents of the buffer will be undefined.

dwFlags - in - 0 or a combination of 1 or more of the following values:

ODM_SILENT - The DMS should not require user interaction while satisfying the call. If the call cannot be satisfied without user interaction then an error should be returned.

lpszFormat - in - A null-terminated string naming the format of the new document's content. Refer to the Document Format Names section for information on how file formats are identified in ODMA. Note that this may be changed later via an **ODMSaveAs** call.

lpszDocLocation - in - Normally DMSs select the location for a new document. But if the document already exists and is large or resides on read-only storage then the calling application can use this parameter to tell the DMS where the document is currently stored. This is a hint to the DMS that the document should be left in this location. Note that some DMSs may ignore this hint and move the document anyway. A DMS may ignore this parameter. The calling application should **not** directly access the document in this location following the call to **ODMNewDoc**; it should use **ODMOpenDoc** to obtain a location for subsequent access to the document. In most cases the application should pass Null in this parameter to allow the DMS to determine the document's storage location.

Return value:

ODM_SUCCESS if successful.
ODM_E_CANCEL if the user cancels the new document creation.
ODM_E_FAIL if the DMS failed to create the new document.
ODM_E_USERINT if the **ODM_SILENT** flag was specified and the DMS could not process the function without user interaction.
ODM_E_APPSELECT if the user indicated that he wants to select the document's filename using the application's regular file selection facilities rather than using the DMS. The application should just display its regular selection dialog.

ODM_E_OFFLINE if the DMS has no local data store and is off-line from its server; generally the caller should treat this the same as **ODM_E_APPSELECT**.

ODM_E_HANDLE if *handle* was invalid.

ODMOpenDoc

ODMSTATUS ODMOpenDoc(ODMHANDLE handle, DWORD flags, LPSTR lpszDocId, LPSTR lpszDocLocation)

This function causes the DMS to make a document available to the application. It performs any necessary pre-processing (mapping network drives, checking security, etc.) and then returns to the application a temporary filename that can be used to access the document during the current session. Note that this function does not open the document file; it merely makes the file temporarily available to the calling application. The application can then open, read, write and close the file as needed.

If **ODM_MODIFYMODE** is requested, the DMS may refuse the request if the user has view-only rights (**ODM_E_ACCESS**) or if the document is currently checked-out (**ODM_E_INUSE**) to another user. It is recommended that the application retry the request specifying **ODM_VIEWMODE** in both cases so that the user can at least view the document and possibly save changes to a new document.

Applications are encouraged to give the user the same level of feedback if a DMS based document is opened for read-only access as they would for a document based in the platform's native file system.

When the application is finished using a file which was opened with either **ODM_MODIFYMODE** or **ODM_VIEWMODE** it must call **ODMCloseDoc** or **ODMCloseDocEx**. When an application is finished using a file that was obtained with the **ODM_REFCOPY** option it does not have to call **ODMCloseDoc** or **ODMCloseDocEx**, however, it must delete the temporary file.

If an application has opened a document in **ODM_VIEWMODE** and wishes to switch to **ODM_MODIFYMODE**, it must first call **ODMCloseDoc** or **ODMCloseDocEx** then call **ODMOpenDoc** requesting **ODM_MODIFYMODE**. The same is true if the application wishes to switch from **ODM_MODIFYMODE** to **ODM_VIEWMODE**. The **ODM_E_ALREADYOPENED** error status is returned by the DMS if the application attempts to re-open a document that it has already opened in either of these two modes.

Parameters:

handle - in - A handle obtained by a previous **ODMRegisterApp** call.

flags - in - One or more of the following flags:

ODM_MODIFYMODE - The DMS should make the document available in a modifiable mode. This mode is assumed if **ODM_VIEWMODE** or **ODM_REFCOPY** is not explicitly requested.

ODM_VIEWMODE - The DMS should make the document available in a view-only mode. Any changes made to the document will **not** be transferred back to the document repository. If **ODM_VIEWMODE** and **ODM_MODIFYMODE** are both specified in the same call the **ODM_E_INVARG** error will be returned.

ODM_REFCOPY - The DMS should make a read-only reference copy of the document available to the calling application. The DMS must return a different filespec in *lpszDocLocation* each time this function is called. It is invalid to specify **ODM_REFCOPY** with either **ODM_VIEWMODE** or **ODM_MODIFYMODE**. The calling application must delete any reference copy file obtained using this option and it should not call **ODMCloseDoc** or **ODMCloseDocEx**.

ODM_SILENT - The DMS should not require user interaction while satisfying the call. If the call cannot be satisfied without user interaction then an error should be returned.

lpszDocId - in - A document ID. This is typically obtained by a call to **ODMSelectDoc** or **ODMNewDoc**.

lpszDocLocation - out - A pointer to a buffer of at least **ODM_FILENAME_MAX** bytes in length. The DMS will store in this buffer a null-terminated string indicating where the caller can access the document during the current session. Typically, this will be the full path/file name of the specified document, but some document formats may dictate another type of location such as a directory name. If an error occurs then the contents of the buffer will be undefined.

Return value:

ODM_SUCCESS if successful.
ODM_E_ACCESS if the user does not have the access rights requested (for example, modify mode was requested but the user only has view rights to the document).
ODM_E_INUSE if the user is currently unable to access the document in modify mode because it is checked-out by another user. This

differs from **ODM_E_ACCESS** in that it is expected that the user might be able to access the document in the specified mode at some point in the future.

ODM_E_DOCID if the document ID is invalid or refers to a document that no longer exists.

ODM_E_OFFLINE if the DMS cannot currently access the document because the user client is off-line.

ODM_E_ARCHIVED if the DMS cannot currently supply the document content because it has been archived.

ODM_E_USERINT if the **ODM_SILENT** flag was specified and the DMS could not make the specified document available without user interaction.

ODM_E_INVARG if both **ODM_OPENMODE** and **ODM_VIEWMODE** have been specified or if either of those modes was specified with **ODM_REFCOPY**.

ODM_E_ALREADYOPENED if the application attempts to reopen a document with **ODM_MODIFYMODE** or **ODM_VIEWMODE** that it has already opened with either of these modes. See below for one possible exception.

ODM_E_FAIL if the DMS is unable to make the document accessible for any other reason.

ODM_E_HANDLE if *handle* was invalid.

If the application attempts to open a document for **ODM_VIEWMODE** that it already has opened for **ODM_VIEWMODE**, then the DMS may either return the **ODM_E_ALREADYOPNED** error status or **ODM_SUCCESS** along with the previously returned temporary file specification. The DMS should not return **ODM_SUCCESS** in this case unless it is maintaining a reference count of the number of times this application has opened the document for **ODM_VIEWMODE** access. The application must call **ODMCloseDoc** or **ODMCloseDocEx** once for every time the document was successfully opened. On the last close call the DMS will delete the temporary file.

ODMQueryCapability

ODMSTATUS ODMQueryCapability(ODMHANDLE handle,
LPCSTR lpszDmsId, DWORD function, DWORD item, DWORD flags);

This function is used by a client application to determine if a DMS supports a given ODMA function, or a specific action code in **ODMActivate** or a specific event code in **ODMSetDocEvent**. It can also be used to determine if the DMS supports getting or setting a given document attribute using **ODMGetDocInfo** or **ODMSetDocInfo**.

Parameters:

handle - in - A handle obtained by a previous **ODMRegisterApp** call.

lpszDmsId - in - A pointer to a null terminated string containing the DMS ID of the DMS to query. A DMS ID can be obtained by calling **ODMGetDMSList**, **ODMGetDMS** or **ODMGetDMSInfo**. If this parameter is set to Null, then the default DMS is queried (the one opened with **ODMRegisterApp**).

function - in - One of the following function ids:

ODM_QC_ACTIVATE	**ODM_QC_QUERYEXECUTE**
ODM_QC_CLOSEDOC	**ODM_QC_QUERYGETRESULTS**
ODM_QC_CLOSEDOCEX	**ODM_QC_SAVEAS**
ODM_QC_GETALTERNATECONTENT	**ODM_QC_SAVEASEX**
ODM_QC_GETDMSINFO	**ODM_QC_SAVEDOC**
ODM_QC_GETDOCINFO	**ODM_QC_SAVEDOCEX**
ODM_QC_GETDOCRELATION	**ODM_QC_SELECTDOC**
ODM_QC_GETLEADMONIKER	**ODM_QC_SELECTDOCEX**
ODM_QC_NEWDOC	**ODM_QC_SETALTERNATECONTENT**
ODM_QC_OPENDOC	**ODM_QC_SETDOCEVENT**
ODM_QC_QUERYCLOSE	**ODM_QC_SETDOCRELATION**

item - in - Optional item, action code or event, depending on the function being queried. Specify a value of 0 to query if the DMS supports the function. If **ODM_QC_ACTIVATE** is specified in *function*, then an action value can be specified in this parameter. If **ODM_QC_GETDOCINFO** or **ODM_QC_SETDOCINFO** is specified in *function*, then an item code can be specified in this parameter. If *function* is **ODM_QC_SETDOCEVENT** then an event code can be specified in *item*. See the individual functions for the constants that can be used in the *item* parameter.

flags - in - Many ODMA functions have a flags parameter. This parameter can be used to test if a DMS supports specified input or output flags for a function. For example an application might test to see if a DMS can support unattended operations by specifying the **ODM_SILENT** flag here. It is ignored if the function specified in the *function* parameter does not have a flags parameter.

Return value:

ODM_SUCCESS if the DMS supports the specified function or the specified item/action/event.
ODM_E_NODMS if a DMS ID was specified in *lpszDmsId* which the ODMA Connection Manager could not find.
ODM_E_NOSUPPORT if the DMS does not support the function specified in *function*.
ODM_E_ITEM if the DMS does not support the item, action code or event code specified in *item*.
ODM_E_USERINT if **ODM_SILENT** was specified in *flags* and the DMS cannot satisfy the function specified in *function* without user interaction.
ODM_E_REQARG if nothing was specified in the *function* parameter.
ODM_E_INVARG if *flags* was invalid for the function specified in *function*.
ODM_E_HANDLE if *handle* was invalid.
ODM_E_FAIL if the DMS was unable to process the function for other reasons.

ODMQueryClose

ODMSTATUS ODMQueryClose(ODMHANDLE handle, LPCSTR queryId)

This function is used by a client application to indicate that it has finished using the query and that the involved DMS(s) should release any resources and/or memory for the result set.

Parameters:

> *handle* - in - A handle obtained by a previous **ODMRegisterApp** call.
>
> *queryId* - in - A pointer to a null-terminated string containing a query id returned by a previous call to **ODMQueryExecute.**

Return value:

> **ODM_SUCCESS** on success, where upon the *queryId* is no longer valid.
> **ODM_E_HANDLE** if *handle* was invalid.
> **ODM_E_FAIL** if one or more input values are invalid.

ODMQueryExecute

ODMSTATUS ODMQueryExecute(ODMHANDLE handle, LPCSTR lpszQuery, DWORD flags, LPCSTR lpszDMSList, LPSTR queryId)

This function will pass the *lpszQuery* along to each DMS as specified by the *flags* parameter. It will return a query ID that can be used in subsequent calls to **ODMQueryGetResults** and **ODMQueryClose**. If this function is successful, the calling application must eventually call **ODMQueryClose** to release this query.

Parameters:

> *handle* - in - A handle obtained by a previous **ODMRegisterApp** call.
>
> *lpszQuery* - in - The query to be processed by one or more DMSs. Refer to the Query Syntax section for the exact syntax specification.
>
> *flags* - in - One of the following:
>
> > **ODM_ALL** - All DMS providers on this workstation.
> > **ODM_SPECIFIC** - A set of specific DMSs to run query against.
>
> *lpszDMSList* - in - Valid only if value of *flags* is **ODM_SPECIFIC**. A buffer containing a list of DMSs. The format for the list is a collection of null-terminated strings. The list is terminated by an empty string (i.e. “<id#1>\0<id#2>\0\0”).
>
> *queryId* - out - An id to be used for search identification to subsequent calls. This is the query ID a client application uses to communicate with Connection Manager, which in turn maps the ID to a DMS-specific query ID. For the case of multiple DMS’s, this ID is mapped to multiple DMS-specific query IDs. Such mapping is handled by Connection Manager and is transparent to the client application. The buffer size for *queryId* should be at least **ODM_QUERYID_MAX** bytes.

Return value:

> **ODM_SUCCESS** if successful.
> **ODM_E_CANCEL** if the user canceled the query.
> **ODM_E_PARTIALSUCCESS** if one or more DMSs returned a successful status and one or more DMSs return a failure.
> **ODM_E_HANDLE** if *handle* was invalid.

ODM_E_FAIL if invalid data was sent in, or no DMS was specified or all DMSs failed to process the query.

ODMQueryGetResults

ODMSTATUS ODMQueryGetResults(ODMHANDLE handle, LPCSTR queryId, LPSTR lpszDocId, LPSTR lpszDocName, WORD docNameLen, LPWORD pwDocCount)

This function will fetch *pwDocCount* number of documents at a time.

Parameters:

handle - in - A handle obtained by a previous **ODMRegisterApp** call.

queryId - in - A pointer to a null-terminated string containing a query id returned by a previous call to **ODMQueryExecute.**

lpszDocId - out - A pointer to a buffer to receive document Ids for documents that satisfied the search referenced in *queryId*. This buffer needs to be at least (**ODM_DOCID_MAX** * *pwDocCount*) + 1 bytes in length.

lpszDocName - out - A pointer to a buffer to receive **ODM_NAME** attribute values for documents that satisfied the search referenced in *queryId*. This buffer needs to be at least (*docNameLen* * *pwDocCount*) + 1 bytes in length.

docNameLen - in - The length of a document name including the null-terminator.

pwDocCount - in/out - A pointer to a variable used to pass a document count. On input, *pwDocCount* contains the number of documents expected by the client application. On output, *pwDocCount* contains the actual number of documents returned by this function.

Return value:

ODM_SUCCESS if no error and there are more rows available.
ODM_E_NOMOREDATA if there is no more data. The contents of *lpszDocId*, *lpszDocName* and *docCount* are undefined.
ODM_E_REQARG if a required parameter is not specified.
ODM_E_HANDLE if *handle* was invalid.
ODM_E_FAIL if an error occurs.

ODMQueryInterface

HRESULT ODMQueryInterface(ODMHANDLE handle, LPSTR lpszDocId, REFIID riid, LPVOID FAR *ppvObj)

An application can use this function to get a COM interface from an ODMA provider. All ODMA providers support the **IODMDocMan** interface and some support **IODMDocMan2** and **IODMQuery**. Individual DMSs may support other interfaces as well. Note that this function is prototyped in odmacom.h instead of odma.h, so that non-COM-aware applications do not have to #include the header files that define interface IDs.

Parameters:

handle - in - A handle obtained by a previous **ODMRegisterApp** call.

lpszDocId - in - An ODMA document ID or Null. If Null then the application's default DMS is queried for the interface. Otherwise the DMS that created this document ID is queried.

riid - in - The interface to be obtained from the DMS.

ppvObj - out - If the requested interface is supported by the DMS then it is returned here. Otherwise *ppvObj* is set to Null.

Return value:

S_OK if successful.
E_INVALIDARG if *handle* or *lpszDocId* is invalid.
E_NOINTERFACE if the requested interface is not supported by the DMS.
E_ACCESSDENIED if the DMS refuses the calling application.
E_FAIL if the DMS fails to initialize itself or fails for any other reason.

ODMRegisterApp

ODMSTATUS ODMRegisterApp(ODMHANDLE FAR *pHandle, WORD version, LPSTR lpszAppId, DWORD dwEnvData, LPVOID pReserved)

ODMRegisterApp registers an application with the appropriate Document Management System (DMS) and returns a session handle that can be used in calls to other ODMA functions.

An application normally calls this function before calling any other ODMA function. Some functions such as **ODMGetDMSCount**, **ODMGetDMSList**, **ODMGetDMS** and **ODMSetDMS** can be called before **ODMRegisterApp**. These are used by applications that wish to control which DMS they will access when **ODMRegisterApp** is called.

A task may call **ODMRegisterApp** more than once. Each call will return a different handle, each of which must be released with **ODMUnRegisterApp**.

Parameters:

pHandle - out - If successful, a session handle is returned here that can be used in calls to other ODMA functions. If the registration fails then 0 is returned here.

version - in - Specifies the version of the API required by the application. 100 should be passed to indicate version 1.0, 150 for version 1.5, 200 for version 2.0, etc. The macro **ODM_API_VERSION** can be used to get the current version number at compile time. All versions of the ODMA API will be downward compatible, so this should be interpreted as the minimum version number that the calling application expects the DMS to support. If the DMS does not support the specified version or a higher version then it should return an error.

lpszAppId - in - A unique identifier for the application. The maximum length for this string is **ODM_APPID_MAX**, which is 16 characters including the terminating Null. The application ID cannot begin with a digit. It is recommended that a Windows application use the application class name it uses in the registry as its ODMA Application ID, but this is not required.

dwEnvData - in - Environment data. On Windows platforms this is a Window handle for a parent window in the calling application. The DMS may use this window handle as the parent window for any dialogs or other windows it displays in response to ODMA calls. This handle must remain valid for the duration of the ODMA session (i.e. until **ODMUnRegisterApp** is called).

pReserved - in - Reserved for future use. Must be set to Null.

Return value:

ODM_SUCCESS if successful.
ODM_E_NODMS if no Document Management System has been registered for the calling application.
ODM_E_CANTINIT if a DMS is registered for the calling application, but it fails to initialize itself.
ODM_E_VERSION if the DMS does not support the requested version of the API.
ODM_E_REFUSED if the DMS has been configured to refuse ODMA access for the calling application. This is not considered an error condition. If this status is returned, the application should simply act as if ODMA is not on the system.

ODMSaveAs

ODMSTATUS ODMSaveAs(ODMHANDLE handle, LPSTR lpszDocId,
LPSTR lpszNewDocId, LPSTR lpszFormat,
ODMSAVEASCALLBACK pcbCallBack, LPVOID pInstanceData)

This function causes the DMS to return a new document ID for a document that is based on an existing document. An application would typically call this function in response to the user selecting a File Save As menu option. **ODMSaveAs** causes the DMS to display options to the user for selecting the destination for the new document. This might be an entirely new document or a new version of the current document. Any dialog box displayed by the DMS should be task/application modal.

Often the application will want to present additional options to the user at this point such as different file formats or encrypting the document. This is accomplished via the *pcbCallBack* parameter. ODMA implementers should provide a method for users to access this function if desired. For example, the DMS may show a dialog that includes an Options button. If the user clicks this button, the DMS would call the application's callback function which would give the application a chance to display other save options.

Note that following a successful call to **ODMSaveAs** the calling application may have two different document IDs to work with. This is different than the situation with **ODMSaveDoc** where the new ID replaces the old one in the current session. The state of the document specified by the old ID remains the same after the call. The document specified by the new ID will be in the closed state following the call. A typical sequence of operations an application might follow in response to the user selecting File **|** Save As would be:

1. Application passes the currently open document's ID to **ODMSaveAs**. If an empty string is returned for the new ID, then the application should call **ODMSaveDoc** with the current document ID to indicate to the DMS that document should be saved in the document repository. If a new ID for the document is returned then continue with the steps below.
2. Application calls **ODMOpenDoc** on the new ID. This returns a new filename for the document.
3. Application saves the document to the new filename and then calls **ODMSaveDoc** on the new ID to indicate to the DMS that the new document should be saved in the document repository.
4. Application calls **ODMCloseDoc** on the old ID.
5. The application can now forget about the old ID and use the new ID for all subsequent operations on the file. When the current session is completed it will call **ODMCloseDoc** on the new ID.

Parameters:

handle - in - A handle obtained by a previous **ODMRegisterApp** call.

lpszDocId - in - A null-terminated document ID. This is typically obtained by a call to **ODMSelectDoc**, **ODMSelectDocEx** or **ODMNewDoc**. This document may or may not be open at the time that **ODMSaveAs** is called. Its open status will remain the same after this call.

lpszNewDocId - out - A pointer to a buffer where the DMS will return the ID of the new document. This buffer must be at least **ODM_DOCID_MAX** bytes in length. If successful then a null-terminated document ID will be returned here, unless the document is saved with the current ID. In this case the first byte of this buffer will be set to Null and 0 will be returned. Otherwise the contents of the buffer will be undefined.

lpszFormat - in - A null-terminated string naming the format in which the application expects to save the document. This may be passed as a parameter to the *pcbCallBack* function which may return a different format. Refer to the Document Format Names section for information on how file formats are identified in ODMA.

pcbCallBack - in - A pointer to a callback function that can be used by the application to make other saving options available to the user. Any UI presented by this callback function should be task modal. The function should return a pointer to a format string which may or may not be the same as the original format string. This parameter may be null if the application does not wish to present any options. Under Microsoft Windows the procedure-instance address of the callback function (obtained from **MakeProcInstance**) should be used. The callback function's interface is as follows:

> LPSTR __cdecl SaveAsCallBack(DWORD dwEnvData, LPSTR lpszFormat, LPVOID pInstanceData)
>
> *dwEnvData* - in - Environment data. On Windows platforms this is a Window handle for a parent window to associate with the callback function's display. If the DMS displayed a dialog in response to the **ODMSaveAs** call then it should pass the window handle for that dialog. Otherwise it should pass the window handle obtained from the **ODMRegisterApp** call. The callback function may use this window handle as the parent window for any dialogs or other windows it displays.

lpszFormat - The currently selected document format. The DMS should allocate at least **ODM_FORMAT_MAX** for this buffer. If the application handling the callback wishes to update this DMS buffer and return its pointer back as the value of the callback, then it should take care to not overwrite the buffer's length.

pInstanceData - The instance data passed from the calling application.

pInstanceData - in - A pointer to caller context information that will be passed to the *pcbCallBack* function. This data will not be accessed by ODMA.

Return value:

ODM_SUCCESS if successful.
ODM_E_CANCEL if the user cancels the creation of the new document.
ODM_E_DOCID if the document ID is invalid or refers to a document that no longer exists.
ODM_E_APPSELECT if the user selected to save the document as a non-profiled document.
ODM_E_OFFLINE if the DMS has no local data store and is off-line from its server; generally the caller should treat this the same as **ODM_E_APPSELECT**.
ODM_E_FAIL if the DMS is unable to display its Save As dialog box or create the new document.
ODM_E_HANDLE if *handle* was invalid.

ODMSaveAsEx

ODMSTATUS ODMSaveAsEx(ODMHANDLE handle, LPSTR lpszDocId,
LPSTR lpszNewDocId, LPSTR lpszFormat,
ODMSAVEASCALLBACK pcbCallBack, LPVOID pInstanceData,
LPDWORD pdwFlags)

The **ODMSaveAsEx** function extends **ODMSaveAs** by adding a *pdwFlags* parameter to allow for unattended document creation. It also can be called without first calling **ODMNewDoc**.

This function causes the DMS to return a new document ID for a document that is based on an existing document. An application would typically call this function in response to the user selecting a File Save As menu option. **ODMSaveAsEx** causes the DMS to display options to the user for selecting the destination for the new document. This might be an entirely new document or a new version of the current document. Any dialog box displayed by the DMS should be task/application modal.

Often the application will want to present additional options to the user at this point such as different file formats or encrypting the document. This is accomplished via the *pcbCallBack* parameter. ODMA implementers should provide a method for users to access this function if desired. For example, the DMS may show a dialog that includes an Options button. If the user clicks this button, the DMS would call the application's callback function which would give the application a chance to display other save options.

Note that following a successful call to **ODMSaveAsEx** the calling application may have two different document IDs to work with. This is different than the situation with **ODMSaveDoc** or **ODMSaveDocEx** where the new ID replaces the old one in the current session. The state of the document specified by the old ID remains the same after the call. The document specified by the new ID will be in the closed state following the call. A typical sequence of operations an application might follow in response to the user selecting File **|** Save As would be:

1. Application passes the currently open document's ID to **ODMSaveAsEx**. If an empty string is returned for the new ID, then the application should call **ODMSaveDoc** or **ODMSaveDocEx** with the current document ID to indicate to the DMS that document should be saved in the document repository. If a new ID for the document is returned then continue with the steps below.
2. Application calls **ODMOpenDoc** on the new ID. This returns a new filename for the document.

3. Application saves the document to the new filename and then calls **ODMSaveDoc** or **ODMSaveDocEx** on the new ID to indicate to the DMS that the new document should be saved in the document repository.
4. Application calls **ODMCloseDoc** or **ODMCloseDocEx** on the old ID.
5. The application can now forget about the old ID and use the new ID for all subsequent operations on the file. When the current session is completed it will call **ODMCloseDoc** or **ODMCloseDocEx** on the new ID.

Parameters:

handle - in - A handle obtained by a previous **ODMRegisterApp** call.

lpszDocId - in - A null-terminated document ID. This is typically obtained by a call to **ODMSelectDoc**, **ODMSelectDocEx** or **ODMNewDoc**. This document may or may not be open at the time that **ODMSaveAsEx** is called. Its open status will remain the same after this call.

This *lpszDocId* parameter can optionally be set to NULL if the calling application did not first call **ODMNewDoc**. In this case, the DMS will return a document ID in the buffer specified in *lpszNewDocId*.

lpszNewDocId - out - A pointer to a buffer where the DMS will return the ID of the new document. This buffer must be at least **ODM_DOCID_MAX** bytes in length. If successful then a null-terminated document ID will be returned here, unless the document is saved with the current ID. In this case the first byte of this buffer will be set to Null and 0 will be returned. Otherwise the contents of the buffer will be undefined.

lpszFormat - in - A null-terminated string naming the format in which the application expects to save the document. This may be passed as a parameter to the *pcbCallBack* function which may return a different format. Refer to the Document Format Names section for information on how file formats are identified in ODMA.

pcbCallBack - in - A pointer to a callback function that can be used by the application to make other saving options available to the user. Any UI presented by this callback function should be task modal. The function should return a pointer to a format string which may or may not be the same as the original format string. This parameter may be null if the application does not wish to present any options. Under Microsoft Windows the procedure-instance address of the callback function (obtained from MakeProcInstance) should be used. The callback function's interface is as follows:

LPSTR __cdecl SaveAsCallBack(DWORD dwEnvData, LPSTR lpszFormat, LPVOID pInstanceData)

dwEnvData - in - Environment data. On Windows platforms this is a Window handle for a parent window to associate with the callback function's display. If the DMS displayed a dialog in response to the **ODMSaveAsEx** call then it should pass the window handle for that dialog. Otherwise it should pass the window handle obtained from the **ODMRegisterApp** call. The callback function may use this window handle as the parent window for any dialogs or other windows it displays.

lpszFormat - The currently selected document format. The DMS should allocate at least **ODM_FORMAT_MAX** for this buffer. If the application handling the callback wishes to update this DMS buffer and return its pointer back as the value of the callback, then it should take care to not overwrite the buffer's length.

pInstanceData - The instance data passed from the calling application.

pInstanceData - in - A pointer to caller context information that will be passed to the *pcbCallBack* function. This data will not be accessed by ODMA.

pdwFlags - in/out - A pointer to a variable containing flags used on both input and output. On input, 0 or the following flag:

ODM_SILENT - The DMS should not require user interaction while satisfying the call. If the call cannot be satisfied without user interaction then an error should be returned. If the DMS has enough information to create a new document then it should return **ODM_SUCCESS**.

ODMA 2.0 does not define any output flags at this time.

Return value:

ODM_SUCCESS if successful.
ODM_E_CANCEL if the user cancels the creation of the new document.
ODM_E_DOCID if the document ID is invalid or refers to a document that no longer exists.
ODM_E_APPSELECT if the user selected to save the document as a non-profiled document.

ODM_E_OFFLINE if the DMS has no local data store and is off-line from its server; generally the caller should treat this the same as **ODM_E_APPSELECT**.

ODM_E_USERINT if the **ODM_SILENT** flag was specified and the DMS does not support document creation without user interaction or the DMS does not have enough information to create the document without user interaction.

ODM_E_FAIL if the DMS is unable to display its Save As dialog box or create the new document.

ODM_E_HANDLE if *handle* was invalid.

ODMSaveDoc

ODMSTATUS ODMSaveDoc(ODMHANDLE handle, LPSTR lpszDocId,
LPSTR lpszNewDocId)

This function tells the DMS that the document should be saved to the document repository. An application would typically call this function after saving changes to the temporary file returned by **ODMOpenDoc**. A new document ID is returned to the application which should be used for all subsequent operations on the document. This ID replaces the previous document ID in the current session. The new ID may or may not be the same as the original ID. It will usually be the same unless the DMS saved the document as a new version or as a new document. If the new ID is different from the previous ID then the previous ID cannot be used subsequently without doing an **ODMOpenDoc** on it.

Parameters:

handle - in - A handle obtained by a previous **ODMRegisterApp** call.

lpszDocId - in - A document ID. This is typically obtained by a call to **ODMSelectDoc** or **ODMNewDoc**. This document must have been previously opened in modify mode by a call to **ODMOpenDoc**.

lpszNewDocId - out - A pointer to a buffer where the DMS will return the new ID of the document. This buffer must be at least **ODM_DOCID_MAX** bytes in length. If successful then a null-terminated document ID will be returned here. Otherwise the contents of the buffer will be undefined.

Return value:

ODM_SUCCESS if successful.
ODM_E_NOOPEN if the document wasn't previously opened with **ODMOpenDoc**.
ODM_E_OPENMODE if the document was not previously opened in modifiable mode.
ODM_E_FAIL if the DMS is unable to save the document to the document repository.
ODM_E_HANDLE if *handle* was invalid.

ODMSaveDocEx

ODMSTATUS ODMSaveDocEx(ODMHANDLE handle, LPSTR lpszDocId, LPSTR lpszNewDocId, LPDWORD pdwFlags)

This function tells the DMS that the document should be saved to the document repository. **ODMSaveDocEx** is basically the same as **ODMSaveDoc**, except it has a *pwdFlags* parameter. This provides the application with more control during the document creation or save process. There is an **ODM_SILENT** flag that facilitates unattended document creation or saving. Some DMSs display a user interface when saving a document (i.e. to have the user specify revision or version information). This flag allows the calling application to suppress this UI if necessary or to influence it, if the DMS should allow that.

An application would typically call this function after saving changes to the temporary file returned by **ODMOpenDoc**. A new document ID is returned to the application which should be used for all subsequent operations on the document. This ID replaces the previous document ID in the current session. The new ID may or may not be the same as the original ID. It will usually be the same unless the DMS saved the document as a new version or as a new document. If the new ID is different from the previous ID then the previous ID cannot be used subsequently without doing an **ODMOpenDoc** on it.

Parameters:

handle - in - A handle obtained by a previous **ODMRegisterApp** call.

lpszDocId - in - A document ID. This is typically obtained by a call to **ODMSelectDoc** or **ODMNewDoc**. This document must have been previously opened in modify mode by a call to **ODMOpenDoc**.

lpszNewDocId - out - A pointer to a buffer where the DMS will return the new ID of the document. This buffer must be at least **ODM_DOCID_MAX** bytes in length. If successful then a null-terminated document ID will be returned here. Otherwise the contents of the buffer will be undefined.

pdwFlags - in/out - A pointer to a variable containing flags used on both input and output. On input, 0 or one or more of the following flags:

ODM_SILENT - The DMS should not require user interaction while satisfying the call. If the call cannot be satisfied without user interaction then an error should be returned.

ODM_VERSION_SAME - If saving an existing document the DMS should keep the version information the same. If the DMS does not support versioning this flag should be ignored.

ODM_VERSION_MAJOR - If saving an existing document the DMS should increment the major version number. If the DMS displays a dialog this flag is an indication of how the dialog might be initialized. If the DMS does not support versioning this flag should be ignored.

ODM_VERSION_MINOR - If saving an existing document the DMS should increment the minor version number. A DMS that doesn't support minor versions should just treat this like a major version. If the DMS displays a dialog this flag is an indication of how the dialog might be initialized.

Upon return, one of the following flags must be set by the DMS unless an error occurred:

ODM_VERSION_SAME - The DMS either kept the version the same or it does not support versioning.

ODM_VERSION_CHANGED – The DMS changed the version number, possibly with user interaction.

Return value:

ODM_SUCCESS if successful.
ODM_E_CANCEL if the DMS displayed a dialog and the user canceled the save operation.
ODM_E_NOOPEN if the document wasn't previously opened with **ODMOpenDoc**.
ODM_E_OPENMODE if the document was not previously opened in modifiable mode.
ODM_E_USERINT if the **ODM_SILENT** flag was specified and the DMS could not save the document without user interaction or the DMS does not have enough information to create a new document without user interaction.
ODM_E_INVARG if the value in *pdwFlags* was invalid. More than one of the **ODM_VERSION*** flags was set.
ODM_E_OFFLINE if the DMS has no local data store and is off-line from its server.
ODM_E_FAIL if the DMS is unable to save the document to the document repository.
ODM_E_HANDLE if *handle* was invalid.

ODMSelectDoc

ODMSTATUS ODMSelectDoc(ODMHANDLE handle, LPSTR lpszDocId, LPDWORD pdwFlags)

This function causes the DMS to return a document ID representing a document that has been selected for some action. Typically the DMS will display searching and other dialogs that allow the user to interactively select a document from among those managed by the DMS. An application would typically call this function whenever the user needs to select a document to be opened or imported. Any dialog box displayed by the DMS should be task/application modal.

If the user elects to open a document in **ODM_VIEWMODE**, then the application should open the document with **ODMOpenDoc** specifying **ODM_VIEWMODE**. If a user selects **ODM_MODIFYMODE**, the application is free to open the document with either the **ODM_VIEWMODE** or **ODM_REFCOPY** options if it only needs read-only access to it.

Parameters:

handle - in - A handle obtained by a previous **ODMRegisterApp** call.

lpszDocId - out - A pointer to a buffer where the DMS will return the ID of the document selected by the user. This buffer must be at least **ODM_DOCID_MAX** bytes in length. If successful then a null-terminated document ID will be returned here. Otherwise the contents of the buffer will be undefined.

pdwFlags - in/out - A pointer to a variable containing flags used on both input and output. On input, 0 or one or more of the following flags:

ODM_SILENT - The DMS should not require user interaction while satisfying the call. If the call cannot be satisfied without user interaction then an error should be returned.

ODM_VIEWMODE - On input this is a hint to the DMS that the application will only be opening the selected document for read-only access. Support for this input flag is optional for the DMS. A DMS that supports this might modify its document selection user interface appropriately and set only the **ODM_VIEWMODE** flag on output.

Upon return, one of the following flags will be set unless an error occurred:

ODM_MODIFYMODE - The user indicated that the selected document should be opened in a modifiable mode.

ODM_VIEWMODE - The user indicated that the selected document should be opened in a view-only mode.

Return value:

ODM_SUCCESS if successful.
ODM_E_CANCEL if the user does not make a selection.
ODM_E_APPSELECT if the user indicated that he wants to make a selection using the application's regular file selection facilities rather than using the DMS. The application should just display its regular selection dialog.
ODM_E_OTHERAPP if the user selected a document and the DMS allowed the user to open it with another application or its native application; generally the caller should treat this the same as **ODM_E_CANCEL**.
ODM_E_USERINT if the **ODM_SILENT** flag was specified and the DMS could not select a document without user interaction.
ODM_E_OFFLINE if the DMS has no local data store and is off-line from its server; generally the caller should treat this the same as **ODM_E_APPSELECT**.
ODM_E_FAIL if the DMS is unable to select a document for any other reason.
ODM_E_HANDLE if *handle* was invalid.

In the case that **ODM_E_APPSELECT** is returned and a document is opened through the application's regular file selection facilities, the following behaviors are recommended:

1. File | Save and File | Close should be handled as though the DMS were not present.

2. File | Save As should he handled through ODMA (specifically through the **ODMSaveAs** function). This will allow documents to be imported into the DMS. The exception to this is if the application allows the user to select a DMS from within its native file system Save As dialog.

ODMSelectDocEx

ODMSTATUS ODMSelectDocEx(ODMHANDLE handle, LPSTR lpszDocIds, LPWORD pwDocIdsLen, LPWORD pwDocCount, LPDWORD pdwFlags, LPSTR lpszFormatFilter)

ODMSelectDocEx extends **ODMSelectDoc**. This function causes the DMS to return one or more document IDs representing documents that have been selected for some action. An application may call this function if it allows the user to open multiple documents in one command. Typically the DMS will display searching and other dialogs that allow the user to interactively select documents from among those managed by the DMS. If a DMS does not support multiple selection, then it is free to limit the user to selecting only one document. Any dialog box displayed by the DMS should be task/application modal.

The **ODMSelectDocEx** function has an *lpszFormatFilter* parameter which will allow an application to specify to a DMS what file types the user should select. This can provide for closer integration between the application and the DMS.

If the user elects to open a document in **ODM_VIEWMODE**, then the application should open the document with **ODMOpenDoc** specifying **ODM_VIEWMODE**. If a user selects **ODM_MODIFYMODE**, the application is free to open the document with either the **ODM_VIEWMODE** or **ODM_REFCOPY** options if it only needs read-only access to it.

Parameters:

handle - in - A handle obtained by a previous **ODMRegisterApp** call.

lpszDocIds - out - A pointer to a buffer where the DMS will return the IDs of the documents selected by the user. If successful then a series of null-terminated document IDs will be returned here, with the last ID followed by an extra Null terminator. Otherwise the contents of the buffer will be undefined. This buffer must be at least **ODM_DOCID_MAX *** *pwDocCount* + 1 bytes in length.

pwDocIdsLen - in/out - On input this parameter is a pointer to a variable that specifies the length of the buffer pointed to by *lpszDocIds*. If the call fails because the buffer is not big enough, this variable is updated by the DMS to contain the required buffer size.

pwDocCount - in/out - On input this parameter is a pointer to a variable that specifies the maximum number of documents the calling application can handle in the buffer specified in *lpszDocIds*. The DMS should not allow the user to select more than this number of documents. On output the DMS must update this value to contain the actual number of documents returned in *lpszDocIds*.

pdwFlags - in/out - A pointer to a variable containing flags used on both input and output. On input, 0 or one or more of the following flags:

ODM_SILENT - The DMS should not require user interaction while satisfying the call. If the call cannot be satisfied without user interaction then an error should be returned.

ODM_VIEWMODE - On input this is a hint to the DMS that the application will only be opening the selected document for read-only access. Support for this input flag is optional for the DMS. A DMS that supports this might modify its document selection user interface appropriately and set only the **ODM_VIEWMODE** flag on output.

ODM_TEMPLATES - The DMS should, if possible, only show the user document templates. This option allows the application to access document templates which the DMS maintains for work-groups or the current user.

Upon return, one of the following flags will be set unless an error occurred:

ODM_MODIFYMODE - The user indicated that the selected document(s) should be opened in a modifiable mode. If the DMS user interface allows the user to select some documents for modify access and others for view-only access, then it is recommended that this flag be set.

ODM_VIEWMODE - The user indicated that the selected document(s) should be opened in a view-only mode.

lpszFormatFilter - in - If specified, a long pointer to a buffer containing a sequence of one or more null-terminated format names. The last sequence must be terminated with two null characters. The information in *lpszFormatFilter* is a hint to DMS of which file types or templates the application would like the user to select. The DMS may use this format list as is, augment it with other formats, or ignore it altogether. The user is free to select any document type, so the calling application must be prepared for the possibility that the user will select a file format that isn't specified in the *lpszFormatFilter* parameter. Refer to the Document

Format Names section for information on how file formats are identified in ODMA.

Return value:

ODM_SUCCESS if successful.
ODM_E_CANCEL if the user does not make a selection.
ODM_E_APPSELECT if the user indicated that he wants to make a selection using the application's regular file selection facilities rather than using the DMS. The application should just display its regular selection dialog.
ODM_E_OTHERAPP if the user selected a document and the DMS allowed the user to open it with another application or its native application; generally the caller should treat this the same as **ODM_E_CANCEL**.
ODM_E_USERINT if the **ODM_SILENT** flag was specified and the DMS could not select a document without user interaction.
ODM_E_NOSUPPORT if the DMS does not support the function.
ODM_E_INVARG if the variables pointed to by *pwDocCount* or *pwDocIdsLen* have value of 0.
ODM_E_REQARG if a required parameter is not specified (i.e. *lpszDocIds*, *pwDocIdsLen* or *pwDocCount*.
ODM_E_TRUNCATED if the buffer specified in *lpszDocIds* is too small to hold the selected document Ids. When this error occurs the *pwDocIdsLen* variable contains the required buffer size.
ODM_E_OFFLINE if the DMS has no local data store and is off-line from its server; generally the caller should treat this the same as **ODM_E_APPSELECT**.
ODM_E_HANDLE if *handle* was invalid.

ODMSetAlternateContent

ODMSTATUS ODMSetAlternateContent(ODMHANDLE handle,
LPSTR lpszDocId, LPDWORD pdwFlags, LPSTR lpszFormat,
LPSTR lpszDocLocation)

This function allows an application to create or update an alternate content file for the specified document in the DMS. The application can either provide the alternate content file or it can instruct the DMS to create or update the content file itself. The format of the alternate content file is specified in the **ODMSetAlternateContent** function call. The DMS may or may not accept the specified alternate content file. The application is responsible for deleting the alternate content file when it is finished using it.

Parameters:

handle - in - A handle obtained by a previous **ODMRegisterApp** call.

lpszDocId - in - A null-terminated document ID. This is typically obtained by a call to **ODMSelectDoc**, **ODMSelectDocEx** or **ODMNewDoc**. The specified document may or may not be open when this function is called. This document ID refers to the main document for which an alternate content file is being created or updated.

pdwFlags - in/out - A pointer to a variable containing flags used on both input and output. On input, zero (0) or one or more of following flags:

ODM_SILENT - The DMS should not require user interaction while satisfying the call. If the call cannot be satisfied without user interaction then an error should be returned.

ODM_ALT_DELETE - The DMS should delete the specified alternate content file if the user has the appropriate access rights.

ODMA 2.0 does not define any output flags at this time.

lpszFormat - in - A null-terminated string containing the format name of the alternate format file which is being created or updated. Refer to the Document Format Names section for information on how file formats are identified in ODMA. If the DMS cannot accept the specified format it will return the **ODM_E_NOSUPPORT** status.

lpszDocLocation - in - A pointer to a null-terminated string containing the file specification of the alternate content file. The file referenced in this parameter should be in the format indicated in the *lpszFormat* parameter. This parameter may be set to Null if the calling application wants the DMS to create the alternate rendition file itself. If the DMS cannot create or update the alternate content file then it will return **ODM_E_NOSUPPORT**. It is the responsibility of the calling application, not the DMS, to delete this file when it is finished with it.

Return value:

ODM_SUCCESS if successful.
ODM_E_ACCESS if the user does not have the appropriate access rights to create, update or delete an alternate content file for the main document.
ODM_E_INUSE if the user is currently unable to create, update or delete the alternate content file because the main document is checked-out by another user.
ODM_E_DOCID if the document ID is invalid or refers to a document that no longer exists.
ODM_E_OFFLINE if the DMS cannot currently access the document because the user client is off-line.
ODM_E_ARCHIVED if the DMS cannot currently update the alternate content because the main document has been archived.
ODM_E_USERINT if the **ODM_SILENT** flag was specified and the DMS could not make the specified document available without user interaction.
ODM_E_INVARG if *lpszFormat* is Null.
ODM_E_NOSUPPORT if the DMS does not support the function or it cannot accept or generate the requested alternate content format for the specified document.
ODM_E_FAIL if the DMS is unable to complete the function for any other reason.
ODM_E_HANDLE if *handle* was invalid.

ODMSetDMS

ODMSTATUS ODMSetDMS(LPCSTR lpszAppId, LPCSTR lpszDmsId)

This function provides an application a programmatic way to set its default DMS. **ODMSetDMS** does not change any default DMS settings in the registry. The DMS ID set using this function will take effect the next time the application calls **ODMRegisterApp**. It will not alter the default DMS in any existing session the application has already established with **ODMRegisterApp**.

Parameters:

lpszAppId - in - A pointer to a null-terminated string containing an Application ID.

lpszDmsId - in - A pointer to a null-terminated string containing a DMS ID. This is the DMS the application will be connected to when it calls **ODMRegisterApp**. If this parameter is set to Null the Connection Manager will set the application to use to the default DMS in the registry.

Return value:

ODM_SUCCESS if successful.
ODM_E_NODMS if the DMS specified in *lpszDmsId* is not valid.
ODM_E_REQARG if the *lpszAppId* parameter is Null.

ODMSetDocEvent

ODMSTATUS ODMSetDocEvent(ODMHANDLE handle, LPSTR lpszDocId, DWORD flags, DWORD event, LPVOID lpData, DWORD dwDataLen, LPSTR lpszComment)

An application can use this function to pass event information about the document to the DMS. Events like printing, sending, posting can be sent by the application to the DMS. The DMS may or may not accept the event information.

Parameters:

handle - in - A handle obtained by a previous **ODMRegisterApp** call.

lpszDocId - in - A null-terminated document ID. This is typically obtained by a call to **ODMSelectDoc**. The specified document may or may not be open when **ODMSetDocEvent** is called.

flags - in - 0 or a combination of 1 or more of the following values:

ODM_SILENT - The DMS should not require user interaction while satisfying the call. If the call cannot be satisfied without user interaction then an error should be returned.

event - in - One of the following:

ODM_EVENT_PRINTED - The document was printed.
ODM_EVENT_POSTED - The document was posted into a discussion group.
ODM_EVENT_SENT - The document was mailed.
ODM_EVENT_FAXED - The document was faxed.
ODM_EVENT_ROUTED - The document was routed using a workflow system.
ODM_EVENT_COPIED - The document was copied or imported into another DMS or application.
ODM_EVENT_CONVERTED - The document was used as source to a format conversion or scanning (i.e. OCR) operation that resulted in the creation of a new document or another version or rendition of an existing document.
ODM_EVENT_DMSDEFINED - The *lpData* parameter contains a DMS-specific indication of the data being passed as well as the data itself. Note that an application must know which DMS it is talking to and must understand the data

indications supported by that DMS in order to use this event name.

lpData - in - The application may pass other information regarding the event in this parameter. The calling application and the DMS will have to coordinate on the format of this information. Null should be passed if the application has no meaningful information to pass through this parameter.

dwDataLen - in - The length of the data passed in the *lpData* parameter. Ignored if *lpData* is Null.

lpszComment - in - An optional, null-terminated, string containing either a user supplied or application generated comment. Set this parameter to Null if a comment isn't specified.

Return value:

ODM_SUCCESS if successful.
ODM_E_DOCID if the document ID is invalid or refers to a document that no longer exists.
ODM_E_OFFLINE if the DMS cannot currently process the event for the document because the user client is off-line.
ODM_E_ITEM if *event* is invalid or not supported by the DMS.
ODM_E_NOSUPPORT if the DMS does not support the function.
ODM_E_USERINT if the **ODM_SILENT** flag was specified and the DMS could not process the call without user interaction.
ODM_E_REQARG if *lpData* is specified and *dwDataLen* is 0.
ODM_E_HANDLE if *handle* was invalid.
ODM_E_FAIL if the DMS was unable to process the function for other reasons.

ODMSetDocInfo

ODMSTATUS ODMSetDocInfo(ODMHANDLE handle, LPSTR lpszDocId, WORD item, LPSTR lpszData)

An application can use this function to pass information about the document to the DMS. The DMS may or may not accept the information; most DMSs validate attributes like document types and authors against predefined tables.

It is recommended that the DMS not display any user interface while processing this function.

Parameters:

handle - in - A handle obtained by a previous **ODMRegisterApp** call.

lpszDocId - in - A null-terminated document ID. This is typically obtained by a call to **ODMSelectDoc** or **ODMNewDoc**. The specified document may or may not be open when **ODMSetDocInfo** is called.

item - in - A single Item Id.

Refer to Appendix A for a complete list of Item Ids for document attributes which can be specified in this parameter.

ODM_DMS_DEFINED - Can be used for DMS specific attributes not explicitly defined by ODMA. The *lpszData* parameter contains a DMS-specific indication of the data being passed as well as the data itself. Note that an application must know which DMS it is talking to and must understand the data indications supported by the DMS in order to use this item id.

lpszData - in - Pointer to the data being passed to the DMS. Must be null-terminated. If *item* is **ODM_DMS_DEFINED** then the value in this parameter must contain both an indication of the attribute to set and the data value. It is recommended that the DMS be able to recognize a null-terminated string containing one of its known attribute names, followed by an equal sign delimiter character, followed by the data value.

Return value:

ODM_SUCCESS if successful.

ODM_E_ACCESS if the user does not have the appropriate access rights to set the specified document attribute.

ODM_E_DOCID if the document ID is invalid or refers to a document that no longer exists.

ODM_E_OFFLINE if the DMS cannot currently access the document because the user client is off-line.

ODM_E_ITEM if *item* is invalid or unknown to this DMS.

ODM_E_NOSUPPORT if the DMS does not support the function or does not support setting the specified document attribute.

ODM_E_HANDLE if *handle* was invalid.

ODM_E_FAIL if the specified data was invalid or the DMS was unable to accept it for other reasons.

ODMSetDocRelation

ODMSTATUS ODMSetDocRelation(ODMHANDLE handle, LPSTR lpszDocId, LPDWORD pdwFlags, LPSTR lpszLinkedId, LPSTR lpszFormat, LPSTR lpszPreviousId)

An application can use this function to maintain relationship and hierarchy information in the DMS about a compound document structure that it is manipulating external to the DMS. This function would typically be used as part of saving a new or updated version of the master document, or when the user requests the application to review and update the version that has been linked to.

Parameters:

handle - in - A handle obtained by a previous **ODMRegisterApp** call.

lpszDocId - in - A null-terminated document ID. This is typically obtained by a call to **ODMSelectDoc** or **ODMNewDoc**. The specified document may or may not be open when **ODMSetDocRelation** is called.

pdwFlags - in/out - A pointer to variable containing flags used on both input and output. On input, one or more of the following flags:

ODM_REL_PARENT - Create a link between the documents represented in *lpszDocId* and *lpszLinkedId* with *lpszDocId* as the parent. This flag cannot be used in combination with **ODM_REL_CHILD** or **ODM_REL_NONE**.

ODM_REL_CHILD - Create a link between the documents represented in *lpszDocId* and *lpszLinkedId* with *lpszDocId* as the child. This flag cannot be used in combination with **ODM_REL_PARENT** or **ODM_REL_NONE**.

ODM_REL_NONE - If a relationship exists between the documents represented by *lpszDocId* and *lpszLinkedId* then delete it. This flag cannot be used in combination with **ODM_REL_PARENT** or **ODM_REL_CHILD**.

ODM_REL_FIXED - The relationship is frozen between the particular document versions represented by *lpszDocId* and *lpszLinkedId*. This may also be used to signify that the application and/or the user want to retain control over updating the link, rather than have the DMS do it.

ODM_REL_RELEASED - The link should be maintained to the Current or Released version of the child document. Released is available in some DMS to distinguish a version that has been approved for release. If the DMS does not distinguish between Released and Latest, then Latest is used. The value is called Released and not Current, to avoid confusion with the version that is already (currently) linked.

ODM_REL_LATEST - The link should be maintained to the latest version of the child document.

ODM_SILENT - The DMS should not require user interaction while satisfying the call. If the call cannot be satisfied without user interaction then an error should be returned.

ODMA 2.0 does not define any output flags at this time.

lpszLinkedId - in - A null-terminated document ID. This is typically obtained by a call to **ODMSelectDoc**, **ODMNewDoc**, **ODMSaveAs**, **ODMSaveDoc**, **ODMGetDocRelation** or **ODMGetDocInfo**. The specified document may or may not be open when **ODMSetDocRelation** is called.

lpszFormat - in/out - This is an optional parameter. If specified it must be a pointer to a buffer of at least **ODM_FORMAT_MAX** characters. This buffer contains a null-terminated string naming the file format of the rendition to be retrieved when the link is followed. Refer to the Document Format Names section for information on how file formats are identified in ODMA. This parameter allows the caller to identify the best format for the intended use of the main document. For instance, a graphic may be saved in several resolutions, one for on-line use and another for printing.

On input, this is a hint to the DMS of the preferred format of the child to be linked to. If the application does not provide a format it should set the buffer in *lpszFormat* to a zero length string. In this case the DMS should use the primary format of the document.

On output, the DMS should write the format name of the rendition it used. This may be the closest rendition format the DMS could find. If the DMS does not record rendition formats when relating documents, then it should return a zero length string in this buffer.

If this parameter is Null, then the DMS should use the primary format of the child document (the one used in **ODMOpenDoc**).

lpszPreviousId - in - A null-terminated document ID. When adding child documents, if the DMS maintains positional information in a logically ordered list of linked documents, then the application is requesting the DMS to add *lpszLinkedId* after *lpszPreviousId* and re-link whatever was the "next Id" now to *lpszLinkedId*. To add *lpszLinkedDocId* as the first document in a list, provide a zero length, null-terminated document ID. Null should be passed if the application has no meaningful information to pass through this parameter, or to add *lpszLinkedDocId* as the last document in the list.

Return value:

ODM_SUCCESS if successful.
ODM_W_NOACTION if the link already existed and was configured in the requested manner.
ODM_E_NORELATION if the application requests deletion of a relationship that does not exist.
ODM_E_ACCESS if the user does not have the appropriate access rights to create, update or delete this document relationship.
ODM_E_INUSE if the user is currently unable to create, update or delete this document relationship because one or more of the related documents is checked-out by another user.
ODM_E_DOCID if a document ID is invalid or refers to a document that no longer exists.
ODM_E_USERINT if the **ODM_SILENT** flag was specified and the DMS could not satisfy the call without user interaction.
ODM_E_NOSUPPORT if the DMS does not support the function.
ODM_E_INVARG if the value specified in *flags* was invalid.
ODM_E_REQARG if *lpszDocId* or *lpszLinkedId* is not specified.
ODM_E_HANDLE if *handle* was invalid.
ODM_E_FAIL if the specified data was invalid or the DMS was unable to accept it for other reasons.

ODMUnRegisterApp

void ODMUnRegisterApp(ODMHANDLE handle)

An application that previously registered itself with a DMS via **ODMRegisterApp** should call this function when it is finished using the DMS. This would typically be done when the application is shutting down. After this call returns, the DMS session handle is no longer valid and cannot be used for subsequent calls to ODMA functions.

Parameters:

> *handle* - in - A handle obtained by a previous **ODMRegisterApp** call.

Return value:

> None.

DMS Interface

A DMS interfaces with ODMA through a function called **ODMGetODMInterface** and through three COM interfaces; **IODMDocMan**, **IODMQuery** and **IODMDocMan2**. When an application calls the ODMA connection manager's **ODMRegisterApp** function, the connection manager calls the appropriate DMS's **ODMGetODMInterface** function to establish a connection with the DMS.

The **IODMDocMan** interface implements all of the basic document management functionality defined by the ODMA 1.0 specification. The **IODMQuery** interface implements the query extension of the ODMA 1.5 specification. The **IODMDocMan2** interface implements the extended document management functionality defined in the ODMA 2.0 specification. The methods of these interfaces each correspond directly with a function from the **ODMA API** section of this document. (For example, the **SelectDoc** method of the **IODMDocMan** interface corresponds to the **ODMSelectDoc** function.) With only one exception, the functionality of each method is completely detailed by the description of its corresponding **ODMA API** function. For that reason, this section of the document only provides a list of each method with its prototype and no further details. (The one exception is the **QueryExecute** method of the **IODMQuery** interface. Additional details, only important to the DMS's implementation of this method, are provided below.)

ODMGetODMInterface

HRESULT ODMGetODMInterface(REFIID riid, LPVOID FAR *ppvObj, LPUNKNOWN pUnkOuter, LPVOID pReserved, LPSTR lpszAppId, DWORD dwEnvData)

The ODMA connection manager calls this function to get an interface from a DMS. This function is **not** available to ODMA-aware applications. The DMS returns its status using OLE HRESULT codes which the Connection Manager then maps to ODMA status codes.

Parameters:

riid - in - ID of the interface being requested. This will typically be **IID_IODMDocMan**, which all ODMA DMS providers must support, or other ODMA interfaces. The Connection Manager may also pass through calls for other interfaces from the client.

ppvObj - out - The requested interface should be returned here. If the DMS cannot return the requested interface, *ppvObj* should be set to Null.

pUnkOuter - in - The controlling IUnknown interface for the aggregate object. ODMA providers **must** support aggregation.

pReserved - in - Reserved for future use. It must always be set to Null.

lpszAppId - in - The ID of the calling application. See **ODMRegisterApp**.

dwEnvData - in - Environment specific data. On Windows platforms this will be a window handle from the calling application. See **ODMRegisterApp**.

Return value:

S_OK if successful.
E_NOINTERFACE if the requested interface is not supported.
E_ACCESSDENIED if the DMS refuses the application specified in *lpszAppId*. The Connection Manager will return **ODM_E_REFUSED** to the calling application.
E_FAIL if the DMS fails to initialize itself. The Connection Manager will return **ODM_E_CANTINIT** to the calling application.
E_ODM_VERSION if the DMS does not support the requested version of the API.

IODMDocMan Interface

SelectDoc

STDMETHOD_(ODMSTATUS, SelectDoc) (THIS_ LPSTR lpszDocId, LPDWORD pdwFlags) PURE;

OpenDoc

STDMETHOD_(ODMSTATUS, OpenDoc) (THIS_ DWORD flags, LPSTR lpszDocId, LPSTR lpszDocLocation) PURE;

SaveDoc

STDMETHOD_(ODMSTATUS, SaveDoc) (THIS_ LPSTR lpszDocId, LPSTR lpszNewDocId) PURE;

CloseDoc

STDMETHOD_(ODMSTATUS, CloseDoc) (THIS_ LPSTR lpszDocId,

DWORD activeTime, DWORD pagesPrinted, LPVOID sessionData,
WORD dataLen) PURE;

NewDoc

STDMETHOD_(ODMSTATUS, NewDoc) (THIS_ LPSTR lpszDocId,
DWORD dwFlags, LPSTR lpszFormat, LPSTR lpszDocLocation) PURE;

SaveAs

STDMETHOD_(ODMSTATUS, SaveAs) (THIS_ LPSTR lpszDocId,
LPSTR lpszNewDocId, LPSTR lpszFormat,
ODMSAVEASCALLBACK pcbCallBack, LPVOID pInstanceData) PURE;

Activate

STDMETHOD_(ODMSTATUS, Activate) (THIS_ WORD action,
LPSTR lpszDocId) PURE;

GetDocInfo

STDMETHOD_(ODMSTATUS, GetDocInfo) (THIS_ LPSTR lpszDocId,
WORD item, LPSTR lpszData, WORD dataLen) PURE;

SetDocInfo

STDMETHOD_(ODMSTATUS, SetDocInfo) (THIS_ LPSTR lpszDocId,
WORD item, LPSTR lpszData) PURE;

GetDMSInfo

STDMETHOD_(ODMSTATUS, GetDMSInfo) (THIS_ LPSTR lpszDmsId,
LPWORD pwVerNo, LPDWORD pdwExtensions) PURE;

GetLeadMoniker

STDMETHOD_(ODMSTATUS, GetLeadMoniker) (THIS_ LPSTR lpszDocId,
LPMONIKER FAR *ppMoniker) PURE;

IODMQuery Interface

QueryExecute

STDMETHOD_(ODMSTATUS, QueryExecute) (THIS_ LPCSTR lpszQuery,
LPSTR dmsQueryId) PURE;

Additional Notes: Upon success, the string returned in *dmsQueryId* should be of the form: "::ODMA\DM_ID\DMS_SPECIFIC_QUERY_ID>". Note that this query ID is not necessarily the one returned to the client application. Note also that multiple query IDs from different DMSs will be mapped to the same query ID for a single query call from the client application. The *dmsQueryId* string may be up to **ODM_DOCID_MAX** bytes in length.

The return value of **ODM_E_PARTIALSUCCESS** is a valid return value for **ODMQueryExecute** but is not a valid return value for this method.

QueryGetResults

STDMETHOD_(ODMSTATUS, QueryGetResults) (THIS_ LPCSTR queryId, LPSTR lpszDocId, LPSTR lpszDocName, WORD docNameLen, LPWORD pwDocCount) PURE;

QueryClose

STDMETHOD_(ODMSTATUS, QueryClose)(THIS_ LPCSTR dmsQueryId) PURE;

IODMDocMan2 Interface

CloseDocEx

STDMETHOD_(ODMSTATUS, CloseDocEx) (THIS_ LPSTR lpszDocId, LPDWORD pdwFlags, DWORD activeTime, DWORD pagesPrinted, LPVOID sessionData, WORD dataLen) PURE;

SaveAsEx

STDMETHOD_(ODMSTATUS, SaveAsEx) (THIS_ LPSTR lpszDocId, LPSTR lpszNewDocId, LPSTR lpszFormat, ODMSAVEASCALLBACK pcbCallBack, LPVOID pInstanceData, LPDWORD pdwFlags) PURE;

SaveDocEx

STDMETHOD_(ODMSTATUS, SaveDocEx) (THIS_ LPSTR lpszDocId, LPSTR lpszNewDocId, LPDWORD pdwFlags) PURE;

SelectDocEx

STDMETHOD_(ODMSTATUS, SelectDocEx) (THIS_ LPSTR lpszDocIds, LPWORD pwDocIdsLen, LPWORD pwDocCount, LPDWORD pdwFlags, LPSTR lpszFormatFilter) PURE;

QueryCapability

STDMETHOD_(ODMSTATUS, QueryCapability) (THIS_ LPCSTR lpszDmsId, DWORD function, DWORD item, DWORD flags) PURE;

SetDocEvent

STDMETHOD_(ODMSTATUS SetDocEvent) (THIS_ LPSTR lpszDocId, DWORD flags, DWORD event, LPVOID lpData, DWORD dwDataLen, LPSTR lpszComment) PURE;

GetAlternateContent

STDMETHOD_(ODMSTATUS GetAlternateContent) (THIS_ LPSTR lpszDocId, LPDWORD pdwFlags, LPSTR lpszFormat, LPSTR lpszDocLocation) PURE;

SetAlternateContent

STDMETHOD_(ODMSTATUS SetAlternateContent) (THIS_ LPSTR lpszDocId, LPDWORD pdwFlags, LPSTR lpszFormat, LPSTR lpszDocLocation) PURE;

GetDocRelation

STDMETHOD_(ODMSTATUS GetDocRelation) (THIS_ LPSTR lpszDocId, LPDWORD pdwFlags, LPSTR lpszLinkedId, LPSTR lpszFormat, LPSTR lpszPreviousId) PURE;

SetDocRelation

STDMETHOD_(ODMSTATUS SetDocRelation) (THIS_ LPSTR lpszDocId, LPDWORD pdwFlags, LPSTR lpszLinkedId, LPSTR lpszFormat, LPSTR lpszPreviousId) PURE;

Binding to the API

Windows 3.x, Windows 95 and Windows NT

The ODMA connection manager, ODMA.DLL (Windows 3.x) or ODMA32.DLL (Windows 95 and NT), and a corresponding link library, ODMA.LIB (Windows 3.x) or ODMA32.LIB (Windows 95 and NT), will be made freely available to any application vendor wishing to use them. Applications can choose either of the methods described below to bind to ODMA:

1. Link to ODMA.LIB/ODMA32.LIB. Applications choosing this approach will need to install a copy of ODMA.DLL/ODMA32.DLL into the Windows system directory at the time the application is installed unless a newer copy of ODMA.DLL/ODMA32.DLL already resides there. At startup time the application can call **ODMRegisterApp** to determine whether or not an ODMA provider is present for the calling application.

2. Dynamically load ODMA.DLL/ODMA32.DLL at startup time with a call to LoadLibrary. With this approach the application has to call GetProcAddress to get a pointer to each ODMA function it will call, but the application does not have to do anything related to ODMA at installation time. If ODMA.DLL/ODMA32.DLL doesn't exist or if **ODMRegisterApp** returns **ODM_E_NODMS** then the application knows that no ODMA provider is present for the calling application.

There is currently no thunk layer that exists to allow a 32-bit application and a 16-bit DMS, or vice versa, to interact with each other.

Installing a DMS

Windows 3.x, Windows 95 and Windows NT

The ODMA connection manager, ODMA.DLL for Windows 3.x and ODMA32.DLL for Windows NT and Windows 95, will be provided to all DMS vendors that want to support the ODMA specification. These DLLs will serve as a common entry point for applications that wish to call ODMA functions. Its implementation of **ODMRegisterApp** will figure out which DMS is registered for default use by the calling application. It will then load the appropriate DMS's DLL and pass through the function call. All other ODMA function calls that do not include a document ID will be passed to this default DMS.

When an ODMA call is made that includes a document ID, the ODMA connection manager will determine which DMS created the ID. If that DMS has not been initialized yet by the calling application, the connection manager will call the provider's **ODMGetODMInterface** function and will then pass through the current ODMA function call.

As part of its installation, a DMS should check whether a copy of ODMA.DLL and/or ODMA32.DLL already exists in the Windows System directory. (On Windows NT the ODMA32.DLL belongs in the System32 directory.) If the DLLs do not exist or if the DMS has newer versions then it should copy its version of ODMA.DLL and/or ODMA32.DLL to the Windows System directory. As of ODMA 2.0, the Connection Manager DLLs will contain version resource information. Next the DMS should register with the ODMA connection manager by adding one or more keys to the registration database.

Each DMS must add a subkey to the root level ODMA key. This subkey should be named the same as the DMS's DMS ID, and the subkey's value should specify the location of the DLL containing the DMS's ODMA entry points. For example, suppose a DMS with a DMS ID of DDD implemented its ODMA entry points in H:\ABC\XYZ.DLL. It would add the following key and value to the registration database under the HKEY_CLASSES_ROOT key:

ODMA\DDD = H:\ABC\XYZ.DLL	for Windows 3.x
ODMA32\DDD = H:\ABC\XYZ.DLL	for Win32 (i.e. WinNT & Windows 95)

The DMS ID is limited to 8 characters and is often cryptic. The DMS must add a FullName subkey under its DMS ID subkey which provides the full name of the DMS. The DMS is free to include any information necessary in the FullName (i.e. vendor name, version and platform). For example, our fictional DMS with the DDD DMS ID would define FullName like this:

ODMA\DDD\FullName = “Dynamic Document Depot V1.0 for Windows”
ODMA32\DDD\FullName = “Dynamic Document Depot V1.0 for Windows”

Applications should never display a DMS ID in a user interface. These DMS IDs can be obtained using **ODMGetDMSList** or directly reading the registration database. Applications should look up the FullName of the DMS and display that instead.

If the DMS wanted to serve as the default ODMA provider to be used with all ODMA-aware applications that were not registered with a specific DMS, it would add the DEFAULT subkey to its DMS ID key as shown below:

ODMA\DDD\DEFAULT	for Windows 3.x
ODMA32\DDD\DEFAULT	for Win32 (i.e. Windows NT and Windows 95)

If the DMS wanted to specifically register itself as the ODMA provider for one or more applications, it would also add a subkey to the relevant applications' root level keys. Note that this is only necessary if a different DMS is registered as the default ODMA provider. Each time an application calls **ODMRegisterApp**, the connection manager looks in the registration database for a root-level key that matches the specified application ID. If it finds such a key then it looks under it for a subkey called ODMA. The value of this key is the DMS ID of the specifically registered DMS. If the application ID key is not found or if the subkey ODMA or ODMA32 does not exist under it, the connection manager looks at the subkeys of the root-level key ODMA or ODMA32 for a default ODMA implementation.

For example, suppose a document management system with a DMS ID of DDD wanted to support only one particular application that used QQQ as its application ID. It would add the following key and value to the registration database in addition to the ODMA subkey described above:

QQQ\ODMA = DDD	for Windows 3.x
QQQ\ODMA32 = DDD	for Win32 (i.e. Windows NT and Windows 95)

Installing a Client Application

Windows 3.x, Windows 95 and Windows NT

This section defines the registration database entries ODMA compliant client applications should make when they are installed. These entries will help provide improved out-of-the-box integration between applications and Document Management Systems. They will allow an application to:

1. Enumerate 16-bit and 32-bit ODMA compliant applications on the system,
2. Enumerate file types whose native applications are ODMA compliant, and
3. Quickly determine if a given file type's native application supports ODMA and if it expects a 16-bit or 32-bit DMS provider.

Most Document Management and Workflow Management applications have a function that can push a document to the native application which supports the document's content format. The Windows platform provides several functions which can be used to launch an application to open a document. Examples are **ShellExecute**, **WinExec** and **CreateProcess** (Win32 only). However, the calling application needs to be able to determine in a seamless way if it should be passing a file specification or an ODMA Document ID to the application. For this reason, applications which support an ODMA Document ID in their command line or DDE interface must record this information in the registration database.

In addition to the above, the calling application may need to know if the ODMA application to be launched is 16-bit or 32-bit. For example, if the calling DMS application is only 16-bit and the application to be launched is 32-bit, then perhaps the calling application can downgrade to some other form of application invocation that doesn't required a 32-bit DMS.

File type entries take the following form, under the HKEY_CLASSES_ROOT key:

ODMA.FileTypes\<FileType> = "<AppSubkey>"
ODMA32.FileTypes\<FileType> = "<AppSubkey>"

<FileType> is the same value used in the HKEY_CLASSES_ROOT\.ext entry for the file extension. Sometimes this value is referred to as a file type class name. This provides a link back to the file type's other registration entries. Note that often applications have more than one file extension pointing to a single file type class name.

<AppSubkey> is the same as the ODMA application's AppId, used in **ODMRegisterApp**. If the AppId contains spaces they must be removed, because spaces are not allowed in a subkey.

Application entries are as follows (under the HKEY_CLASSES_ROOT key):

ODMA.ClientApps\<AppSubkey> = "<AppId>"
ODMA32.ClientApps\<AppSubkey> = "<AppId>"

The <AppSubkey> is the same as above. The <AppId> is the regular string used by the application in **ODMRegisterApp**. It can contain spaces and is limited to 15 characters (see the **ODM_APPID_MAX** constant).

The FullName of a client application can be provided as follows:

ODMA.ClientApps\<AppSubkey>\FullName = "The app name"
ODMA32.ClientApps\<AppSubkey>\FullName = "The app name"

Use the following procedure to determine if the application that supports a given file type can also handle an ODMA document ID in its command line or DDE interface. First lookup the file extension entry in the Registry to get the file type class name. If the extension isn't found then there is no application supporting the file type on the system. Next append the file type class name to the ODMA32.FileTypes\ key and see if the key exists in the registry. If the key exists, then a 32-bit ODMA-aware application that can accept an ODMA document ID in its command line or DDE interface should be present on the system. If the key was not found then the ODMA.FileTypes key can be tested to determine if a 16-bit ODMA-aware application is present. If a 16-bit DMS client is initiating this process, it can avoid launching a 32-bit application if it doesn't have the correct thunking layers installed on the platform. The reverse is also true, where a 32-bit DMS might avoid launching a 16-bit application if it does not support 16-bit ODMA on the platform.

Connection Manager Trace Logging

Windows 3.x, Windows 95 and Windows NT

As of ODMA 2.0, the ODMA Connection Manager is capable of logging function calls made by any application to any DMS on the system. This trace log contains information about each function call made, including the App ID of the calling application and the input values of individual function parameters. It also includes the information returned by the DMS, like function values and output parameter values. The trace log contains information about function calls that are handled by the Connection Manager without DMS involvement (i.e. **ODMGetDMS**, **ODMSetDMS**, **ODMGetDMSList**, etc.).

The following Windows Registry sub-keys, under HKEY_CLASSES_ROOT, control ODMA trace logging:

ODMA.ConnectionManager\Logging\Enabled
ODMA32.ConnectionManager\Logging\Enabled

If the entire sub-key described above is defined then the Connection Manager will do trace logging. To disable trace logging simply remove the "Enabled" sub-key. There are no default data names or values assigned to this sub-key at this time.

ODMA.ConnectionManager\Logging\File = "fullpath"
ODMA32.ConnectionManager\Logging\File = "fullpath"

The default filename for the trace log file is "ODMA.LOG" for the 16-bit Connection Manager and "ODMA32.LOG" for the 32-bit Connection Manager. The default path is the Windows System or System32 directory as appropriate. If the File sub-key does not exist then ODMA uses its defaults. If the fullpath is specified in the data value it must contain a full path and filename. If the fullpath is invalid the Connection Manager will revert to its default log filename and path.

ODMA.ConnectionManager\Logging\Options = "token1,token2"

ODMA32.ConnectionManager\Logging\Options = “token1,token2”

The Options data value is simply a comma separated string of tokens. There is no default value defined at this time. The various tokens control trace logging. They are:

> NEWLOG -- If defined then when the Connection Manager DLL is loaded into memory it will delete any existing log file and create a new log file. The default behavior is for the Connection Manager to append to any existing log file.

Appendix A

Document Attributes

The following table describes document attributes or properties defined by ODMA. Applications can get attribute values for a document by using the **ODMGetDocInfo** function. Some attribute values can be set using the **ODMSetDocInfo** function. Not every DMS will support every document attribute listed here. Applications can use the **ODMQueryCapability** function to determine if a DMS supports a given document attribute.

Note that the ODMA standard does not currently define maximum lengths for any specific document attribute. This is DMS specific. In most cases the DMS will truncate the attribute value returned if it does not fit in the caller supplied buffer.

Attribute/Item ID	Description	Type	Set
ODM_ALTERNATE_RENDERINGS	If the DMS supports alternate renderings for the specified document, then it must return a comma-separated list of format names representing the alternate formats it can return in **ODMGetAlternateContent**. Refer to the Document Format Names section for information on how file formats are identified in ODMA.	String	No
ODM_AUTHOR	Author of the document.	String	Yes
ODM_CONTENTFORMAT	The format name string indicating the format of the document's contents. Refer to the Document Format Names section for information on how file formats are identified in ODMA. In **ODMGetDocInfo** It is recommended that the DMS return either a MIME Content Type or a Windows file type/extension. Note that when this item is used in **ODMSetDocInfo**, it merely informs the DMS of a change in the document's format; it does not cause a conversion process to take place within the DMS.	String	Yes
ODM_CHECKEDOUTBY	Username of the person who has the document checked-out. This username is in the form used by the DMS. If the document is not currently checked-out a zero length string is returned.	String	No
ODM_CHECKOUTCOMMENT	The comment, if any, that the user supplied when they checked-out the document.	String	No
ODM_CHECKOUTDATE	The date/time the document was checked-out. If the document isn't currently checked-out, a zero-length string is returned.	Date	No
ODM_CREATEDBY	Username of the person who initially created the document in the DMS.	String	No
ODM_CREATEDDATE	The date/time the document was created in the DMS.	Date	No
ODM_DOCID_LATEST	A null-terminated document ID for the Latest version of the document represented by *lpszDocId*. If *lpszDocId* is the Latest version, it will be the same document ID.	DocId	No

Attribute/Item ID	Description	Type	Set
ODM_DOCID_RELEASED	A null-terminated document ID for the Released version of the document represented by *lpszDocId*. If *lpszDocId* is the Released version, it will be the same document ID. If the DMS does not differentiate Released versions, return the version ID of the Latest version of the document. A DMS may allow this attribute to be set. An application such as a workflow can set a version of a document to be the released version by specifying this attribute in a call to **ODMSetDocInfo**. In this case, the *lpszData* parameter should be set to Null.	DocId	Yes
ODM_DOCVERSION	A null-terminated string which is the version ID for *lpszDocId*. This allows the application to display the version information explicitly (without doing the impossible and decoding the document ID), for situations such as where the user is making the decision to re-link to a different version. It is the DMS responsibility to consistently translate version information into a string that can be used for comparison and display. Since it is a string, and comparisons will not be made between strings from different DMS, the application does not have to be concerned that different DMS have different formats, meanings and capabilities for version, sub-version, branching and so on.	String	No
ODM_DOCVERSION_LATEST	A null-terminated string which is the version ID for the Latest version of the document represented by *lpszDocId*.	String	No
ODM_DOCVERSION_RELEASED	A null-terminated string which is the version ID for the Released version of the document represented by *lpszDocId*. If the DMS does not differentiate Released versions, return the version ID of the Latest version of the document.	String	No
ODM_LOCATION	A null-terminated string containing DMS specific information describing the logical location (e.g. folder) of the document within the data store. The DMS should format the information in this attribute so that it can be safely displayed in a user-interface. The application should not assume the information in this attribute is persistent across ODMA sessions. A DMS may allow an application to set this attribute in **ODMSetDocInfo** only for documents that have not been created yet.	String	Yes
ODM_KEYWORDS	A comma separated list of keywords assigned to the document.	String	Yes
ODM_LASTCHECKINBY	Username of the last person to modify or check-in the document.	String	No
ODM_LASTCHECKINDATE	The date/time the document was last modified or checked-in.	Date	No
ODM_MODIFYDATE	The data/time the document was last modified.	Date	No
ODM_MODIFYDATE_LATEST	The date/time that the Latest version of the document was last modified.	Date	No
ODM_MODIFYDATE_RELEASED	The date/time that the version of the document that is Released was last modified. If the DMS does not differentiate Released versions, return the date/time that the Latest version of the document was last modified.	Date	No

Attribute/Item ID	**Description**	**Type**	**Set**
ODM_NAME	Name of the document. This is a descriptive name for the document, not the filename.	String	Yes
ODM_OWNER	The owner of the document.	String	No
ODM_SUBJECT	The subject matter of the document.	String	Yes
ODM_TITLETEXT	Suggested text to display in the document window's title bar. This may include one or more fields from the document's profile and possibly other information as well.	String	No
ODM_TITLETEXT_RO	Same as ODM_TITLETEXT, except if the document is currently opened by the calling application in **ODM_VIEWMODE**, the DMS should append " (Read-Only)" to the text.	String	No
ODM_TYPE	Type of the document. This is typically an indication of the format or content of the document, i.e. correspondence, memo, contract, etc.	String	Yes
ODM_URL	Uniform Resource Locator (URL) which provides an alternative way to access the document. Note that a DMS can never truncate a URL if it is too long to fit into the caller supplied buffer.	String	No

String = A null-terminated string containing the document attribute value.

Date = A null-terminated string containing a date and time in the following format: “YYYYMMDDhhmmsscc +hhmm”. The “YYYYMMDD” section contains a 4 digit year, 2 digit month and 2 digit day. The “hhmmsscc” section contains the time in 24 hour format (hours, minutes, seconds, hundredths of seconds). A DMS should return zeros for any part of a time it doesn’t support. The “+hhmm” section is an optional Greenwich Mean Time (GMT) offset. If the GMT string is included in a date/time string it is preceded by a space, followed by either a + (plus) or - (minus) character, then a 2 digit hour value and a 2 digit minute value. A DMS should only return the GMT offset if it knows the offset for the data store containing the specified document. If the GMT offset is not included, then the date/time value is the local date and time for the document in the DMS.

DocId = A null-terminated string containing an ODMA document ID.

The “Set” column indicates if the attribute can be set with **ODMSetDocInfo**. It is possible that a DMS may not allow certain document attributes to be set.

Appendix B

Preferred Call Usage for Initial File Creation

There has been some misunderstandings among both DMS and application vendors on the correct use of the ODMA 1.0 calls to initially create a file. This appendix is basically from a proposal made in October 1994 by Rod Schiffman of Novell. There has been no dissension on the issue since the original proposal was made. Until the standard is modified to provide other methods for the initial creation of a new document in a DMS, this provides a sequence for the calls that will work with all major DMS vendors.

From Rod Schiffman:

It is probably useful for me to describe the terminology and concepts I will use in this discussion. A DMS is fundamentally more complex than my description, but it will suffice for my purposes. A DMS is a black box that manages documents. My Local System is the particular machine and Operating System, or environment, that I use. It has its own file system, applications and interfaces. When I need to do something to a existing document in the DMS, I select it using tools the DMS provides and retrieve it into my local system. I use an application to perform operations on the document. I may want to create a new document with my application. The new document will be stored on my local system until I save it into the DMS. The DMS may want to tell me where I should save it on the Local System until I eventually save it into the DMS. I may also want to update the copy in the DMS while I am still working on it. I call this writing the document through to the DMS. When I am finally done working on the document, I can save it to the DMS. It is then closed. At some point in the process of creating a new document, the DMS will require me to give it a profile. When I have saved the document to the DMS, it is closed and the document no longer has a presence on my local machine. ODMA has a vaguely defined concept of formats that are attached to a document, and while the timing of when and how formats are defined plays a central role in this discussion, formats, conversions and their association to documents does not play a role in this discussion.

ODMSaveDoc is the only ODMA function that actually saves a document into a DMS. It uses a Document ID (DocID) supplied by the DMS. The initial creation of the first DocID for the document can only be supplied by **ODMNewDoc**. **ODMSaveAs** and **ODMSaveDoc** can return a new DocID, but they must already have an initial DocID to start with. **ODMSelectDoc** only returns a DocID that already exists. This means that **ODMNewDoc** must be called to generate the initial DocID. After the application has a DocID, it can call **ODMOpenDoc** to get a local file name to save the document to and **ODMSaveDoc** to write the document through to the DMS. This would imply that it is not necessary to call **ODMSaveAs** in the process of saving the document the first time. This has caused a number of people to assume that the usage of **ODMNewDoc** was synonymous with the initial “Save” function in most applications. However, the spec says that the format specified in **ODMNewDoc** “may be changed later via an **ODMSaveAs** call.” This implies that it is OK to call **ODMSaveAs** after calling **ODMNewDoc**. **ODMSaveAs** is the only call that allows an application to interact with the DMS and supply application specific information such as formats and passwords. If an app only calls **ODMNewDoc** and does not call **ODMSaveAs**, it loses the ability to provide application specific information. **ODMSaveAs** requires that a user interface is supplied to the user. If **ODMNewDoc** also supplies a user interface, then the user interface comes up twice for each document creation. Once in **ODMNewDoc** and again in **ODMSaveAs**. This is probably not something that the user wants. Also, how do we reconcile the ability to call **ODMNewDoc** without a user interface. If **ODMNewDoc** is called with no user interface and a DocID is returned, then when is the profile filled in? **ODMOpenDoc** and **ODMSaveDoc** and **ODMCloseDoc** have no obvious provisions for providing a user interface.

ODMNewDoc was originally intended to be synonymous with the “New” menu choice, not the “Save” menu choice, in most applications. For example, although SoftSolutions and PC Docs do not care about a document until it is time to save it to the DMS the first time, DEC TeamLinks does have the concept of letting the DMS know a document is going through initial creation. There is nothing in the spec that indicates whether **ODMNewDoc** should be called at document creation or just before saving the document the first time. This is not an accident according to Brad Clements, the original author of the ODMA spec. It can be done at whatever time makes sense for the DMS and application. **ODMNewDoc** does not have provisions for the callback routine because it notifies the DMS about the creation of a document not the saving of a document. At document creation time, the user is probably not worried about the what the final format will be or if there will be a password. The user may not, and this is crucial, even save the document to the DMS. It is possible that if **ODMNewDoc** is called without a user interface, that the DocID returned to the application may not eventually be used, if the user is allowed to have the file saved to the local file system in a later call to **ODMSaveAs**.

It turns out that many applications and Document Management Systems do not have or need a concept of providing application and DMS interaction until the document is ready to be saved in the DMS for the first time. Certainly a DMS must deal with applications that will not provide any interaction with the DMS until the document is already created in the edit buffer and is ready to be saved. At this point I would like to point out a scenario that follows the spec, and still works with major applications that may not have followed the spec according to the interpretation given here.

At the time of initial document creation, or when it is time to save the document the application should call **ODMNewDoc** with the ODM_SILENT flag. This will return a temporary DocID that the application can use. Technically, the application could call **ODMOpenDoc** to have a local temporary file to use until the document is finally saved into the DMS, but I know that every DMS vendor I have talked to would frown on this idea. They prefer the idea that all saves are written through to the DMS and that there is a full profile for anything that exists in the DMS. It is OK for the DMS vendor require this by returning an ODM_E_USERINT error to indicate that **ODMOpenDoc** cannot be called until the app makes a call to **ODMNewDoc** or **ODMSaveAs** that brings up an user interface. This implies that the application would be better to avoid the situation and not call **ODMOpenDoc** until after an user interface is provided to the user. After calling **ODMNewDoc** in silent mode, take the DocID that is returned and call **ODMSaveAs** to have the profile created, and allow access to the callback routines. If the **ODMNewDoc** returns an ODM_E_USERINT error because it does not support the silent mode, then call **ODMNewDoc** with a user interface and skip calling **ODMSaveAs**. This performs the same functionality, but does not provide access to the callback routines. If the application still wants the user to see the callback dialog, then it can call the callback dialog itself after calling the **ODMNewDoc** call. This is not optimal, but does work. In either case the user interface is only presented to the user one time.

There are a couple caveats for DMS vendors. The DocID returned by **ODMNewDoc** must be a temporary ID. It is possible that the user will throw away the file before it is saved, or that it will be saved to the native OS through the options button in the SaveAs dialog. This could be easily handled by returning a well known dummy value like ::ODMA\<app id>\NULL. When **ODMSaveAs** sees this null value as the DocID, it can know that it is creating the document for the first time and take appropriate action as opposed to the normal SaveAs action. It does not matter if there are multiple documents using this DocID at the same time as long as the DMS returns ODM_E_USERINT to the **ODMOpenDoc** call that passed in the null DocID.

Basically, this turns the **ODMNewDoc** into a fancy NOP routine. There may be situations where this may not be the best action, but since there is no compelling reason for the existence of the **ODMNewDoc** call in the 1.0 spec, it seems to solve more problems than it creates. It would probably be a good idea to add an extension to the 1.0 spec in ODMA 1.1 where the **ODMSaveAs** call can allow a NULL to be passed in as the lpszDocID parameter to indicate that the document is being saved for the first time. This would free the application from having to use the **ODMNewDoc** call. Of course, that is what the addition of a dummy DocID value returned from **ODMNewDoc** accomplishes. In the ODMA 1.1 timeframe a new **ODMNewDoc** call can be created if someone comes up with a compelling reason for its existence. Simply using the **ODMSaveAs** call when an application is creating a new file in the DMS, whether it is the first save or an actual SaveAs, matches the way most applications work today.

Additional information from Bob St.Jean:

ODMA 2.0 introduces an **ODMSaveAsEx** function that extends **ODMSaveAs**. This addresses the specification change Rod Schiffman had envisioned. The *lpszDocId* parameter in **ODMSaveAsEx** can optionally be set to Null by the calling application. This frees the application from calling **ODMNewDoc** during a Save As sequence for a new document. If the application calls **ODMSaveAsEx** without first calling **ODMNewDoc**, then the DMS will return a DocID in the *lpszNewDocId* parameter. This may be a temporary DocID that will later be replaced by a new DocID when **ODMSaveDoc** or **ODMSaveDocEx** is called.

Note that there is a purpose for the **ODMNewDoc** call in the Save As sequence. The temporary DocID returned by the DMS in **ODMNewDoc** will provide a document context which the application can use to pre-set some document attributes for the new document before calling **ODMSaveAs** or **ODMSaveAsEx**. This is the only way for an application to pre-set items that might appear in the DMS's Save As dialog box. Examples of such attributes are ODM_AUTHOR, ODM_NAME, ODM_TYPE, ODM_KEYWORDS, ODM_SUBJECT and perhaps others. The application can set these attributes by calling **ODMSetDocInfo** using the DocID returned by **ODMNewDoc**. The application can call **ODMQueryCapability** to find out if the DMS will accept the attributes in **ODMSetDocInfo**. However, there is no way for an application to know if the attributes will be visible in the DMS's Save As dialog box.

The ODMA 2.0 specification also introduces a way to create documents without user interaction. The new **ODMSaveAsEx**, **ODMSaveDocEx** and **ODMCloseDocEx** functions all have an ODM_SILENT flag for this purpose. As outlined above, **ODMNewDoc** is used to create a temporary document context and **ODMSetDocInfo** is used to set document attributes.

Additionally, ODMA 2.0 recommends a standard for applications and DMSs to use to identify file format types.