Dennis E. Hamilton
NuovoDoc
4401 44th Ave SW
Seattle, WA 98116-4114
+1-206.932.6970
http://NuovoDoc.com

Open Document Management API

ODMA
AIIM

Guide & Usage Scenarios

Review Draft 0.20

ODMJNI 1.0: Java-ODMA Reference Integration

Guide & Usage Scenarios

ODMJNI 1.0:
Java-ODMA Reference Integration

1 Overview

REVIEW DRAFT: This is a preliminary version of the Guide & Usage Scenarios document for early review of feasibility and usage in conjunction with a pilot desktop application. The material is incomplete and tentative. This document will be revised as implementation experience is gained.

IMPORTANCE FOR EARLY ADOPTERS: This document need not be fully completed in order for ODMJNI to be placed in use. The intention is to have sufficient material in this document, the development progression notes, and working code to expedite early simple use in production. Non-critical refinements to this document will be made when ODMJNI is stable enough for possible adoption in other Java-based desktop applications.

IMPORTANT REVIEW POINT FOR EARLY ADOPTERS: It is important to ensure that the basic principles, especially those for Simple ODMA-Awareness, can be accommodated in the application being considered. It is especially important to review these principles in the case of an existing application to which ODMA-awareness is to be added.

The ODMJNI 1.0 Java-ODMA integration implements simple ODMA-aware operation for Java-based desktop applications.

ODMJNI 1.0 does not attempt to deliver the ODMA API in Java form. Instead, there are simple Java interfaces that support common desktop scenarios involving optionally-managed documents. The purpose is to provide smooth integration with the common document operations that desktop applications perform. Detailed understanding of ODMA operation is not required.

1.1 Simple ODMA-Awareness Principles

Simple principles apply for desktop document applications that implement the basic essentials of ODMA-aware operation.

1. The application provides some equivalent of a single- or multiple-document interface of the Microsoft Windows Platform.

    a. It uses typical/common menus and dialogs for making new documents and storing, retrieving, and manipulating documents that are stored in the local file system and other file systems that may be accessed through remote connections as if local files.

    b. The ODMA-awareness of Microsoft Office Word editions since Office 98 is the popular benchmark that all widely-used ODMA-compliant Document Management Systems accept with ease.

2. When no ODMA Connection Manager is found, or there is no ODMA DMS available for default use by the application, the software simply behaves as an ordinary ODMA-unaware desktop application, using menus and dialogs that are familiar for those purposes.

3. When the ODMA Connection Manager is present and there is an ODMA-compliant Document Management System (DMS) available for default use, the desktop application is expected to give the ODMA DMS **first refusal** in all operations that create, save, access, and update documents for retention and management by the DMS. The application automatically initiates operation with the ODMA DMS via the ODMA Connection Manager API (directly or via library support, such as that of ODMJNI).

4. As part of obtaining details and confirmation from the users of desktop applications, the ODMA DMS presents its own dialogs and information windows for the user. The user always has options to

    a. Cancel the operation requested of the application,

    b. Request that the application perform the operation instead of the DMS, or

    c. Use the DMS, possibly after an initial DMS sign-on and provision of other information requested by the DMS.

5. When the application is informed that the operation was cancelled, it should behave the same as its own file-oriented dialog had been cancelled.

6. When the application is informed that the application should perform its version of the operation, the application should do so, providing its own dialogs for the operation (e.g., Open, Save, or Save As).

7. When the application is informed that the DMS performed the operation, the application continues the same as it would after performing the operations itself.

8. If the application is informed that the DMS failed, the application should continue the same as if a file-system operation failed or was rejected. Depending on the scenario being followed, there may need to be special care to ensure that the user's work is not lost and there is no unbreakable cycle of ODMA re-attempts.

9. It will be necessary for an application to accurately track whether any working document held within the application

    a. Is not yet associated with a location on disk or in a DMS

    b. Is associated with a conventional file-system location

    c. Is associated with an ODMA DMS, having an ODMA Document ID and a file-system `docLocation` used for transferring the document file between the application and the DMS

    d. Has any content that has not been reflected in a stored file in either a file-system location or an ODMA DMS

ODMJNI 1.0 simplifies the simple scenarios by reducing the number of operations that a Java-based desktop application must perform in coordinating with ODMA and an ODMA DMS. The classes that implement the ODMJNI bridge handle the fine details automatically and relieve the application from any direct use of the ODMA API.

1.2 Design Principles

The ODMJNI 1.0 interfaces are designed to fit the requirements of a typical ODMA 1.0-aware desktop application.

1. The design permits straightforward following of simple scenarios.

2. The implementations of interfaces that introduce ODMA awareness for the application perform all housekeeping necessary for adjusting to the availability of the ODMA Connection Manager and the presence of any default ODMA DMS for the application.

3. It is not necessary for the application to keep track of whether or not the ODMA connection is usable.

    a. The interface can be checked prior to attempting to coordinate individual operations. If ODMA is not available, the application can simply perform its local alternative at once.

    b. The operation can be attempted and, if ODMA is not usable, that will be quickly reflected in the result, and the local alternative can be performed.

    c. The intention is to permit sequences of operations that may be the most convenient fit with the application's implementation of file-oriented operations.

4. There are no checked exceptions thrown by ODMJNI interface methods.

    a. Fabricating exceptions from native method implementations is costly, and the ODMA API is not exception-oriented. There is always some result, even for unsuccessful operations.

    b. There are simple method calls for determining the outcome of an operation and whether exceptional treatment is required.

    c. Operations that return interfaces for new objects, such as a document that is managed by an ODMA DMS, always return valid interfaces. When there is no valid document, an easily-recognized null-document interface is returned.

    d. Methods that return data (usually Java String or arithmetic values) will return default "null" values when there is no usable data value (e.g., for the `docLocation` when no ODMA DMS document is available).

1.3 Integration Principles

ODMJNI 1.0 is implemented as components. Substitution of component implementations is supported by having all references to ODMA-aware functions be through implementation-independent interfaces. The following principles apply:

1. No classes with native methods are exposed to the desktop application. This allows the application to operate entirely with interfaces that conceal platform dependencies.

2. An application can choose among available implementation classes for ODMA-aware features by having one simple interface-reference declaration (section 2.1, Establish the Connection Interface).

    a. Compatible substitutions of the named implementation class can be installed without having to recompile or update the application.

    b. Different but compatible implementation classes can be substituted by recompiling only that application class in which the initial interface reference is created.

3. Once an implementation is chosen, all further interaction is through references to the common, implementation-independent interface.

4. When a method of an ODMJNI component interface delivers another ODMJNI component implementation as its result, that result is always a valid reference to an interface.

    a. The implementation classes that provide the interface are not disclosed to the application.

    b. Error and exception conditions that apply to the delivered interface can be determined by using methods on the delivered interface.

2 Basic Connection Lifecycle

For a Java application to be ODMA-aware, it must maintain an ODMA connection for the duration of operation.

This is accomplished by creating a Connection interface and maintaining it until no document operations are required.

The connection need not be active at all times. The ODMJNI implementation delays establishment of an actual connection until the application requires availability of the connection to be known in order to advise the application whether ODMA document operations or local desktop operations should be performed in a particular situation.

It is also valuable to close the connection when it is known that further operations will not be required.

2.1 Establish the Connection Interface

2.1.1 Standard Connection Interface

Connections via ODMJNI implement the Java interface

`info.odma.simple100.Connection`

Ideally, the desktop application will establish a single connection and employ only that interface from then on. For example, the Java declaration

```
info.odma.simple100.Connection

    MyOdma = new info.odma.odmjni.Simple100Connection("MyAppId");
```

uses the ODMJNI `Simple100Connection` class to implement a `Connection` interface referenced by application variable `MyOdma`. All subsequent ODMA operations can be carried out via methods on `MyOdma` and the additional interfaces delivered by any of those methods.

The `info.odma.simple100` package is for those interfaces and related elements that supply simple ODMA-awareness to an application using only features of the ODMA 1.0 API, the common subset of all ODMA versions.

2.1.2 Substitute Connection Interfaces

Having this single point where the implementation is specified makes it easy to test applications and to upgraded to alternative implementations, so long as the same interface is supported. For example, the declaration

```
info.odma.app100.Connection

    MyOdma = new info.odma.simple100.NullConnection("MyAppId");
```

uses a standard null connection. In this case, all use of `MyOdma` methods will respond exactly as if ODMA is unavailable for use by the current application execution.

This and other substitute connection-implementation classes can be used to confirm that the application is behaving properly when ODMA operations are not to be used.

{Author Note 2006-10-15: The implementation of interface and the interface needs to be distinguished more clearly. There are also two cases:

1. **naming of a specific implementation in the declaration of a reference to the interface, as shown above**
2. **using the same implementation name but substituting components relied upon for the implementation**

The second method might be relied upon during initial development, and may be more useful in troubleshooting where the source code of the ODMA-aware application is not available.}

2.2 Use the Connection Interface

There are two kinds of methods on the Connection interface: ones for determining the status of the connection with ODMA and an ODMA DMS, others for conducting operations with ODMA DMS documents

2.2.1 Check Status of ODMA Connection

The connection method

```
boolean dmsAvailable()
```

returns `true` when there is an ODMA DMS available for operating with this execution of the desktop application. Otherwise, `false` is returned and other operations will behave the same as for a null connection implementation.

A connection interface's `dmsAvailable` `false` result will never change back to `true`. It is safe to skip use of further ODMA operations in this case. Continuing to attempt additional operations is harmless, however.

ODMJNI delays linking to the ODMA Connection Manager and determining availability of an ODMA DMS until the first Connection interface operation is performed. Performing a `dmsAvailable` check qualifies.

2.2.2 Obtain DMS Document Interfaces

Three basic connection methods engage an ODMA DMS directly. Each of them delivers a document interface of type

`info.odma.simple100.Document`

The methods are

```
Document acceptNewDocument()

Document chooseDocument()

Document openKnownDocument(String docID)
```

Choice of method depends on the nature of the scenario being implemented:

- Having a new document that is not yet associated with a local file or ODMA document
- Requesting provision of an ODMA DMS document file of the user's choosing
- Requesting provision of an ODMA DMS document file whose ODMA Document ID is already known

The resulting document interface is used for operations with the specific document, including determination of successful availability.

{Author Note: There are optional parameters that will be defined when the scenario sections are expanded.}

{Author Note: It is a peculiarity of ODMA that a known document ID can be used for any DMS registered on the computer that the application is running on. However, the operation can't be performed unless there is already a specific DMS established as the default DMS for the application, even though that is not the DMS for which a known ID is to be used. There is no attempt to work around this situation in ODMJNI 1.0.}

2.3 Close the Connection Interface

A connection interface (such as the one referenced by `MyOdma` in section 2.1.1) will be released automatically by the Java system some time after it is no longer reachable by the application.

The method

```
void close()
```

allows the connection implementation to release no-longer-needed resources as early as possible, even though the application is not terminating and there may still be document interfaces in use.

Using `close()` is an assurance from the application to the connection implementation that no further requests for usable document interfaces need be satisfied.

When a connection implementation is closed, further use of connection interface methods will be the same as for a null connection. In particular, `dmsAvailable()` will return `false`.

Closing a connection has no impact on still-available document interfaces previously obtained via that connection.

{Author Note: Although closing a connection while there are still-usable document interfaces from that connection may require retention of most DMS-related resources until the documents themselves are released, closing the connection assures that the resources will be released as soon as all current document interfaces are released.

If there have been errors in ODMA operations that suggest the DMS is not functioning properly, or the user has a way of indicating that ODMA should not be used for further document creation and access operations, closing the connection is the simplest way to provide null-connection responses for any additional ODMA-aware operations at the connection level.}

2.4 Restrictions

The instantiation of a connection implementation and all use of interfaces for ODMA connections and documents must be conducted on the same Java thread. This must be the application user-interface thread.

This restriction is required by Java. It is also required in order for the connected ODMA Document Management System (DMS) to communicate with the application user via the user interface.

{Author Note: This requirement will be defined more precisely after implementation testing. The user-interface thread case will be verified in the early testing of the ODMJNI pilot integration.}

2.5 Additional Considerations

Connection-implementing classes can throw unchecked exceptions from constructors and interface methods. There are no workarounds. The application will be terminated.

Although the exceptions are unchecked, they are all documented in the class and interface definitions.

For example, if an ODMA AppId provided to the `NullConnection` constructor is not correctly formatted, the constructor will fail with an unchecked exception. Since AppId strings are generally built into the program, there is no reason for this check to fail in a correctly running production application.

In case some standard ODMA parameters are obtained from input sources and not built into the program, there are static methods that can be used to verify the correct format of data before submitting the data to the ODMA methods.

{Author Note: Name the class of the unchecked exception and reference the section where the exceptions are defined.}

{Author Note: Identify the class used for format-checking ODMA parameter strings and reference its definition.}

3 Basic Document Lifecycle

{To be supplied. This is about the general care and feeding of the info.odma.simple100.Document interfaces obtained from Connection interfaces (section 2.2.2).}

4 Scenario: New Managed-Document Creation

This scenario involves there being an application document for which no associated file-system or DMS location has been determined – the document exists in its present state only in the custody of the application and nowhere else and we speak of it as an **anonymous document**.

An anonymous document can always be saved at the request of the user. There are also situations where the application invites the user to save the document lest work be lost. Either way, the first step is to offer the document to ODMA unless it is already known that `dmsAvailable()` is `false`.

When an anonymous document is to be saved, the ODMA-aware application must first offer the document to the ODMA Document Management System via connection-interface method `acceptNewDocument()`. If that request is declined, the application will either abandon the save operation or offer its own File | Save As … dialog to the user.

If the `acceptNewDocument()` returns a Document interface for which `Document.operationSucceeded()`is `true`, completion of the save operation should be carried out using that `Document` interface.

If the `acceptNewDocument()` request returns a `Document` interface for which `Document.operationCancelled()` is true, the application should cancel the save operation only when doing so does not lose any work. This will be when the document can be kept open and the user will still have the opportunity to choose to save the document or not. If the document can't be kept open (e.g., because of a pending shutdown), the request should be treated as if `Document.localOperationRequested()` is `true`.

If, instead, the returned Document interface has `Document.localOperationRequested() true`, or `Document.operationSucceeded() false`, the application should automatically offer its own equivalent File | Save As … dialog.

Even when `Document.operationSucceeded()` is `true`, the additional steps involved in saving the document may fail. Continuation from those failures should be as if the original `acceptNewDocument()` request was cancelled or failed, as appropriate.

{Author Note: The overall scenario continues with saving the file to the location specified by the value of `Document.docLocation` and then advising the DMS that the file is available for transfer to custody of the DMS. If that succeeds, continuation is essentially the same as if there had been a successful `openKnownDocument` at that point. This scenario needs to be laid out simpler and then deeper. That can be figured out when we see how to refactor the presentation of all of the scenarios to be economical and straight-forward.}

4.1 Conditions for Saving of Anonymous Documents

{Author Note: I am not satisfied with having all of this material under this heading. Basically, it is a matter of whether or not there is unsaved content. Then it matters whether the document is anonymous or not, and then whether it is an ODMA document or not. After more material is gathered here, it will need to be refactored to be presented better. Also, there is some level of scenario that applies to the user's world and not so much what is happening internal to the ODMA-aware application.}

- In document creation applications, this can occur when there is a fresh document in which content has been entered. For example, when Microsoft Office Word is started directly, an initial blank document (with a provisional name such as "Document1") is available for receipt of content.

- Generally, if there is a new document open that has received no content, there is not considered to be any requirement to save the document. For example, if Microsoft Office Word is closed with a fresh blank document open, the blank document is simply discarded without any action required of the user.)

- When documents are constructed based on templates, it depends on the design of the application whether or not there is considered to be unsaved content. For example, when Microsoft Office Word is used to start a new document using an identified template, Word will request confirmation that the document be preserved once any content customization has occurred:

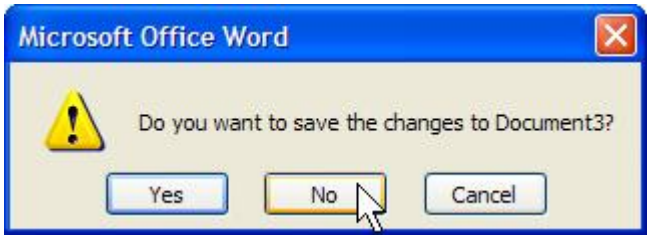

Figure 1. Word invitation to save a fresh template-based document

- Whether or not any content has been introduced, the user can request saving of the open but anonymous document to a persistent location.

- When an anonymous document is to be closed and there is unsaved content, the application is expected to give the user an opportunity to save any work before the document is closed and the work is lost. This can occur in a wide variety of situations. E.g.,

- The user selects File | Close on an application menu.
- The user selects a "Close Window" button in the application window that displays the document

- The user selects a "Close" button for the application itself and there are one or more anonymous open documents

- "Logoff" of the user session is requested, leading to an implicit close of the application.
- "Turn Off Computer" is selected and the application is still open in the current session

- When an anonymous document is saved, the behavior is always equivalent to the usual File | Save As … menu selection because there is no understood location of the document.

{To be developed further. See section 2.2.2 for the basic idea. There are similar situations when the document is not anonymous, but has been changed. Then there must also be an invitation to save the content, and the only difference is that it will be saved to the known location when that is available. If it is not available, a Save As may be forced anyhow. The additional prospect of saving it to a DMS or not saving it to a DMS makes the ODMA-aware scenario more complicated.}

5 Scenario: Choosing Managed-Document to Use

{To be supplied. See section 2.2.2 for the basic idea.}

6 Scenario: Using a Known Managed-Document

The `openKnownDocument` request is used when it is known that an ODMA-managed document is to be used. The ODMA-aware application already has an ODMA Document ID and the corresponding document is to be opened for use by the application.

An application may encounter an ODMA Document ID in any one of the following ways:

1. An ODMA Document ID may be provided as a command-line parameter to the application. The format of ODMA Document IDs is such that they cannot be confused with ordinary file-system names.

2. An ODMA Document ID may be carried in the application's recent documents list and be selected by the user as a document to be opened.

3. An ODMA Document ID may be carried in the Microsoft Windows "Recent Documents" list in a way that is associated with the current application, leading to case (1).

4. ODMA Document IDs are stored in documents of the application as links to other, related documents. Exercising one of those links in the application may result in having to access the known document.

5. ODMA Document IDs may be received by users (e.g., via e-mail or workflow-system messages) and supplied in file-open dialogs or by direct interaction with the Document Management System, leading to case (1).

For these situations, there is no first-refusal situation. The application has an explicit requirement to access an ODMA document. Fall-back behavior in the event that `openKnownDocument()` fails is entirely application-dependent.

{To be continued. See section 2.2.2 for the basic idea.}

7 Scenario: Building Managed-Documents from Templates

In many applications for managed documents, the document creation procedure involves reliance on pre-existing templates for creation of documents. These template-based documents may also fit into a workflow discipline. The overall procedure is something like this:

1. Template Creation. The template for a type of document or form is created and available as part of a work process. The template may itself be a managed document or it may be distributed by some other means. Templates may be developed in different applications than those used in document initiation and content entry. Templates may be introduced into the DMS via the DMS rather than via an ODMA-aware application.

2. Document Template Selection. When a document of a particular type is to be created, the template is named as part of document initiation (as in the Microsoft Word File | New … dialog). The template could also be retrieved as a managed document.

3. Document Initiation. For applications that (optionally) operate from templates, it is important that the application treat the document triggered by the template as if it is an anonymous document. That is, saving the document should not be considered submission of a replacement for the template. The template is not the document being created or manipulated. In this case, the document that is started from the template is treated essentially the same as a new document (section 4, Scenario: New Managed-Document Creation), and normal ODMA-aware behavior places the new document under management and under any automated workflow procedures.

4. Alternative Document Initiation. When the ODMA-aware application does not distinguish between templates and pre-existing documents, the effect of specialized document types and their templates can be achieved by procedure rather than by reliance on a template feature of the application. This is accomplished by employing read-only documents as boilerplate for customization in use of the application. It is important, in this case, that any DMS-provided template documents be delivered as read-only so that a Save As … operation is required in order to introduce the modified boilerplate as a new managed document .

5. Once a template-based document (initiated by either method) managed as an ODMA document, modified versions are returned to the DMS in the usual way, as part of Save and Close operations in the application. Whether the document is held in a working state or is checked in for progressing in some workflow procedure is determined external to the ODMA-aware application by interaction between the user and the DMS, using the dialogs presented by the DMS in response to ODMA-aware actions of the desktop application.

8 Class, Interface, and Method Documentation

{To be supplied}

9 Packaging and Deployment

{To be supplied}

10 Confirmation and Troubleshooting of Operation

{To be supplied}

11 Resources and Reference Materials

Liang, Sheng (1999).
The Java Native Interface: Programmer's Guide and Specification. Addison-Wesley (Reading, MA: 1999). ISBN 0-201-32577-2. Available at http://java.sun.com/docs/books/jni/. The files for worked examples are also available. The necessary header files and the javah command-line tool are provided as part of the Java SDK.

{Accurate list to be supplied:

Add link to the Microsoft Office Word application whose behavior is used as an example.

Add links to the relevant Java specifications on packaging, interface usage in component development, and style/naming practices.

Add links to the ODMA Specifications where the simple ODMA-aware principles can be teased out.

Add link to the ODMA FAQ that describes the basic connection principle.}

12 Appendix: Revision History

Review Draft 0.20 (2006-10-15): An additional scenario for building documents from templates (section 7) is introduced for its importance in forms-based applications, often found in workflow processes. The notion of anonymous documents is introduced to motivate the situations in which `acceptNewDocument()` is used (section 4). The importance of openKnownDocument() is identified by a list of the ways that it can show up in document-processing workflows and in returning to previous work (section 6). These sections are still very rough and will need to be refactored to have all cases covered in a straightforward way. They are presented for early review so that usability requirements for an initial pilot operation can be appraised.

Review Draft 0.10 (2006-10-02): The basic outline is created with anticipation of separate usage scenarios for anonymous unsaved documents, choosing existing documents, and choosing known documents. The basic principles of the connection interface are sketched, along with identification of three different methods for establishing managed documents.