ORACLE
INSURANCE

Oracle® Documanage

# Documanage Programmer's Guide

version 6.6

Part number: E14904-01

March 2009

**THIRD PARTY SOFTWARE NOTICES**

This product includes software developed by Apache Software Foundation (http://www.apache.org/).
THIS SOFTWARE IS PROVIDED "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.



---

This product includes software distributed via the Berkeley Software Distribution (BSD) and licensed for binary distribution under the Generic BSD license.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS ``AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.



---

This product includes software developed by Jean-loup Gailly and Mark Adler. This software is provided 'as-is', without any express or implied warranty. In no event will the authors be held liable for any damages arising from the use of this software.



---

This product includes software developed by the Massachusetts Institute of Technology (MIT).
THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.



---

This software is based in part on the work of the Independent JPEG Group (http://www.ijg.org/).

---

This product includes software developed by Sam Leffler of Silicon Graphics.
THE SOFTWARE IS PROVIDED "AS-IS" AND WITHOUT WARRANTY OF ANY KIND, EXPRESS, IMPLIED OR OTHERWISE, INCLUDING WITHOUT LIMITATION, ANY WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.
IN NO EVENT SHALL SAM LEFFLER OR SILICON GRAPHICS BE LIABLE FOR ANY SPECIAL, INCIDENTAL, INDIRECT OR CONSEQUENTIAL DAMAGES OF ANY KIND, OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER OR NOT ADVISED OF THE POSSIBILITY OF DAMAGE, AND ON ANY THEORY OF LIABILITY, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE



---

This product includes software components distributed by Steve Souza.

THIS SOFTWARE IS PROVIDED BY THE AUTHOR AND CONTRIBUTORS "AS IS'" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Copyright © 2002, Steve Souza (admin@jamonapi.com). All Rights Reserved.

---

This product includes software components distributed via the Berkeley Software Distribution (BSD).

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS'" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Copyright © 2000-2006 www.hamcrest.org. All Rights Reserved.

---

This product includes software components developed by the Independent JPEG Group and licensed for binary distribution under the Independent JPEG Group license.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS'" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Copyright © 1994-1998 AIIM International. All Rights Reserved.

---

This product includes software components developed by Sam Stephenson.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Copyright (c) 2005-2007 Sam Stephenson

---

This product includes software components developed by Sun Microsystems.

Copyright (c) 1995-2008 Sun Microsystems, Inc. All rights reserved.

---

This product includes software components distributed by Vbnet and Randy Birch.

Copyright © 1996-2008 Vbnet and Randy Birch. All Rights Reserved

---

This product includes software components distributed by MetaStuff.

THE SOFTWARE IS PROVIDED BY METASTUFF, LTD. AND CONTRIBUTORS "AS-IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDINGBUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL METASTUFF, LTD. OR ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS AND SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, ARISING IN ANY WAY OUT OF THE USE OF THE SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.



---

This product includes software components distributed by JSON.

The Software shall be used for Good, not Evil.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.



---

This product includes software components distributed by OpenSSL (http://www.openssl.org/).

THIS SOFTWARE IS PROVIDED BY THE OPENSSL PROJECT "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE OPENSSL PROJECT BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA OR PROFITS; OR BUSINESS INTERRUPTION) HOWVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE



---

This product includes software components distributed by Yahoo! Inc.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS'" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.



---

# Publication history

First issue for Version 6.4: July 2004
Revision 1 for Version 6.4 Service Release 1: December 2004
Revision 2 for Version 6.4 Service Release 2: May 2005
Revision 3 for Version 6.4 Service Release 4: September 2005
Revision 4 for Version 6.4 Service Release 5: September 2006
Revision 5 for Version 6.4 Service Release 6: July 2006
Revision 6 for Version 6.5: December 2006
Revision 7 for Version 6.5: September 2007
Revision 8 for Version 6.6: March 2009

# Documanage documentation roadmap

### Administrating

Administrator's Guide

Installation Guide

Database Administrator's Guide

### Programming

API help

**Programmer's Guide**

### Workstation

Workstation Guide

Workstation help

# Table of Contents

# Preface

## Introduction

Skywire's Documanage from Oracle document management system provides flexible, robust, scalable solutions for enterprise document management and workflow. A key aspect of this flexibility is the ability of programmers to write custom additions, interfaces and applications to the system. This document describes the facilities provided with Documanage which allow programmers to build such extensions and how to use them to build client applications. There are multiple Application Programming Interfaces (API's) described here which serve several different purposes. Some are broadly applicable interfaces to the system overall while others provide extensions to specific services such as workflow functions. Not all this information will be critical to every programmer, a goal of this document is to provide both an inventory of what is available along with guidance to the appropriate facility to use.

For customers desiring a portable multi-platform solution, the general Documanage API is supplied as a C-language callable library of functions known as the DmgAPI. This library provides programmers access to the full range of Documanage capabilities from many popular platforms and programming environments. Libraries are provided for Microsoft Win32, Linux on Intel, Solaris, AIX and IBM MVS or z/OS. The C-language interface makes the API accessible from virtually any language including C, C++, and visual languages like Microsoft Visual BASIC. This API is now the preferred way to access Documanage programmatically, from any supported client configuration.

The DmgAPI is not documented in this guide. Instead, the documentation is provided in a useful cross-platform HTML-based help system with the

software distribution. This help system is the complete and definitive guide to the DmgAPI and providing it in this form insures it can stay up to date and synchronized with the libraries as they are updated. See your product distribution media for the folder or directory containing the DmgAPI help pages.

For Microsoft Windows clients, the general Documanage API is supplied as ActiveX controls (OCX's) with underlying dynamic link libraries (DLL's). These controls are completely described in this Programmers' Guide. Many of the Documanage client applications are themselves built using these controls. Not all of the controls are required for every application, but they may be mixed and matched supplying a customizable set of features and interface elements for any application's needs. These controls are particularly useful in visual ActiveX environments, such as Microsoft Visual BASIC. With the introduction of the DmgAPI library interface, the ActiveX controls will continue to receive limited support. The useful visual interface controls will continue to receive support.

Several custom extensions to the workflow capabilities of Documanage are described in this guide. Two workflow task-types allow for extensions designed to run on the server or on the workstation, the Poller and Launcher tasks. To make use of these tasks custom programming is required as described in this guide. Additionally, an external workstation-side program is now supplied with Documanage that can automate processing of any generic workflow task, including interfacing to external systems such as e-mail or other notifications, examining information within Documanage in complex ways to make go/no-go decisions or virtually any other automation you can imagine. This program is called the Workflow Daemon application and executes libraries of custom routines provided according to the API documented here and the DmgAPI. This guide describes how the daemon application can be integrated into a workflow process and provides some example routines for simple functions. The daemon application and the sample code is provided with your product distribution media.

# Who Should Use This Manual?

This manual is for programmers building applications which will interface directly with Skywire's Documanage from Oracle document management system. It describes a set of ActiveX controls provided with the system to accomplish critical client component functions. The controls provide the primary Application Programming Interface (API) for client software to communicate with and request functions of a Documanage server.

Users of the controls should be familiar with programming using ActiveX controls in their target language platform. They should also be familiar with Documanage concepts and system architecture. Knowledge of the Documanage Administration module and Documanage Workstation Client are essential.

# What You Need to Get Started

Before you start, having a few of the basics installed, working and tested will provide a solid foundation and environment for your development work.

- A working Documanage server system, including router, server and administration components. This may be on the development machine itself or elsewhere on the LAN.
- If you will be working with workflow, the Workflow Designer application should be installed.
- The programmer's workstation should have at least the following installed and operational:
  - Documanage Workstation client software
  - If you plan to access Documanage documents using ODMA interfaces, then the ODMA client software should be installed.

- Development environment of choice and programming tools
- Documanage SDK

Documanage ActiveX controls can be used in various programming environments such as Microsoft Visual Basic, Delphi, and Microsoft Visual C++ or over the Internet using ActiveX capable browsers such as Internet Explorer.

# Development Projects

Using these controls, applications can establish sessions with the Documanage servers, perform queries, import and retrieve documents, browse through folders, display documents and manipulate workflow projects. The controls are divided into several functional categories as well as into “visual” and “non-visual” controls.

“Visual” controls, as the name implies, provide not only a functional interface to the server back-end, but a graphical interface to the end-user as well. These controls are used, for example, in the Documanage client to provide Cabinet and Folder views, as well as viewing documents and annotations within the client application.

“Non-visual” controls are supplied for use either behind the scenes in a visual application (for example in establishing a session to the server, or when the visual controls interface is inappropriate) or by non-visual applications such as import/export utilities, middleware such as a web bridge application or administrative functions where graphic display is not required.

Finally, there are three custom development opportunities that allow you to extend the workflow functionality in Documanage. The Launcher, Poller and Workflow Daemon tasks are special workflow task types which call custom code during the Documanage installation and setup. You can extend the

sample code provided for each of these types to create meaningful workflow tasks.

# Using this manual

Various constants are referred to in this manual. Each constant is in all caps and begins with “EZP_”. For Visual Basic, these constants are defined in a Visual Basic Module file called DocumanageAPI.bas. (This file is included with you materials included in your toolkit.) C++ programmers have an equivalent .h file. Users of other languages must convert one of these files into a usable format.

# Contents

This manual is organized as follows:

- **General Topics:** Chapter 1 provides technical information about general topics of interest to users of the Documanage ActiveX controls.
- **Non-Visual controls:** Chapter 2 provides technical information about various non-visual Documanage ActiveX controls.
- **Visual controls:** Chapter 3 provides technical information about various visual Documanage ActiveX controls.
- **Custom Workflow Tasks:** Chapter 5 provides technical information about implementing custom Documanage Workflow Launcher and Poller tasks.

# Conventions

The *Programmers Guide to DocuManage* manual provides consistent typographic conventions and keyboard formats to help you locate and interpret information easily. These conventions are provided below.

### Typographic and Keyboard Conventions

| Convention | Description |
|---|---|
| *Italics* | Command, dialog box, icon, and field names |
| San serif font | Directory, folder, and file names |
| 1 **Numbered lists** | Provide step-by-step procedures for performing an action |
| ◆ Bulleted lists | Provide grouped information, not procedural steps |

# Related documents

In addition to this manual, the following related publication(s) are also available from Oracle Software:

- Administratror’s Guide
- DocuManage™ Workstation Guide

# Suggestions

We welcome your comments, suggestions, and concerns about this manual or any Oracle Software publication. Send your comments to:

Skywire's Documanage from Oracle Software Technical Documentation
3353 Peachtree Rd NE, Ste 800
Atlanta, Georgia 30326

# General Topics

## Control Inventory

Here is a complete list of the Documanage controls, arranged by category as described in this document. Each control is registered and thus only one copy of each control is active on a given Windows system installation at once:

| Non Visual Controls | Description | VB Component |
|---|---|---|
| DmgDiary.ocx | Diary Control | DmgDiary ActiveX Control Module |
| PODocSpecifier.dll | Document Specifier Classes | PODocSpecifier |
| PODocument.ocx | Document Control | PowerOffice Document Control |
| POFolder.ocx | Folder Control | PowerOffice Folder Control |
| POPoller.ocx | Import Poller Control | PowerOffice DirectroyPoller Control |
| POProject.ocx | Project Control | PowerOffice Project Control |
| POSession.ocx | Session Control | PowerOffice Session Control |
| POQuery.ocx | Query Control | PowerOffice Query Control |
| POVolume.dll | Volume Control | POVolume Module |

| Visual Controls | Description | VB Component |
|---|---|---|
| DMDTPicker.ocx | DateTime Picker Control | DMDTPicker ActiveX Control Module |
| DMGQBE.ocx | Query By example Control | DMGQBE ActiveX Control Module |
| DmgViewer.ocx | Viewer Control | Documanage Viewer Control |
| POFLD_Doc.ocx | Folder/Document Control | PowerOffice Fld. and Docs. Control |
| POFLD_List.ocx | Folder List Control | PowerOffice Folder List Control |
| POTree.ocx | Tree Control | PowerOffice Tree Control |

# Handles and Hierarchy

At a high level, Documanage controls allow you to:

- Create a connection to the Documanage document management server.
- Open a Cabinet.
- Find, view and manipulate items in that Cabinet (i.e., folders and documents).

These three different actions correspond to 3 different “objects”:

- An open connection to the server.
- An opened Cabinet.

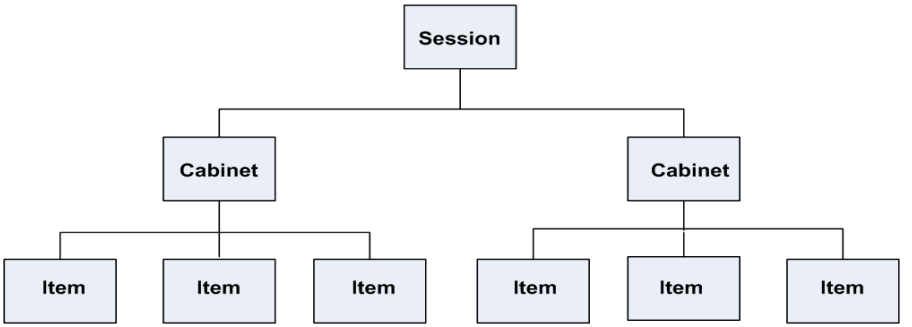
- Items in a cabinet.

Each of these “objects” are referenced in code by “Handles” – discrete 32-bit opaque identifiers (i.e., "cookies") which identify these objects. These values should never be changed and should only be used in the various properties and methods of the controls.

Since each object is different, each has its own handle:

- A connection to the Server is referenced by a Session Handle (called hSession).
- An opened Cabinet is referenced by a Cursor Handle (called hCursor).
- Items inside a Cabinet are referenced by Item Handle (called hItem).

Each of these three objects relates to the others. Think of the whole thing as one big tree.




At the top of the tree is a single Session. Under the Session are one or more Cabinets (which we also call Cursors). Under each Cabinet are many Items. Cabinet handles (hCursors) are useless without knowing to what Session (hSession) they belong. Folders and documents (hItems) are also useless without knowing in what Cabinet they exist.

Different ActiveX Controls manage each object in the system:

- The Documanage Session Control creates and breaks connections to the Server and generates hSession handles for other controls to use.
- Cabinets are created and destroyed by the Documanage Query Control. These cabinets can be accessed by other controls by an hCursor handle it provides for you. This control also allows access to the different elements inside a Cabinet, referred to by hItem handles.
- The Documanage Document Control and Folder Control allow detailed access to documents and folders inside a cabinet.

To sum up:

- **hSession Handle:** The Session control is used when a user connects/logs in to the Documanage server. A unique hSession handle is created for every connection to the server. The hSession is passed between the various controls to identify the proper session when requesting information from the server. The hSession is required for all controls specified in the documentation.
- **hCursor Handle:** The Query control is used when accessing a cabinet. A unique hCursor handle is created to identify a cabinet. A new hCursor is assigned to each instance of a cabinet when it is opened. There may be several hCursors in an hSession, since a user may have several cabinets open at the same time.
- **hItem Handle:** hItem is used to identify a specific cabinet root, folder or document in an open query. Each hItem belongs to a hCursor, just as a folder or document belongs to a specific cabinet.

# Document Specifiers

Documanage controls provide access to documents through handles, which are only valid for a particular session and open cabinet. A Document Specifier, or DocSpecifier for short, provides a way to keep a permanent reference to a document between sessions. A DocSpecifier can refer to either the latest version (whatever it is, as it continues to change over time) or a specific version of a document. Refer to the “Document Specifier Controls” section.

# Using Array Properties

Many properties may be of type string array, long array, etc. Properties of these types have to be called in this format:

MyVal = Control.Items(MyIndex)

The size of these arrays are often stored in another property of the same name with the word “total” appended to the beginning. These arrays are zero based so that the valid range is from 0 to Total – 1. Newer additions to the controls will not use array properties but a Method to access the array. These methods are often in the format:

“Get*SomeArrayOfValues*(index)” where index is from 0 to the property: “Total*SomeArrayOfValues.*”

For example, in the Document control the Document Types are exposed as an array property but the newer version history information is exposed via a method. Some controls will provide both array properties and access methods. Some environments that can use ActiveX controls have difficulties with these array properties, such as early versions of Visual Basic Scripting Edition. Standard Visual Basic 4.0 and 5.0 and Visual C++ v4.2 and later have no problems with these properties.

# Dynamic Loading of Controls Using CreateObject or Type Libraries

Advanced users may be familiar with OLE Automation Servers (now called ActiveX DLLs). The Documanage Session, Query, Folder, and Document controls are both ActiveX controls and ActiveX DLLs at the same time. By using their OLE names (listed below) one can dynamically create these objects using CreateObject. Be warned that this uses “late binding”; the controls will be VB Objects which provide no feedback or warning for syntax. An even better approach is to import the ActiveX Type Libraries into your project. Using type libraries allows “early binding”; VB will know everything about the interface to the controls, will provide feedback and warnings for syntax and be much faster. You cannot, however, include one specific Documanage Control into your project (the standard way) and then try to use a type library to dynamically create that same type of control. If this case arises then you are forced to use the CreateObject call and standard (non-typed) Visual Basic Objects. This is a limitation of VB and OLE.

Visual C++ (using MFC) allows similar type library access to the controls. You must direct the class wizard to create a new class from a type library. Create an instance of this class and call CreateDispatch(OLE-Name).

The type libraries are named the same as the Active control’s files except they end in “.tlb” instead of “.ocx”:

| Control OLE Name | Type Library | OLE Name |
|---|---|---|
| PODocument.ocx | PODocument.tlb | PODOCUMENT.PODocumentCtrl.1 |
| POFolder.ocx | POFolder.tlb | POFOLDER.POFolderCtrl.1 |
| POSession.ocx | POSession.tlb | POSESSION.POSessionCtrl.1 |
| POQuery.ocx | POQuery.tlb | POQUERY.POQueryCtrl.1 |

# Dates

All properties in the controls should be set in the following format:

“yyyy-mm-dd hh:mm:ss”

Visual Basic provides the Format function to easily create dates in any form.

Documanage controls show the dates and times in the specific format used in the user’s locale, as specified in the Regional Options of the Windows Control Panel. The programmatic behavior of the controls remains the same, so clients should use the format shown here in the programmatic interface.

# Error Handling

Errors are handled in one of two ways. If an error occurs in response to a property or method an error is “Raised” in Visual Basic. “On Error” code must be present to handle these errors. The Err.Description contains a readable message describing the error. The Err.Number is a number that relates to a specific Documanage error. The Err.Description contains both the description of the error and the Documanage error number. This string must be parsed to determine the Documanage error number. Optionally, error description parsing can be avoided by checking which Documanage control raised an error and then checking its LastErrorNumber property. Care should be taken in this case if you are using more then one Documanage control since it maybe difficult to determine which control raised the error. VB’s Err object attempts to report what object raised the error via the Err.Source property but sometimes this is set inconsistently by VB. When multiple controls are referenced in the scope of one error handler, parsing is the safest solution.

Here is a VB Routine to parse Err.Description:

```
Public Sub ParseErrorDesc(strError As String,
ByRef nNumber As Long, ByRef strDescription As
String)
```

```
Dim strPrefix As String
strPrefix = "DocuManage Error #"
If Left(strError, Len(strPrefix)) = strPrefix Then
nNumber = Val(Mid(strError, Len(strPrefix) + 1, 4))
strDescription = Mid(strError, Len(strPrefix) + 7, (Len(strError)) - (Len(str-
Prefix) + 7))
Else
nNumber = 0
strDescription = ""
End If
End Sub
```

The other way errors are handled are when errors occur *not* in response to accessing a property or calling a method. ActiveX controls are forced in these cases to generate an Error event. This is the error event:

- ***Error***
  Error(Number As Integer, Description As String, Scode As Long, Source As String, HelpFile As String, HelpContext As Long, CancelDisplay As Boolean)

  **Remarks:** Fired in response to an error when error raising is not possible.

  Only three parameters are of interest.

  **Parameters:**

  - *Number:* The Documanage error number.
  - *Description:* The Documanage error description.
  - *CancelDisplay:* Passed ByRef. You should always set this to FALSE to prevent the ActiveX libraries from generating a message box.

Every control contains a LastErrorSource and LastErrorNumber property which can assist in handling errors:

- ***LastErrorNumber* Data Type:** long (Read Only)
  The last DocuManage error.

- ***LastErrorSource* Data Type:** string (Read Only)
  The location of the last DocuManage error. This is useful for reporting bugs to Oracle.

# Valid IsActionAllowed Checks

**Document**
EZP_ACTION_MAKE_REFERENCE
EZP_ACTION_COPY
EZP_ACTION_DELETE
EZP_ACTION_CHECKOUT
EZP_ACTION_VIEW
EZP_ACTION_ANNOTATE_LOCAL
EZP_ACTION_BLACKOUT_LOCAL
EZP_ACTION_ANNOTATE_GLOBAL
EZP_ACTION_BLACKOUT_GLOBAL
EZP_ACTION_END_LOCAL_ANOTATE
EZP_ACTION_END_GLOBAL_ANNOTATE
EZP_ACTION_MOVE
EZP_ACTION_CHECKIN_MAJOR
EZP_ACTION_CHECKIN_MINOR
EZP_ACTION_CHECKIN_SAME
EZP_ACTION_EDIT_CONTENTS
EZP_ACTION_EDIT_ATTRIBUTES
EZP_ACTION_VIEW_HISTORY
EZP_ACTION_PUBLISH
EZP_ACTION_VIEW_WORKING

EZP_ACTION_GET_UNIQUE_KEY
EZP_ACTION_DOWNLOAD
EZP_ACTION_CHECKIN_UNDO

**Document Version**
EZP_ACTION_VIEW
EZP_ACTION_DOWNLOAD

**Folder**
EZP_ACTION_INSERT
EZP_ACTION_EDIT_ATTRIBUTES
EZP_ACTION_CHECKOUT_IN
EZP_ACTION_MAKE_REFERENCE_IN
EZP_ACTION_INSERT_IN
EZP_ACTION_GET_UNIQUE_KEY
EZP_ACTION_DELETE
EZP_ACTION_GETLEVEL

**Project Folder**
EZP_ACTION_CHECKOUT
EZP_ACTION_CHECKIN_SAME
EZP_ACTION_CHECKIN_PROJ
EZP_ACTION_CHECKIN_PROJ_MOVE
EZP_ACTION_CHECKIN_PROJ_ASSIGN

# Filter/OrderBy Chart

| Documanage Name | Internal Name | SQL Type | Length |
|---|---|---|---|
| FileType | Type | varchar | 32 |
| DocumentName | Tag | varchar | 64 |
| Major Version Number | nVersion | smallint | 2 |
| Volume | Volume | varchar | 10 |
| Author | AddedBy | varchar | 32 |
| AddedOn | AddedOn | datetime | 8 |
| AddedFrom | AddedFrom | varchar | 8 |
| Current Buis.Table | SourceTable | varchar | 32 |
| CheckedOutBy | CheckedOutBy | varchar | 32 |
| Category | ObjectClass | varchar | 32 |
| Description | Description | varchar | 240 |
| Date | DocumentDate | datetime | 8 |
| LastEditBy | LastEditBy | varchar | 32 |
| LastEditOn | LastEditOn | datetime | 8 |
| CheckedOutFor | CheckedOutFor | varchar | 24 |
| DueDate | DueDate | datetime | 8 |
| SubCategory | ObjectSubClass | varchar | 15 |

| Documanage Name | Internal Name | SQL Type | Length |
|---|---|---|---|
| Status | Status | varchar | 10 |
| Keyword1 | Keyword1 | varchar | 15 |
| Keyword2 | Keyword2 | varchar | 15 |
| Flag1 | UserFlag1 | varchar | 15 |
| Flag2 | UserFlag2 | varchar | 15 |
| MinorVerisonNumber | nSubVersion | smallint | 2 |

# Non-Visual Controls

Non-Visual controls have no user interface. At design time they appear as a 32x32 square icon, while at run-time they are invisible. Full Documanage functionality is available using only these controls. Visual controls are described in Chapter 2 of this manual.

## DmgDiary Control

In Documanage, a diary item (diary entry) is a group of scheduling properties that define an entity representing an action-item or assignment. Attributes such as due date, next alert and priority can be set for each defined action item. For example, an action item such as "turn in travel expense report" can be defined with a due date, a reminder and assigned a level of priority using Diary Control properties.
The Diary control allows for creation and deletion of diary items for each user, and modification of diary item properties.
Any user can add diary entries. Only the owner can delete or modify a diary entry.

### Properties

- ***Title*** **DataType**: String (Read/Write)
  Title(long index)

  A short summary or subject of the diary item.

**Parameters:**

- *index*: The position of the diary item in diary items list for the user. From 0 to DiaryItemsCount - 1.

- ***Description*** **Data Type:** String (Read/Write)
  Description(long index)

  A complete description of the diary item.

  **Parameters:**

  - *index*: The position of the diary item in diary items list for the user. From 0 to DiaryItemsCount - 1.

- ***Priority*** **DataType**: Short (Read/Write)
  Priority(long index)

  The priority of the diary item.

  **Parameters:**

  - *index*: The position of the diary item in diary items list for the user. From 0 to DiaryItemsCount - 1.

- **CreatedOn** **DataType**: String (Read Only)
  CreatedOn(long index)

  The date and time when the diary item was created.

  **Parameters:**

  - *index*: The position of the diary item in diary items list for the user. From 0 to DiaryItemsCount - 1.

- ***StartOn*** **DataType**: String (Read/Write)
  StartOn(long index)

  The date and time when the diary item is scheduled to start.

  **Parameters:**

  - *index*: The position of the diary item in diary items list for the user. From 0 to DiaryItemsCount - 1.

- ***DueOn*** **DataType**: String (Read/Write)
  DueOn(long index)

  The date and time when the diary item is due to be completed.

  **Parameters:**

  - *index*: The position of the diary item in diary items list for the user. From 0 to DiaryItemsCount - 1.

- ***CompletedOn*** **DataType**: String (Read/Write)
  CompletedOn(long index)

  The date and time when the diary item was completed.

  **Parameters:**

  - *index*: The position of the diary item in diary items list for the user. From 0 to DiaryItemsCount - 1.

- ***LastModifiedOn*** **DataType**: String (Read Only)
  LastModifiedOn(long index)

  The date and time when the diary item was last modified.

**Parameters:**

- *index*: The position of the diary item in diary items list for the user. From 0 to DiaryItemsCount - 1.

- ***NextAlertTime*** **DataType**: String (Read/Write)
  NextAlertTime(long index)

  The date and time of next reminder for the diary item.

  **Parameters:**

  - *index*: The position of the diary item in diary items list for the user. From 0 to DiaryItemsCount - 1.

- ***AssignedBy*** **DataType**: String (Read Only)
  AssignedBy(long index)

  The user who created/modified the diary item.

  **Parameters:**

  - *index*: The position of the diary item in diary items list for the user. From 0 to DiaryItemsCount - 1.

- ***Owner*** **DataType**: String (Read/Write)
  Owner(long index)

  The user who owns the diary item.

  **Parameters:**

  - *index*: The position of the diary item in diary items list for the user. From 0 to DiaryItemsCount - 1.

- ***Reference*** **DataType**: String (Read/Write)
  Reference(long index)

  A reference to a file, document, project or URL.

  **Parameters:**

  - *index*: The position of the diary item in diary items list for the user. From 0 to DiaryItemsCount - 1.

- ***Status*** **DataType**: Short (Read/Write)
  Status(long index)

  The status of the diary item.

  **Parameters:**

  - *index*: The position of the diary item in diary items list for the user. From 0 to DiaryItemsCount - 1.

- ***DiaryID*** **DataType**: String (Read Only)
  DiaryID(long index)

  The ID which uniquely identifies the diary item.

  **Parameters:**

  - *index*: The position of the diary item in diary items list for the user. From 0 to DiaryItemsCount - 1.

- ***KeyString*** **DataType**: String (Read Only)
  KeyString(long index)

  The fully qualified name of the OT_Diaries table primary key associated with the diary item.

**Parameters:**

- *index*: The position of the diary item in diary items list for the user. From 0 to DiaryItemsCount - 1.

- ***DiaryItemsCount*** **DataType**: Long (Read Only)
  The total number of diary items for the user. Used in conjunction with Initialize().

- ***DiaryFilter*** **DataType**: String (Read/Write)
  A specially formatted string (SQL OrderBy clause) to be used to filter which diary items of the user will be shown. This property should be set before calling Initialize().

- ***MinDiaryItems*** **DataType**: Short (Read/Write)
  The point (zero-based index) in the diary items list from where diary items for the user are retrieved. This property should be set before calling Initialize().

- ***MaxDiaryItems*** **DataType**: Short (Read/Write)
  The maximum number of diary items to be retrieved for the user. This property should be set before calling Initialize().

- ***hSession*** **DataType**: Long (Read/Write)
  The handle of the current session with Documanage Server. hSession is a value passed from Session control, which identifies the login to Documanage Server. This property should be set before using any property or method of the control.

- ***DiaryOrderBy*** **DataType**: String (Read/Write)
  A specially formatted string (SQL where clause) that changes the order of diary items shown. This property should be set before calling Initialize().

- ***LastErrorNumber*** **DataType**: Long (Read Only)
  The number of the last Documanage error.

- ***LastErrorSource*** **DataType**: String (Read Only)
  The location of the last Documanage error.

## Methods

- ***Initialize***

  This method initializes the control to get diary items list for the user.

- **CreateNewDiaryItem**
  CreateNewDiaryItem() As Long

  This method creates a new diary item for the user.

  **Return Value:** Index of the new diary item in the diary items list for the user.

- **SaveDiaryItem**
  SaveDiaryItem(index As Long) As String

  This method updates existing diary item or saves new diary item with the properties set by the user.

  **Parameters:**

  - *index*: The position of the diary item in diary items list for the user. From 0 to DiaryItemsCount - 1.

  **Return Value:** DiaryID of the saved diary item.

- **DeleteDiaryItem**
  DeleteDiaryItem(index As Long)

  This method deletes the diary item.

  **Parameters:**

  - *index*: The position of the diary item in diary items list for the user. From 0 to DiaryItemsCount - 1.

- **GetDiaryItemIndexFromID**
  GetDiaryItemIndexFromID(diaryID As String) As Long

  This method gets the diary item's index in the diary items list for the user using its DiaryID.

  **Parameters:**

  - *diaryID*: ID of the diary item.

  **Return Value:** Index of the diary item in the diary items list for the user.

# PODocument Control

The Document control allows you to display or edit the internal Documanage properties of a new or an existing Documanage document. Giving this control a valid hCursor and hItem handle (which specifies the cabinet and document, respectively), provides you with access to all the Documanage document properties. The Document control also implements the Check-In, Check-Out, and Version Control methods. The Document control has no visual interface and is invisible during run-time.

**NOTE:** The properties marked Read/Write are editable only when the document is checked out or being imported.

## Properties

- ***AddedByLabel*** **Data Type:** String (Read Only)
  Reserved for future use.

- ***AddedFrom*** **Data Type:** String (Read/Write)
  The method in which the document was brought into Documanage.

- ***AddedFromLabel*** **Data Type:** String (Read/Write)
  Reserved for future use.

- ***AddedOn*** **Data Type:** String (Read Only)
  The date on which the document was added to Documanage.
  This property is set by the server.

- ***AddedOnLabel*** **Data Type:** String (Read Only)
  Reserved for future use.

- ***AnoAnnotationsTag*** **Data Type:** Long (Read Only)
  A number that identifies the current version of the document's annotation file content.

- ***ApplicationTabLabel*** **Data Type:** String (Read Only)
  Reserved for future use.

- ***Approved*** **Data Type:** Integer (Read Only)
  The document's "Approved" content management flag value.

- ***Author*** **Data Type:** String (Read/Write)
  The user name of the person who added the document to Documanage.

- ***CheckedOutBy*** **Data Type:** String (Read/Write)
  The user name of the person who checked out the document.

- ***CheckedOutByLong*** **Data Type:** String (Read Only)
  A string with additional information about the **CheckOutByUser**.

- ***CheckedOutFor*** **Data Type:** String (Read Only)
  The reason a document is being checked out.

- ***CMResetAllowed*** **Data Type:** Integer (Read Only)
  Current user may reset the content management flags for this document.

- ***CMSetAllowed*** **Data Type:** Integer (Read Only)
  Current user may reset the content management flags for this document.

- ***ContentsTag*** **Data Type:** Long (Read Only)
  Number that identifies the current version of the document's file content.

- ***DataTypes*** **Data Type:** Long Array (Read Only)
  Array of field data types for Extended Document Properties for the current document category. Its array size is based the ItemCount value.

- ***Date*** **Data Type:** String (Read/Write)
  A user defined Date. A Documanage document level property.

- ***DateLabel*** **Data Type:** String (Read Only)
  Reserved for future use.

- ***Description*** **Data Type:** String (Read/Write)
  A user-definable long description on the document.

- ***DocID*** **Data Type:** Long (Read Only)
  A unique value for a document that remains constant between cursors and sessions. This value can be found in the OT_Docs table of the database containing the table which contains the document.

- ***DocRenditionKey*** **Data Type:** String (Read/Write)
  The rendition key for the current document.

- ***DocumentName*** **Data Type:** String (Read/Write)
  The Documanage name of the document. This is obtained from the filename, excluding the extension.

- ***DocumentSize*** **Data Type:** Long (Read Only)
  Displays the size of a document on the screen that allows you to select which properties to display in the document list.

- ***DocumentType*** **Data Type:** String (Read/Write)
  The Documanage document type of the document.
  NOTE: this is now referred to as the Document Category

- ***DocumentType Selections(Long index)*** **Data Type:** String array (Read Only)
  Array of Document Category selections. Its array size is based on TotalDocumentTypeSelections. Retrieves possible values for Document type. Its array size is based on TotalDocTypeSelections (see “Using

Array Properties").

- ***DueDate*** **Data Type:** String (Read Only)
  The document's Documanage due date, which is set at checkout.

- ***EditingAllowed*** **Data Type:** Boolean (Read Only)
  Checks security to verify a user's rights to edit the properties of a document.

- ***FileType*** **Data Type:** String (Read/Write)
  File extension that identifies the application originating a document.

- ***Flag1*** **Data Type:** String (Read/Write)
  A Documanage document level property.

- ***Flag1Label*** **Data Type:** String (Read Only)
  The label of the Flag1 user-defined drop-down list box.

- ***Flag1Selections*** **Data Type:** String array (ReadOnly)
  Returns possible values for Flag1. Array of UserFlag1 selections for the current Document Category. Its array size is based on TotalFlag1Selections (see "Using Array Properties").

- ***Flag2*** **Data Type:** String (Read/Write)
  A Documanage document level property.

- ***Flag2Label*** **Data Type:** String (Read Only)
  The label of the Flag2 user-defined drop-down list box.

- ***Flag2Selections*** **Data Type:** String array (Read/Write)
  Returns possible values for Flag2. Array of UserFlag2 selections for the current Document Category. Its array size is based on TotalFlag2Selections (see "Using Array Properties").

- ***hCursor*** **Data Type:** Long (Read Only)
  The hCursor corresponding to the hItem of the document being viewed.

- ***hItem*** **Data Type:** Long (Read Only)
  The hItem currently opened for this control. Use Refresh() after changing this property, making certain hCursor is correctly set to the hItem's corresponding hCursor.

- ***hSession*** **Data Type:** Long (Read Only)
  The current session with the Documanage server. hSession is a value passed to you from the Session control, which identifies your login to the Documanage server.

- ***IsADocument*** **Data Type:** Long (Read Only)
  Returns 1 if the current item is a document or 0 otherwise.

- ***IsAVersion*** **Data Type:** Long (Read Only)
  Returns 1 if the current item is a historical document version or 0 otherwise.

- ***IsCheckedOut*** **Data Type:** Boolean (Read Only)
  Indicates if a document is checked out.

- ***IsLocked*** **Data Type:** Boolean (Read Only)
  Indicates if a document is locked. Lock documents cannot be edited or checked-out. This typically indicated that the document is already Checked-Out, but is the original copy of the document, not the Checked-Out copy currently being edited.

- ***IsOwnedByThisUser*** **Data Type:** Boolean (Read Only)
  This is true if IsCheckedOut=TRUE and the current user is the one who checked out the document.

- ***IsReference*** **Data Type:** Boolean (Read Only)
  Indicates a shortcut document.

- ***IsTempItem*** **Data Type:** Boolean (Read Only)
  Indicates a document is a temporary item created from InitializeForNewDocument().

- ***ItemCount*** **Data Type:** Long (Read/Write)
  The number of Extended Document Properties for the current Document Category.

- ***KeyString*** **Data Type:** String (Read/Write)
  An SQL clause that uniquely identifies the current document.

- ***Keyword1*** **Data Type:** String (Read/Write)
  A Documanage document level property.

- ***Keyword1Label*** **Data Type:** String (Read Only)
  The user-defined title of this document level property.

- ***Keyword2*** **Data Type:** String (Read/Write)
  A Documanage document level property.

- ***Keyword2Label*** **Data Type:** String (Read Only)
  The user-defined title of the document level property.

- ***LastEditedBy*** **Data Type:** String (Read/Write)
  A Documanage document level property; this is the user name of the last person who checked-out the document.

- ***LastEditedOn*** **Data Type:** String (Read/Write)
  A Documanage document level property; this is the date of the last time the document was checked-out.

- ***LastErrorNumber*** **Data Type:** Long (Read Only)
  The last Documanage error.

- ***LastErrorSource*** **Data Type:** String (Read Only)
  The location of the last Documanage error. This is useful for reporting bugs to Oracle.

- ***Lengths (Long index)*** **Data Type:** Long (Read/Write)
  Array of field lengths for Extended Document Properties for the current Document Category. Its array size is based the ItemCount value.

- ***MajorVersion*** **Data Type:** Long (Read/Write)
  The document's major version value.

- ***MinorVersion*** **Data Type:** Long (Read/Write)
  The document's minor version value.

- ***Obsolete*** **Data Type:** Long (Read Only)
  The document's "Obsolete" content management flag's value.

- ***PublishType*** **Data Type:** Long (Read/Write)
  Called by GetFileType. 1=Make PDF, 0=leave document type unchanged

- ***QualifiedIndex (Long index)*** **Data Type:** String (Read/Write)
  Array of qualified column names for Extended Document Properties for the current Document Category. Its array size is based the ItemCount value.

- ***Released*** **Data Type:** Long (Read Only)
  The document's "Released" content management flag's value.

- ***RemoteFileName*** **Data Type:** String (Read/Write)
  Reserved for future use.

- ***RenditionDocID(Long index)*** **Data Type:** Long (Read Only)
  The document ID of the rendition located at the specified index.

- ***RenditionID*** **Data Type:** Long (Read/Write)
  The rendition ID of the current document or version.

- ***RenditionKey (Long index)*** **Data Type:** String (Read/Write)
  The key of the rendition located at the specified index.

- ***RenditionMajorVersion (Long index)*** **Data Type:** Long (Read Only)
  The major version of the rendition located at the specified index.

- ***RenditionMinorVersion (Long index)*** **Data Type:** Long (Read Only)
  The minor version of the rendition located at the specified index.

- ***RenditionTag (Long index)*** **Data Type:** String (Read Only)
  The document name of the rendition located at the specified index.

- ***RenditionType (Long index)*** **Data Type:** String (Read Only)
  The document type of the rendition located at the specified index.

- ***RequiredFlags (Long index)*** **Data Type:** Boolean (Read/Write)
  Array of "field is required" flags for Extended Document Properties for the current Document Category. Its array size is based the ItemCount value

- ***SourceDocumentCabinet*** **Data Type:** String (Read Only)
  Used when dealing with Checked-Out and Shortcut documents. Indicates the cabinet of the source document. Useful in conjunction with SourceDocumentFolderFilter.

- ***SourceDocumentFolderData*** **Data Type:** String (Read Only)
  Used when dealing with Checked-Out and Shortcut documents. Indicates in a human readable form the description of the source document's

current folder.

- ***SourceDocumentFolderFilter*** **Data Type:** String (Read Only)
  Used when dealing with Checked-Out and Shortcut documents. Indicates a filter suitable for the Query control's FolderFilter property. By opening a the cabinet in SourceDocumentCabinet with this property's values as FolderFilter you can get to the source document of another document.

- ***SourceDocumentName*** **Data Type:** String (Read Only)
  Used when dealing with Checked-Out documents. Indicates the name of the source document.

- ***SourceDocumentTable*** **Data Type:** String (Read Only)
  Used when dealing with Checked-Out and Shortcut documents. Indicates internal table name of the source document.

- ***Status*** **Data Type:** String (Read/Write)
  The current document Status selection. See *StatusSelections*.

- ***StatusLabel*** **Data Type:** String (Read/Write)
  Reserved for future use.

- ***StatusSelections (Long index)*** **Data Type:** String(Read Only)
  Array of Status selections for the current Document Category. Its array size is based on TotalStatusSelections (see "Using Array Properties"). A Documanage document level property.

- ***SubType*** **Data Type:** String (Read/Write)
  A Documanage document level property.

- ***SubTypeLabel*** **Data Type:** String (Read/Write)
  Reserved for future use.

- ***Sub Type Selections (Long index)*** **Data Type:** String (Read/Write)
  Array of SubType selections for the current Document Category. Its array size is based on TotalSubTypeSelections.

- ***Titles (Long index)*** **Data Type:** String (Read/Write)
  Array of field titles for Extended Document Properties for the current Document Category. Its array size is based the ItemCount value.

- ***TotalDocumentTypeSelections*** **Data Type:** Long (Read Only)
  The total number of elements in DocTypeSelections.

- ***TotalDocumentVersions*** **Data Type:** Long (Read Only)
  The total number of previous versions of a document.

- ***TotalFlag1Selections*** **Data Type:** Long (Read Only)
  The total number of elements in Flag1Selections.

- ***TotalFlag2Selections*** **Data Type:** Long (Read Only)
  The total number of elements in Flag2Selections.

- ***Total Renditions*** **Data Type:** Long (Read Only)
  The total number of renditions for the current document. Returned by a call to GetRenditions.

- ***TotalStatusSelections*** **Data Type:** Long (Read Only)
  The total number of elements in *StatusSelections.*

- ***TotalSubTypeSelections*** **Data Type:** Long (Read Only)
  The total number of elements in Sub Type Selections.

- ***TranslateNullProperties*** **Data Type:** Boolean (Write Only)
  The TranslateNullProperties property determines how the Document control outputs NULL Extended Document properties for the current Document category from the Documanage database.

TranslateNullProperties defaults to True. To change this setting, the property must be set before calling any properties or methods to get Extended Document properties. Once set, TranslateNullProperties affects all subsequent calls to the control.

When TranslateNullProperties is True, the Document control translates NULL Extended Document properties from the database to empty strings if they are character data or to zero if they are numeric data when it returns them to users as variants.

When TranslateNullProperties is set to False, the Document control returns NULL Extended Document properties from the database as NULL variants without translating them to zeros or to empty strings.

The control returns Document Control properties for the current Document Category using the Values( ) property and GetValueByTitle( ) method.

**NOTE:** You can set NULL values for Extended Document properties by passing in NULL variants using the Values ( ) property or the SetValueByTitle ( ) method. Refer to the descriptions of the Values( ) property on page 32 and the SetValueByTitle ( ) method on page 41.

- ***UseFlag1*** **Data Type:** Boolean (Read Only)
  Boolean value to determine if the Flag1 user-defined field is being used.

- ***UseFlag2*** **Data Type:** Boolean (Read Only)
  Boolean value to determine if the Flag2 user-defined field is being used.

- ***UseKeyword1*** **Data Type:** Boolean (Read Only)
  Boolean value to determine if the Keyword1 user-defined field is being used.

- ***UseKeyword2*** **Data Type:** Boolean (Read Only)
  Boolean value to determine if the Keyword2 user-defined field is being used.

- ***Values (Long index)*** **Data Type:** Variant (Read/Write)
  Array of field values for Extended Document Properties for the current Document Category. Its array size is based the ItemCount value.

- ***Version*** **Data Type:** String (Read/Write)
  A Documanage document level property.

- ***Volume*** **Data Type:** String (Read Only)
  A Documanage document level property. Also, the Documanage volume of where the document is stored.

## Methods

- ***AboutBox***
  AboutBox();

  This method displays the PODocument control's about box**.**

- ***AddRendition***
  Function AddRendition(lDoc As Long, lCursor As Long) As Long

  Creates a rendition relationship between the current document and the document specified by the lDoc and lCursor parameters.

- ***CheckIn***
  CheckIn(CheckInType As Integer, CopyStatus As Integer, VersionNotes As String) As Long

  This method checks back in a document in Documanage.

  **Parameters:**

  - *CheckInType:* A flag indicating how to check-in the document. See the core Documanage documentation on checking-in, checking-out, and versioning.

    EZP_ACTION_CHECK_IN_UNDO
    EZP_ACTION_CHECK_IN_SAME
    EZP_ACTION_CHECK_IN_MINOR
    EZP_ACTION_CHECK_IN_MAJOR

  - *CopyStatus:* Reserved for future use. Always pass zero.

  - *VersionNotes:* Specify a reason or comments explaining the check-in. Ignored when checking in as same version. This information is stored in the history of the document.

  **Return Value:** Returns a long value that is reserved for future use and should be ignored.

- ***CheckOut***
  CheckOut(hCursorTo As Long, hItemTo As Long, CheckedOutBy As String, CheckedOutFor As String, CheckedOutByLong As String, DueDate As String) as Long

  This method checks out a document in Documanage.

**Parameters:**

- *hCursorTo:* The destination hCursor corresponding to the hItemTo folder for the checked-out copy document to be placed.
- *hItemTo:* The destination folder Item for the checked-out copy document to be placed.
- *CheckedOutBy:* Reserved for backward compatibility. Pass an empty string.
- *CheckedOutFor:* Reason for the checkout. Retrievable when looking at the checked-out document.
- *CheckedOutByLong:* [**Reserved for future use.**] Pass an empty string.
- *DueDate:* The date the document is due to be checked-back-in.

**Return Value:** The method returns hItem to the CheckedOut Document in the hCursorTo Query. A NULL value is returned if the CheckOut cannot be performed.

- ***Delete()***
  Deletes the currently selected document.
- ***GetFile***
  GetFile(Filename As String) As String

  This method downloads the physical file locally (mainly for editing).

  **Parameters:**

- *Filename:* Indicates full filename with path of where the file should be placed. If you send an empty string, Documanage will create the file in the Windows Temp directory with an internally generated name.

**Return Value:** This method returns the full pathname of the file. If you indicated a filename yourself, you can ignore the return value. If you passed an empty string, the method returns the internally generated filename.

- ***GetRenditions***
  Function GetRenditions(lContentManagementFilter As Long, latestDocVersionsOnly As Boolean) As Long

  Gets the list of documents that are in rendition relationships with the current document or version. For parameter explanations, see "RenditionAPI.doc," page 4, "Get The List Of Peer Renditions For A Document", "resultFilter" and "latestOnly."

- ***GetValueByTitle***
  Function GetValueByTitle(strTitle As String)

  Retrieves the value of an Extended Document Properties field. The title for an XDP field can be determined from the Titles array.

  **Parameter:**

  - *strTitle :* the Extended Document Properties field title.

- ***GetVersionAuthor***
  GetVersionAuthor(VersionIndex As Long) As String

  This method returns the author for a given previous version of this document. This accesses the internal array of version history of the current document.

**Parameters:**

- *VersionIndex:* Range from 0 to (TotalDocumentVersions –1).

**Return Value:** The author.

- ***GetVersionComments***
  GetVersionComments(VersionIndex As Long) As String

  This method returns the comments for a given previous version of a document.

  **Parameters:**

  - VersionIndex range from 0 to (TotalDocumentVersions –1).

- ***GetVersionDate***
  GetVersionDate(VersionIndex As Long) As String

  This method returns the version date for a given previous version of a document.

  **Parameters:**

  - VersionIndex range from 0 to (TotalDocumentVersions –1).

- ***GetVersionHItem***
  GetVersionHItem(VersionIndex As Long) As Long

  This method returns the hItem for a given previous version of a document.

  **Parameters:**

  - VersionIndex range from 0 to (TotalDocumentVersions –1).

  **Return Value:** hItem for a given previous version. Currently the document pointed to by this hItem only allows Viewing and Downloading. Properties cannot be viewed or changed.

- ***GetVersionNumber***
  GetVersionNumber(VersionIndex As Long) As String

  This method returns the version number for a given previous version of a document.

  **Parameters:**

  - VersionIndex range from 0 to (TotalDocumentVersions –1).

  **Return Value:** The Version number in this format: "##.##", e.i. major.minor.

- ***Initialize()***

  This method refreshes the control, verifying that the hCursor, hSession and hItem in the document's properties are set correctly.

- ***InitializeforNewDocument***
  InitializeforNewDocument(ParentFolderHItem As Long) As Long

This method creates and returns a new hItem for the document to be imported into Documanage. The ParentFolderHitem is the hItem of the folder where you want the new document to be stored. You must use the SaveNewDocument method to actually save the newly created hItem with the physical document, as the InitializeForNewDocument only creates a new hItem. An example of using this method together with SaveNewDocument can be found below.

When setting properties of a new document PODocument.Documenttype should always be set first.

**Parameters:**

- *ParentFolderHItem:* The ParentFolderHitem is the hItem of the folder where you want the new document to be stored.

**Return Value:** Returns the hItem of this temporary item. This hItem it not particularly useful and should be ignored.

- ***InitializeXDAIndexes***
  Function InitializeXDAIndexes(strDocType As String) As Long

  Initializes the Extended Document Properties arrays for the designated Document Category. Returns zero if the call was successful. After this call the ItemCount, Titles, DataTypes, Lengths, QualifiedIndex, RequiredFlags, and Values properties will be populated.

  **Parameters:**

  - *strDocType:* the Document Category that defines the Extended Document Properties.

- ***RefreshDocument***
  RefreshDocument()

  Forces the server to update the cached document object with data from the datasource.

- ***RemoveAsRendition***
  Function RemoveAsRendition(lDoc As Long, lCursor As Long) As Long

  Removes the document specified by the lDoc and lCursor parameters from its rendition relationship.

- ***Save() as Boolean***
  This method saves any changes made to the document's properties.

- ***SaveNewDocument***
  SaveNewDocument(Pathname as String) As Long

  This method saves the document hItem created in the InitializeNewDocument method with the physical document located at 'pathname' to Documanage.

  While every document property has defaults, it is recommended the following properties should be set before saving a new document:

  AddedFrom
  DocumentName
  DocumentType

  **Parameters:**

  - *Pathname:* The physical file on disk to import into Documanage.

  **Return Value:** The hItem of the new document.

- ***SendTo***
  SendTo(hDestinationCursor as Long, hDestinationfolderitem as Long, Operation)

  This method takes the current document and moves, copies, or creates a shortcut to a given folder.

  **Parameters:**

  - *hDestinationCursor:* Destination folder's hCursor.
  - *hDestinationfolderitem:* Destination folder's hItem.

  **Operation:**

  The Documanage action code for the desired operation:
  EZP_ACTION_COPY
  EZP_ACTION_MOVE
  EZP_ACTION_MAKEREFERENCE

  **Return Value:** None

- ***SetCMFlag***
  Function SetCMFlag(Flag As Integer, Value As Integer) As Integer

  Sets the content management flag (specified by the "Flag" parameter (DMG_CM_APPROVED, DMG_CM_RELEASED, DMG_CM_OBSOLETE)) to the value (specified by "Value").

- ***SetCMFlags***
  Function SetCMFlags(Mask As Integer, Values As Integer) As Integer

  Sets the content management flags (specified by the "Mask" parameter (combinations of DMG_CM_APPROVED, DMG_CM_RELEASED, DMG_CM_OBSOLETE)) to the value (specified by "Value").

- ***SetValueByTitle***
  Sub SetValueByTitle(strTitle As String, vNewVal)

  Sets the value of an Extended Document Properties field. The title for an XDP field can be determined from the Titles array.

  **Parameters:**

  - *strTitle:* the Extended Document Properties field title.
  - *VNewVal:* a variant containing the field value.

- ***ShowDialog***
  ShowDialog(AllowEdit as Boolean) As Boolean

  This method shows the tabbed dialog box. It is recommended to implement your own custom dialog.

  **Parameters:**

  - *AllowEdit:* If you send TRUE for AllowEdit, the user will be able to change the document properties. Conversely, if you send FALSE, the user will be able to view the properties, but is restricted from modifying them. The AllowEdit parameter is ignored if the Documanage server prohibits editing of the document.

**NOTE:** The following document properties are not editable through the ShowDlg method: Added From, Added On, and Author.

- ***UpdateFile***
  UpdateFile(Filename As String)

  This method replaces the file on the server with the indicated local filename. Typically used after a GetFile(), this method only works on check-out documents owned by the current user. Also typically followed

by a check in.

**Parameters:**

- *Filename:* The local file to replace the remote file with.

- ***UpdateFileEx***
  UpdateFileEx(Filename As String, Flags As Long)

  See explanation for *GetFileEx*, below.

- ***GetFileEx***
  GetFileEx(Filename As String, Flags As Long) as String

  An extended version of UpdateFile and GetFile. While nearly identical to their counterparts these functions allow uploading and downloading of special files associated with the document or the document itself. In all but a few special cases the standard versions of these function should be used.

  **Parameters:**

  - *Filename:* The local file to replace the remote file with.

  - *Flags:* Combination of flags identifying the type of the file to transfer. Can be one of the following:

    EZP_XFER_DOC_CONTENTS
    EZP_XFER_DOC_ANNOTATIONS
    EZP_XFER_DOC_TYPE_OVERLAYS
    EZP_XFER_DOC_TYPE_ANNOTATIONS

  **Return Value:** See UpdateFile and GetFile for details.

# Examples

### *Initializing the Document Control*

```
PODocument.Documenttype

PODocument1.hSession = POSession1.hSession

PODocument1.hCursor   = POQuery1.hCursor

PODocument1.Initialize
```

### *Importing a Document into Documanage*

```
PODocument1.InitializeForNew Document(POTree1.CurrentFolderhItem)

PODocument1.AddedFrom = “FILE”

PODocument1.ShowDialog(True)

PODocument1.SaveNewDocument(strFileName)

POTree1.RefreshItem(0, False, True) 'Zero hItem indicates the current item
```

# Using the Document Control

To load a document into this control, to see and modify all its properties we use the following lines of code:

```
PODocument 1.hSession = POSession1.hSession

PODocument1.hCursor = POCursor1.hCursor

PODocument1.hItem = hItem ‘ Where hItem is ANY Documanage Item
```

handle that is of type ‘ EZP_Document.

You can get document hItems by several means; For example, using the Query’s GetRelativehItem(), or by employing the DocumentAction event on any of the Documanage Visual Controls. You may want to check the type of the hItem before setting it in the control by means of the Query’s ItemType() method.

# POFolder Control

The Folder control displays and allows you to edit a specific folder’s properties. There are multiple array properties that contain lists of all the internal Documanage properties for a particular control. Each element in each array corresponds to the identically indexed element in all the other arrays. The following is an example of the Folder control, with the contents of each of the control’s arrays underlined.

| Titles | Values | Data Types | Length | Edit Flags | Required Flags |
|---|---|---|---|---|---|
| Vendor Number | 12272 | INTEGER | NULL | 0 | 1 |
| Vendor Name | Weiss, Inc. | CHAR | 80 | 1 | 0 |
| Vendor Address | Clayton, NJ | CHAR | 500 | 1 | 0 |
| Vendor PhoneNum | 609/767-9876 | CHAR | 50 | 1 | 0 |

## Properties

- ***DataTypes*** **Data Type:** Long Array (Read Only)
  Returns the SQL data type of a given identified folder property. Types include:

EZP_TYPE_CHAR, EZP_TYPE_NUMERIC, EZP_TYPE_DECIMAL, EZP_TYPE_INTEGER, EZP_TYPE_SMALLINT, EZP_TYPE_FLOAT, EZP_TYPE_REAL, EZP_TYPE_DOUBLE, EZP_TYPE_VARCHAR, EZP_TYPE_DATE, EZP_TYPE_TIME, EZP_TYPE_TIMESTAMP, EZP_TYPE_LONGVARCHAR, EZP_TYPE_BINARY, EZP_TYPE_VARBINARY, EZP_TYPE_LONGVARBINARY.

- ***EditFlags*** **Data Type:** Boolean Array (Read Only)
  Indicates whether or not an identified folder property is editable.

- ***hCursor*** **Data Type:** Long (Read/Write)
  The hCursor of the folder to be opened or is currently open.

- ***hItem*** **Data Type:** Long (Read/Write)
  The hItem of the folder to be opened or is currently open.

- ***hSession*** **Data Type:** Long (Read/Write)
  The session of the folder to be opened or is currently open.

- ***IsIndexesOnly*** **Data Type:** Boolean (Read Only)
  Indicates if a control was initialized with InitializeIndexesOnly().

- ***IsTempItem*** **Data Type:** Boolean (Read Only)
  Indicates if a control was initialized with InitializeforNewItem().

- ***ItemCount*** **Data Type:** Long (Read Only)
  The number of elements in all the control's properties that are arrays. This corresponds to the number of indexes for that folder.

- ***Label*** **Data Type:** String (Read Only)
  The name of the folder.

- ***LastErrorNumber*** **Data Type:** Long (Read Only)
  The last error in Documanage.

- ***LastErrorSource*** **Data Type:** String (Read Only)
  The location of the last Documanage error. This is useful for reporting bugs to Oracle.

- ***Lengths*** **Data Type:** Long Array (Read Only)
  The maximum length of identified folder property values.

- ***NotesCount*** **DataType**: Long (Read Only)
  The total number of notes for the folder. Used in conjunction with InitializeForNotes().

- ***QualifiedIndex*** **Data Type:** String Array (Read Only)
  The database qualified index of identified folder property.

- ***RequiredFlags*** **Data Type:** Boolean Array (Read Only)
  Indicates whether or not identified folder property is required.

- ***Titles*** **Data Type:** String Array (Read Only)
  The title of the identified folder property.

- ***TranslateNullProperties*** **Data Type:** Boolean (Write Only)
  The TranslateNullProperties property determines how the Folder control outputs NULL Folder properties from the Documanage database. TranslateNullProperties defaults to True. To change this setting, the property must be set before calling any properties or methods to get Folder properties. Once set, TranslateNullProperties affects all subsequent calls to the control.

  When TranslateNullProperties is True, the Folder control translates NULL Folder properties from the database to empty strings if they are character data or to zero if they are numeric data when it returns them to users as variants.

  When TranslateNullProperties is set to False, the Folder control returns NULL Folder properties from the database as NULL variants without translating them to zeros or to empty strings.

  The control returns Folder properties using the Values( ) property and the GetValueByTitle ( ) method.

**NOTE:** You can set NULL values for Folder Properties by passing in NULL variants using the Values ( ) property or the SetValueByTitle ( ) method. Refer to the descriptions of the Values( ) property on page 48 and the SetValueByTitle ( ) method on page 52.

- ***Values*** **Data Type:** Variant Array (Read/Write)
  Gets or sets the current value of an identified folder property.

- ***LevelNumber*** **Data Type:** Long (Read Only)
  Indicates what level of the cabinet the folder exists on.

- ***KeyString*** **Data Type:** String (Read Only)
  A unique FolderFilter suitable for use in the Query control that will open a cabinet showing only this folder.

- ***VisibleFlags*** **Data Type:** Boolean Array (Read Only)
  Indicates whether or not an identified folder property should be displayed.

- ***IsKey*** **Data Type:** Boolean Array (Read Only)
  Indicates whether or not an identified folder property is a key.

## Methods

- ***AddNote(description As String, [optional] addedby As Variant, [optional] additionalinfo As Variant)***
  Adds a new note to the folder.

  **Parameters**:

  - *description*: Text of the note.
  - *addedBy*: User who adds the note.
  - *additionalInfo*: Additional information for the note.

- ***DeleteNote(index As Long)***
  This method marks the specified note as deleted.

**Parameters**:

- *index*: Position of the note in notes list. From 0 to NotesCount - 1

- ***Initialize()***
  This method opens a folder in the control. You determine which folder by the hCursor, hItem, and hSession handles.

- ***InitializeForNewFolder***
  InitializeForNewFolder

  This method creates a new, empty folder in a cabinet. By setting the properties and saving the folder, the new folder will contain all the data it needs. Folders will not actually be inserted into the systems until a save occurs without any errors.

- ***InitializeForNotes()***
  This method initializes the control to obtain information about notes associated with this folder.

- ***InitializeIndexesOnly()***
  This method is similar to Initialize, but only retrieves data about the Folder properties, not their values.

- ***Save()***
  This method saves the changes (if any) to the open folder.

- ***InitializeTableIndexes***
  InitializeTableIndexes(strTable As String)

  Similar to InitializeIndexesOnly but can take a table name instead of a hItem of an active folder. This method can be used in conjunction with calls to the Query control to retrieve index information about a Cabinet without actually opening it. Note the Values array is not populated since you are retrieving generic information about a table not a specific folder.

**Parameters:**

- *strTable:* The table to retrieve indexes for.

- ***GetValueByTitle***
  GetValueByTitle(strFolderPropTitle As String) As Variant

  Allows the Value's array to accessed not by index but by the Title of that value. Useful when you know ahead of time the name of the index but do not want to write code to find that index in the arrays.

  **Parameters:**

  - *strFolderPropTitle:* The Title of the index which we want the value for. Titles for indexes can be retrieved by the Titles property.

  **Return Value:** The value of that index.

- ***IsNoteDeleted(index As Long) As Boolean***
  This method is used to determine if a note is marked as deleted.

  **Parameters**:

  - *index*: Position of the note in notes list. From 0 to NotesCount - 1

  **Return Value**: Returns a Boolean which indicates whether the note is marked for deletion or not.

- ***NoteAddedBy(index As Long) As String***
  This method is used to obtain the name of the user who added the specified note.

  **Parameters**:

  - *index*: Position of the note in notes list. From 0 to NotesCount - 1

**Return Value**: Returns a String indicating the name of user who added the specified note.

- ***NoteAddedOn***
  NoteAddedOn(index As Long) As String

  This method is used to obtain the date and time when a note was added.

  **Parameters**:

  - *index*: Position of the note in notes list. From 0 to NotesCount - 1

  **Return Value**: Returns a String containing the date and time when the specified note was added.

- ***NoteAdditionalInfo (index As Long) As String***
  This method is used to obtain additional information for a note.

  **Parameters**:

  - *index*: Position of the note in notes list. From 0 to NotesCount - 1

  **Return Value**: Returns a String containing additional information for the specified note.

- ***NoteDescription (index As Long) As String***
  This method is used to obtain the description of a note.

  **Parameters:**

  - *index*: Position of the note in notes list. From 0 to NotesCount - 1

  **Return Value**: Returns a String containing description of the specified note.

- ***SetValueByTitle***
  SetValueByTitle(strFolderPropTitle As String, vNewVal As Variant)

  Allows the Value's array to be filled in not by index but by the Title of that value. Useful when you know ahead of time the name of the index but do not want to write code to find that index in the arrays.

  **Parameters:**

  - *strFolderPropTitle:* The Title of the index which we want the value for. Titles for indexes can be retrieved by the Titles property.
  - *vNewVal:* The value for that index.

- ***Delete ()***
  Delete the current folder from the system.

- ***RefreshContents ()***
  This call will insure that the documents in this folder are up to date. This function is designed to be used as replacement for RefreshItem in the Folder control.

- ***UndeleteNote(index As Long)***
  This method unmarks a note marked as deleted.

  **Parameters:**

  - *index*: Position of the note in notes list. From 0 to NotesCount - 1

## Using the Folder Control

To load a folder into this control, to see and modify all its properties, we use the following four lines of code:

```
POFolder1.hSession = POSession1.hSession

POFolder1.hCursor = POCursor1.hCursor

POFolder1.hItem = hItem

‘ Where hItem is ANY Documanage Item handle that is of type: EZP_FOLDER.
```

You can get Folder hItems by several different means; For example, using the Query’s GetRelativehItem() or by means of the FolderAction event for any Documanage Visual Controls. You may want to check the type of the hItem before setting it in the control by means of the Query’s ItemType() method.

# POProject Control

## Properties

- ***AddedOn*** **DataType:** String (Read Only)
  This allows the user to get an AddedOn Data String.

- ***AssignedTo*** **DataType:** String (Read Only)
  The user to whom the project has been assigned.

- ***DueDate*** **DataType:** String (Read/Write)
  DueDate of the project.

- ***hCursor*** **DataType:** Long (Read Only)
  The hCursor corresponding to the hItem.

- ***hItem*** **DataType:** Long (Read Only)
  The hItem of the project to be opened or is currently open.

- ***hSession*** **DataType:** Long (Read/Write)
  The Current hSession with the Documanage Server. hSession is a value passed from the Session control, which identifies the login to the Documanage Server.

- ***ItemCount*** **DataType:** Long (Read Only)
  The number of elements in all the control's properties that are arrays. This corresponds to the number of indexes for that project.

- ***Label*** **DataType:** String (Read Only)
  Gives the name of the Project.

- ***LastErrorNumber*** **DataType:** Long (Read Only)
  Gives the Last Error Number of the Server.

- ***LastErrorSource*** **DataType:** String (Read Only)
  Description of the Last Error.

- ***LockedBy*** **DataType:** String (Read Only)
  Indicates the user who locked the project.

- ***NotesCount*** **DataType**: Long (Read Only)
  Indicates the total number of notes for the project. Used in conjunction with InitializeForNotes().

- ***ProjectID*** **DataType:** Long (Read Only)
  A unique ID, to identify a particular project.

- ***ProjectName*** **DataType:** String (Read/Write)
  Gets or Sets a description about the project.

- ***StartedBy*** **DataType:** String (Read Only)
  Indicates the user who started the project.

- ***StartedOn*** **DataType:** String (Read Only)
  Indicates the project start date.

- ***TotalHistoryRecords*** **Type:** Long (Read Only)
  Indicates the total number of history records of a project.

- ***TotalNextTasks(Long CurrentTaskID)*** **DataType:** Long (Read Only)
  Indicates the total number of tasks remaining from the current task.

- ***TotalPendingTasks*** **DataType:** Long (Read Only)
  Indicates total pending tasks for that user. Used in conjunction with InitializeForPendingTasksOnly().

- ***TotalTasks*** **DataType**: Long (Read Only)
  Indicates total task queues (tasks that have pending or suspended projects and empty tasks with no projects also) for the current user. Used in conjunction with InitializeAllTaks().

- ***TotalWorkflowCabinets*** **Data Type:** Long (Read Only)
  Indicates total number of workflow cabinets in the system.

- ***TranslateNullProperties*** **Data Type:** Boolean (Write Only)
  The TranslateNullProperties property determines how the Project control outputs NULL Project properties from the Documanage database. TranslateNullProperties defaults to True. To change this setting, the property must be set before calling any properties or methods to get Project properties. Once set, TranslateNullProperties affects all

subsequent calls to the control.

When TranslateNullProperties is True, the Project control translates NULL Project properties from the database to empty strings if they are character data or to zero if they are numeric data when it returns them to users as variants.

When TranslateNullProperties is set to False, the Project control returns NULL Project properties from the database as NULL variants without translating them to zeros or to empty strings.

The control returns Project properties using the GetValue ( ) method.

**NOTE:** You can set NULL values for Project properties by passing in NULL variants using the SetValue ( ) method. Refer to the description of the SetValue ( ) method on page 77.

- ***Workflow*** **DataType:** String (Read Only)
  Indicates the current workflow.

## Sample code: Adding a Workflow project:

```
POProject1.hSession = POSession1.hSession
POProject1.hCursor = POQuery1.hCursor
POProject1.hItem = POQuery1.hRootItem

POProject1.InitializeForNewProject ""

POProject1.SetValue 0, "107-99-3333"
POProject1.SetValue 1, "John"
POProject1.SetValue 2, "Smith"
POProject1.SetValue 3, "301-589-6300"
POProject1.SetValue 4, "8455 Colesville Rd"
POProject1.SetValue 5, "Silver Spring"
```

```
POProject1.SetValue 6, "MD"
POProject1.SetValue 7, "20910"
POProject1.SetValue 8, 1
POProject1.DueDate = "2001-02-04 00:00:00"
POProject1.Priority = 1
POProject1.AssignedTo = "EZPOWER"
POProject1.ProjectName = "This is a project inserted programatically"
POProject1.Save
```

## CheckOut a Project:

```
POProject1.hSession = POSession1.hSession
POProject1.hCursor = POQuery1.hCursor
POProject1.hItem = hItem
POProject1.Initialize
POProject1.CheckOut
```

## Methods

- ***AddNote(description As String, [optional] addedby As Variant, [optional] additionalinfo As Variant)***

  Adds a new note to the project.

  **Parameters:**

  - *description*: Text of the note.
  - *addedBy*: User who adds the note.
  - *additionalInfo*: Additional information for the note.

- ***CheckIn (lCheckInType As Long, strUserName As String, strComments As String, lDecisionBranch As Long)***

  This method checks back a project.

**Parameters:**

- *lCheckInType:* A flag indicating how to check in a project. It can take the values:

  EZP_ACTION_CHECKIN_PROJ = 29
  EZP_ACTION_CHECKIN_PROJ_MOVE = 30
  EZP_ACTION_CHECKIN_PROJ_ASSIGN = 31

- *strUserName:* User name of the user checking in the project
- *strComments:* CheckIn Comment
- *lDecisionBranch:* Next TaskID in case of human decision.

**Return Value:** none

- ***CheckOut()***
  This method checks out a project.

  **Parameters:** none

  **Return Value:** none

- ***DeleteNote(index As Long)***
  This method soft deletes a note (marks the note for deletion).

  **Parameters**:

  - *index*: Position of the note in notes list. From 0 to NotesCount - 1

- ***GetBranches(decTaskId As Long) As Long***
  This method gives the number of branches for the workflow at a given TaskID.

  **Parameters:**

- *DecTaskId:* TaskID of the current task.

**Return Value:** Returns a long value. Total number of branches.

- ***GetBranchId(index As Long) As String***
  Gives the TaskID(s) by index from 0 to Total Number of braces – 1.

  **Parameters:**

  - *index:* Index of the branch TaskID

  **Return Value:** Returns a String having the branch TaskID.

- ***GetBranchName(index As Long) As String***
  This method gives the branch name given the index.

  **Parameters:**

  - *index:* Index of the branch TaskID

  **Return Value:** Returns a String having the branch name.

- ***GetKeyString() As String***
  A unique ProjectFilter suitable for use in the Query control that will open a project cabinet showing only this folder.

  ***Parameters: none***

  **Return Value:** Returns the KeyString.

- ***GetLength(lIndex As Long) As Long***
  The maximum length of an identified project property values.

  **Parameters:**

- *lIndex:* Project property position in the control.

**Return Value:** Returns the maximum length of the project property.

- ***GetLocked(lIndex As Long)***
  Not Used. For future use.

- ***GetNextTaskDesc(index As Integer) As String***

  **Parameters:**

  - *lIndex:* Position in the control. From 0 to TotalNextTasks - 1

  **Return Value:** Returns a String with the description of the task.

- ***GetNextTaskID(index As Integer) As Long***

  **Parameters:**

  - *lIndex:* Position in the control. From 0 to TotalNextTasks - 1

  **Return Value:** Returns a Long with the next TaskID.

- ***GetNextTaskManager(index As Integer) As String***

  **Parameters:**

  - *lIndex* Position in the control. From 0 to TotalNextTasks - 1

  **Return Value:** Returns a String with the next Task Manager.

- ***GetPendingTask(lIndex As Long, ByRef lTaskID As Long, ByRef lTaskCount As Long, ByRef strInstructions As String, ByRef strManager As String, ByRef strDescription As String, ByRef***

***strCabinet As String, ByRef strFilter As String, ByRef strWorkflow As String, ByRef strWorkflowDescription As String)***

This method is used to get the attributes of a pending task queue at the specified index in the pending tasks list. This method is valid only after calling InitializePendingTasksOnly() that initializes the control with a list of pending tasks for the current user. Note that ByRef parameters are return values.

**Parameters:**

- lIndex: Valid range from 0 to value of the property TotalPendingTasks – 1. Index of the pending task to retrieve information for in the pending tasks list.
- lTaskID: Unique Identifier of the task.
- lTaskCount: Total Number of pending projects in the task.
- strInstructions: Instructions for the task.
- strManager: Name of the manager of the task.
- strDescription: Description of the task.
- strCabinet: Name of the workflow cabinet for the workflow to which the task belngs.
- strFilter: Query filter for the task.
- strWorkflow: Name of the workflow to which the task belongs.
- strWorkflowDescription: Description of the workflow to which the task belongs.

**Return Value:** None, but note that ByRef parameters are return values.

- ***GetPendingTaskV2(lIndex As Long, ByRef lTaskID As Long, ByRef lPendingProjectsCount As Long, ByRef strInstructions As String, ByRef strManager As String, ByRef strDescription As String, ByRef strCabinet As String, ByRef strFilter As String, ByRef strWorkflow As String, ByRef strWorkflowDescription As String, ByRef lMaxTaskDuration As Long, ByRef lMaxWorkDuration As Long, ByRef sDeactivated As Short, ByRef lContainerID As Long)***

  This method is used to get the attributes of a pending task queue at the specified index in the pending tasks list. This method is valid only after calling InitializePendingTasksOnly() that initializes the control with a list of pending tasks for the current user. Note that ByRef parameters are return values. This method is a newer version of GetPendingTask() method and this method returns some more new attributes of the pending task along with the other properties returned by GetPendingTask() method.

**Parameters:**

- lIndex: Valid range from 0 to value of the property TotalPendingTasks – 1. Index of the pending task to retrieve information for in the pending tasks list.
- lTaskID: Unique Identifier of the task.
- lPendingProjectsCount: Total Number of pending projects in the task.
- strInstructions: Instructions for the task.
- strManager: Name of the manager of the task.
- strDescription: Description of the task.
- strCabinet: Name of the workflow cabinet for the workflow to which the task belngs.
- strFilter: Query filter for the task.
- strWorkflow: Name of the workflow to which the task belongs.
- strWorkflowDescription: Description of the workflow to which the task belongs.
- lMaxTaskDuration: The maximum amount of time a project should remain in a task (the time from when the project enters the task's queue to when it is forwarded to the next task).
- lMaxWorkDuration: The maximum amount of time spent actually processing a project within a task (the time from when the project is removed from a task's queue to when it is forwarded to the next task).
- sDeactivated: Flag that specifies if the task is decactivated. 1 – Deactivated, 0 – Not Deactivated.
- lContainerID: Unique Identifier of the task container to which the task belongs. It may be zero if the task does not belong to any container

**Return Value:** None, but note that ByRef parameters are return values

- ***GetPerformedBy(lIndex As Long) As String***
  Gives the user name who performed the specified task.

  **Parameters:**

  - *lIndex* Position in the control. From 0 to TotalHistoryRecords - 1

  **Return Value:** Returns a String with the user name who performed the task.

- ***GetResultID(lIndex As Long) As Long***
  Returns a result TaskID.

  **Parameters:**

  - *lIndex* Position in the control. From 0 to TotalHistoryRecords - 1

  **Return Value:** Returns a Long with Result TaskID.

- ***GetRouteTo(lIndex As Long) As String***
  Returns the user name to whom the project was routed to.

  **Parameters:**

  - *lIndex* Position in the control. From 0 to TotalHistoryRecords - 1

  **Return Value:** Returns a String with the user name.

- ***GetSince(lIndex As Long) As String***
  Returns a date string, indicating when the project was locked.

  **Parameters**:

  - *lIndex* Position in the control. From 0 to TotalHistoryRecords - 1

**Return Value:** Returns a String with the date string.

- ***GetSuspendedTask(lIndex As Long, ByRef lTaskID As Long, ByRef lTaskCount As Long, ByRef strInstructions As String, ByRef strManager As String, ByRef strDescription As String, ByRef strCabinet As String, ByRef strFilter As String, ByRef strWorkflow As String, ByRef strWorkflowDescription As String)***

  This method is used to get the attributes of a suspended task queue at the specified index in the suspended tasks list. This method is valid only after calling InitializeSuspendedTasksOnly() that initializes the control with a list of suspended tasks for the current user. Note that ByRef parameters are return values.

**Parameters:**

- lIndex: Valid range from 0 to value of the property TotalSuspendedTasks – 1. Index of the suspended task to retrieve information for in the suspended tasks list.
- lTaskID: Unique Identifier of the task.
- lTaskCount: Total Number of suspended projects in the task.
- strInstructions: Instructions for the task.
- strManager: Name of the manager of the task.
- strDescription: Description of the task.
- strCabinet: Name of the workflow cabinet for the workflow to which the task belngs.
- strFilter: Query filter for the task.
- strWorkflow: Name of the workflow to which the task belongs.
- strWorkflowDescription: Description of the workflow to which the task belongs.

**Return Value:** None, but note that ByRef parameters are return values.

- ***GetSuspendedTaskV2(lIndex As Long, ByRef lTaskID As Long, ByRef lSuspendedProjectsCount As Long, ByRef strInstructions As String, ByRef strManager As String, ByRef strDescription As String, ByRef strCabinet As String, ByRef strFilter As String, ByRef strWorkflow As String, ByRef strWorkflowDescription As String, ByRef lMaxTaskDuration As Long, ByRef lMaxWorkDuration As Long, ByRef sDeactivated As Short, ByRef lContainerID As Long)***

  This method is used to get the attributes of a suspended task queue at the specified index in the suspended tasks list. This method is valid only after calling InitializeSuspendedTasksOnly() that initializes the control with a list of suspended tasks for the current user. Note that ByRef parameters are return values. This method is a newer version of GetSuspendedTask() method and this method returns some more new attributes of the suspended task along with the other properties returned by GetSuspendedTask() method

  **Parameters**:

  - lIndex: Valid range from 0 to value of the property TotalSuspendedTasks – 1. Index of the suspended task to retrieve information for in the suspended tasks list.
  - lTaskID: Unique Identifier of the task.
  - lSuspendedProjectsCount: Total Number of suspended projects in the task.
  - strInstructions: Instructions for the task.
  - strManager: Name of the manager of the task.
  - strDescription: Description of the task.
  - strCabinet: Name of the workflow cabinet for the workflow to which the task belngs.

- strFilter: Query filter for the task.
- strWorkflow: Name of the workflow to which the task belongs.
- strWorkflowDescription: Description of the workflow to which the task belongs.
- lMaxTaskDuration: The maximum amount of time a project should remain in a task (the time from when the project enters the task's queue to when it is forwarded to the next task).
- lMaxWorkDuration: The maximum amount of time spent actually processing a project within a task (the time from when the project is removed from a task's queue to when it is forwarded to the next task).
- sDeactivated: Flag that specifies if the task is decactivated. 1 – Deactivated, 0 – Not Deactivated.
- lContainerID: Unique Identifier of the task container to which the task belongs. It may be zero if the task does not belong to any container

**Return Value:** None, but note that ByRef parameters are return values.

- ***GetTask(lIndex As Long, ByRef lTaskID As Long, ByRef lPendingProjectsCount As Long, ByRef lSuspendedProjectsCount Long, ByRef strInstructions As String, ByRef strManager As String, ByRef strDescription As String, ByRef strCabinet As String, ByRef strFilter As String, ByRef strWorkflow As String, ByRef strWorkflowDescription As String, ByRef lMaxTaskDuration As Long, ByRef lMaxWorkDuration As Long, ByRef sDeactivated As Short, ByRef lContainerID As Long)***

  This method is used to get the attributes of a task at the specified index in the all tasks list. This method is valid only after calling InitializeAllTasks() that initializes the control with a list of all tasks for the current user. Note that ByRef parameters are return values. This method returns both the pending projects count and suspended projects

count (these counts will be zero for empty task queues) for the returned task.

**Parameters:**

- lIndex: Valid range from 0 to value of the property TotalTasks – 1. Index of the task to retrieve information for in the all tasks list.
- lTaskID: Unique Identifier of the task.
- lPendingProjectsCount: Total Number of pending projects in the task.
- lSuspendedProjectsCount: Total Number of suspended projects in the task.
- strInstructions: Instructions for the task.
- strManager: Name of the manager of the task.
- strDescription: Description of the task.
- strCabinet: Name of the workflow cabinet for the workflow to which the task belngs.
- strFilter: Query filter for the task.
- strWorkflow: Name of the workflow to which the task belongs.
- strWorkflowDescription: Description of the workflow to which the task belongs.
- lMaxTaskDuration: The maximum amount of time a project should remain in a task (the time from when the project enters the task's queue to when it is forwarded to the next task).
- lMaxWorkDuration: The maximum amount of time spent actually processing a project within a task (the time from when the project is removed from a task's queue to when it is forwarded to the next task).
- sDeactivated: Flag that specifies if the task is decactivated. 1 – Deactivated, 0 – Not Deactivated.

- lContainerID: Unique Identifier of the task container to which the task belongs. It may be zero if the task does not belong to any container

**Return Value:** None, but note that ByRef parameters are return values

- ***GetTaskDescription(lIndex As Long) As String***
  Not used. For future use.

- ***GetTaskTeam(lTaskID As Long, index As Integer) As String***
  Returns the TeamID of a specified Task.

  **Parameters**:

  - *lTaskID:* TaskID of the task in question.
  - *index:* Position in the control. 0 to TotalTeamsForTask – 1.

  **Return Value:** Returns a String with TeamID.

- ***GetTaskType(TaskId As Long) As Long***
  Returns a long indicating the TaskType.

  **Parameters:**

  - *TaskId:* Task id of the task in question

  **Return Value:** Returns a Long with Task Type.

- ***GetTitle(lIndex As Long) As String***
  This method gives the Title of a project property by index

  **Parameters:**

  - *lIndex* Position in the control. From 0 to ItemCount - 1

**Return Value:** Returns a String with the Title of the project property.

- ***GetTotalTeamsForTask(lTaskID As Long) As Integer***
  This method gives the Total number of Teams for a specified task.

  **Parameters:**

  - *lTaskID:* Task id of the task in question.

  **Return Value:** Returns an Integer with the TotalTeams for a task.

- ***GetType(lIndex As Long) As Long***
  This method returns the Database type of a particular project property.

  **Parameters:**

  - *lIndex* Position in the control. From 0 to ItemCount - 1

  **Return Value:** Returns a Long with the Database datatype of the project property.

- ***GetValue(lIndex As Long) As VARIANT***
  This method gives the value of a project property by index.

  **Parameters:**

  - *lIndex* Position in the control. From 0 to ItemCount - 1

  **Return Value:** Returns a VARIANT with the value of the project property.

- ***GetWFCabinetDescription(lIndex As Long) As String***
  This method gives description of a workflow cabinet.

- **Parameters:**
  - *lIndex* Position in the control. From 0 to TotalWorkflowCabinets - 1

  **Return Value:** Returns a String with a description of the workflow cabinet.

- ***GetWFWorkTableName(lIndex As Long) As String***
  This method gives the table name on which the workflow is based.

  **Parameters:**

  - *lIndex* Position in the control. From 0 to TotalWorkflowCabinets - 1

  **Return Value:** Returns a String with the table name on which the workflow cabinet is based on.

- ***GetWorkflowCabinetName(lIndex As Long) As String***
  This method gives the name of the workflow cabinet by index

- **Parameters:**
  - *lIndex* Position in the control. From 0 to TotalWorkflowCabinets - 1

  **Return Value:** Returns a String with the Workflow cabinet name.

- ***GetWorkflowID(lIndex As Long) As String***
  This method gives the workflow ID by index.

  **Parameters:**

  - *lIndex* Position in the control. From 0 to TotalWorkflowCabinets - 1

  **Return Value:** Returns a String with the Workflow id.

- ***GetWorkflowMaxProjectDuration(lIndex As Long) As Long***

  This method gives the MaxProjectDuration attribute of the specified workflow in the list of all workflows for the current user.

  **Parameters:**

  - lIndex: Valid range from 0 to value of the property TotalWorkflowCabinets - 1. Index of the workflow to retrieve information for in the workflows list for the current user.

  **Return Value:** Returns the MaxProjectDuration attribute of the specified workflow

- ***Initialize()***

  This method initializes the project control. You can determine which project by hCursor, hSession & hItem handles.

- ***InitializeAllTasks()***

  This method initializes the control to get information about all task queues (tasks that have pending or suspended projects and empty tasks with no projects also) for the current user.

- ***InitializeForNewProject(strWorkflowCabinetName As String)***

  This method creates a new, empty project folder in a workflow cabinet. By setting the properties and saving the project folder, the new project folder will contain all the data it needs. Project folders will not actually be inserted into the systems until a save occurs without any errors.

  **Parameters:**

  - *strWorkflowCabinetName:* Name of the workflow cabinet to which a project folder is to be added.

- ***InitializeForNotes()***
  This method initializes the control to get information about notes associated with the project.

- ***InitializePendingTasksOnly()***
  This methods initializes the control to get information about pending tasks.

- ***IsKey(lIndex As Long) As Boolean***
  This method determines if a particular project property is a key or not.

  **Parameters**:

  - *lIndex:* Position in the control. From 0 to ItemCount - 1

  **Return Value:** Returns a Boolean, indicating the key.

- ***IsNoteDeleted(index As Long) As Boolean***
  This method is used to determine if a note is marked as deleted or not.

  **Parameters**:

  - *index*: Position of the note in notes list. From 0 to NotesCount - 1.

  **Return Value**: Returns a Boolean value which indicates whether the specified note is marked as deleted or not.

- ***IsRequired(lIndex As Long) As Boolean***
  This method determines whether or not the identified project property is required.

  **Parameters:**

  - *lIndex* Position in the control. From 0 to ItemCount - 1

**Return Value:** Returns a Boolean, indicating if the property is required to be set.

- ***IsTaskHumanDec(TaskId As Long) As Boolean***
  This method is used to determine if a particular task is of type, human decision.

  **Parameters:**

  - *TaskId:* Task id of the task in question.

  **ReturnValue:** Returns a Boolean, indicating if the Task is of type human decision.

- ***NoteAddedBy(index As Long) As String***
  This method is used to obtain the name of user who added the note.

  **Parameters:**

  - *index*: Position of the note in notes list. From 0 to NotesCount - 1.

  **Return Value**: Returns a String indicating name of user who added the specified note.

- ***NoteAddedOn(index As Long) As String***
  This method is used to obtain the date and time when the note was added.

  **Parameters**:

  - *index*: Position of the note in notes list. From 0 to NotesCount - 1.

  **Return Value**: Returns a String with the date and time when the specified note was added.

- ***NoteAdditionalInfo (index As Long) As String***
  This method is used to obtain additional information for a note.

  **Parameters:**

  - *index*: Position of the note in notes list. From 0 to NotesCount - 1.

  **Return Value**: Returns a String with additional information for the specified note.

- ***NoteDescription (index As Long) As String***
  This method is used to obtain the description of a note.

  **Parameters:**

  - *index*: Position of the note in notes list. From 0 to NotesCount - 1.

  **Return Value:** Returns a String with description of the specified note.

- ***Save()***
  This method saves changes to project properties (if any). This method must be to add a project to the workflow cabinet.

- ***SetValue(lIndex As Long, vNewValue)***
  This method allows to set project properties by index.

  **Parameters:**

  - *lIndex:* Position in the control. From 0 to ItemCount – 1.
  - *vNewValue:* New value to be set for that index. This is of type VARIANT.

- ***UndeleteNote(index As Long)***
  This method unmarks a note marked as deleted.

**Parameters:**

- *index*: Position of the note in notes list. From 0 to NotesCount - 1.

# POSession Control

The Session control is used to begin or end a user session. This control is always required to be included in a project. You must connect before any other Documanage controls can be used (you need the hSession handle returned as a result of a Connect()). Before your application quits or when you are done using Documanage functionality you must call Connect() again to disconnect.

## Standard Properties

- ***hSession*** **Data Type:** Long (Read Only)
  The current hSession sent by the Documanage server. This will be set after a successful call to Connect().

- ***LastErrorNumber*** **Data Type:** Long (Read Only)
  The last Documanage error.

- ***LastError Source*** **Data Type:** String (Read Only)
  The location of the last Documanage error. This is useful for reporting bugs to Oracle.

- ***License*** **Data Type:** Integer (Read Only)
  Currently Unimplemented.

- ***Password*** **Data Type:** String (Read/Write)
  The password of the specified user logging into the Documanage server. See the Documanage documentation on how Documanage uses NT security.

- ***UserName*** ***Data Type:*** String (Read/Write)
  The user name to login to the Documanage server. This is a valid NT

Username on the specified Domain. See the core Documanage documentation on how Documanage uses NT security.

- ***UsedSessionCredentials*** **Data Type:** Boolean (Read Only)
  Indicates if session credentials are used to login to the Documanage server.

- ***Domain*** **Data Type:** String (Read/Write)
  The NT Domain of the specified user. See the core Documanage documentation on how Documanage uses NT security.

- ***RouterName*** **Data Type:** String (Override Only)
  ***RouterAddress*** **Data Type:** String (Override Only)
  ***RouterProtocol*** **Data Type:** String (Override Only)
  ***RouterEndPoint*** **Data Type:** String (Override Only)
  You can set these properties to override the default settings. You cannot view these properties, only create and view the new properties.

  For more information see the Documanage documentation on the POFFICE.INI's [Router] section.

- ***ComputerName*** **Data Type:** String (Override Only)
  The name of the workstation as shown on the Documanage server console. The default value is the name of the Win95/NT workstation. It is recommended that this default be used. You can set this property to override the default settings.

## Router Only Properties

This Property should only be called when the Session Control is connected via Connect(EZP_ROUTER_ACCESS_ONLY , false) call. See Connect for details.

- ***TotalServers*** **Data Type:** Long (Read Only)
  The total number of servers currently connected. Useful in connection with GetServerInfo().

## Standard Methods

- ***Connect***
  Connect(Mode As Integer, ShowDialog As Boolean) As Long

  This method establishes and disconnects a session with the Documanage server. Typically you call this twice; once to connect, once to disconnect.

  The hSession property contains a valid Session handle after Connect() has been called with Mode =…FULL. hSession is viewed by all other controls. Call Connect() with Mode=EZP_ACCESS_NONE to disconnect.

  **Parameters:**

  - *Mode:* Can be set to one of the following values:

    EZP_ACCESS_FULL – Connect to a Documanage Server as the indicated user. EZP_ACCESS_NONE – Disconnect from a Documanage Server. This must be called before your application exits.

    EZP_ACCESS_VERIFY – Used to confirm if a user is still connected to a server.

    EZP_ACCESS_ROUTER_ONLY – Allows connection not to a Documanage Server but to the Documanage Router. Some calls in this control can only be used while connected in this manner. Disconnecting is not necessary in this mode.

- *ShowDialog:* Used to silently or actively create a session. If True, the control will display a dialog box with the UserName, Password and Domain properties. If False, the control will silently obtain a session for the user based UserName, Password and Domain properties. No dialog box will be shown in this instance.

**Return Value:** Returns a Long value. This is an error code. A non-error successful call will return EZP_SUCCESS (zero). Any other value is a Documanage Error code. This method call may optionally throw a VB error. To be safe, a check of this return value and an "On Error" VB handler should be used to trap errors.

- ***GetTotalWorkflowSchemas*** - DEPRECATED
Returns the number of workflow schemas stored in the Documanage server.

  **Parameters:** None.

  **Return Value**: Returns a Long value, the workflow count.

- ***GetWorkflowSchemaInfo***-DEPRECATED
Retrieves the workflow identified by the workflowIndex parameter.

  **Parameters:**

  workflowIndex (long) – the workflow's collection identifier

  workflowID (String) – On return, this parameter holds the workflow's identifier

  workflowDescription (String) – on return, this parameter holds the workflow's description

  dsnTable (String) – on return, this parameter holds the data source table name

powerCabinet (String) – on return, this parameter holds the workflow's cabinet name

projectDuration (long) – on return, this parameter holds the max project duration for the workflow

**Return Values:** Returns a long value: EZP_SUCCESS on success or EZP_INDEX_NOT_FOUND if the index is out of range.

- ***GetTotalTaskSchemasForWorkflow-*** DEPRECATED
  Returns the number of tasks in the workflow identified by the workflowIndex parameter.

  **Parameters:**

  workflowIndex (long) – the workflow's collection identifier

  **Return Values:** Returns a long value, the task count.

- ***GetTaskSchemaInfo -*** DEPRECATED
  Retrieves the Task identified by the workflowIndex and taskIndex parameters.

  **Parameters:**

  workflowIndex (long) – the parent workflow's collection identifier

  taskIndex (long) – the task's collection identifier

  taskID (long) – on return, contains the task's Documanage identifier

  taskType (long) – on return, contains the task's type identifier

  containerID (long) – on return, contains the task's container identifier or 0 if the task has no parent container

taskDescription (String) – on return, contains the human readable name for the task

taskInstructions (String) – on return, contains the steps to take to complete the task for a project

taskManager (String) – on return, contains the user id of the person managing the task

maxTaskDuration (long) – on return, contains the maximum time a project is allowed in the particular task

maxWorkDuration (long) – on return, contains …

deactivated (long) - on return, contains the deactivation status of the task

**Return Values:** Returns a long value: EZP_SUCCESS on success or EZP_INDEX_NOT_FOUND if an index is out of range.

- ***GetTotalTeamSchemasForTask*** - DEPRECATED
  Returns the number of teams for the task identified by the workflowIndex and taskIndex parameters.

  **Parameters:**

  workflowIndex (long) – the parent workflow's collection identifier

  taskIndex (long) – the task's collection identifier

  **Return Values**: Returns a long value, the team count.

- ***GetTaskTeamSchemaInfo*** - DEPRECATED
  Retrieves the Team identified by the index parameters.

**Parameters:**

workflowIndex (long) – the parent workflow's collection identifier

taskIndex (long) – the parent task's collection identifier

teamIndex (long) – the team's collection identifier

teamName (String) – on return, contains the team's group identifier

**Return Values:** Returns a long value: EZP_SUCCESS on success or EZP_INDEX_NOT_FOUND if an index is out of range.

- ***GetTotalContainerSchemasForWorkflow*** **-** DEPRECATED
  Returns the number of Containers in the workflow identified by the workflowIndex parameter.

  **Parameters:**

  workflowIndex (long) – the parent workflow's collection identifier

  **Return Values:** Returns a long, the container count.

- ***GetContainerSchemaInfo*** **-** DEPRECATED
  Retrieves the Container identified by the workflowIndex and containerIndex parameters.

  **Parameters:**

  workflowIndex (long) – the parent workflow's collection identifier

  containerIndex (long) – the container's collection identifier

containerID (long) – on return, contains the container's Documanage identifier

parentID (long) – on return, contains the container's parent container identifier or 0 if no parent container exists

containerName (String) – on return, contains the human readable name for the container

containerDescription (String) – on return, contains the description of the task

containerManager (String) – on return, contains the user id of the person managing the container

**Return Values:** Returns a long value: EZP_SUCCESS on success or EZP_INDEX_NOT_FOUND if an index is out of range.

- ***GetTotalTeamSchemasForContainer*** **-** DEPRECATED
  Returns the number of teams for the container identified by the workflowIndex and containerIndex parameters.

  **Parameters:**

  workflowIndex (long) – the parent workflow's collection identifier

  containerIndex (long) – the container's collection identifier

  **Return Values:** Returns a long value, the team count.

- ***GetContainerTeamSchemaInfo*** **-** DEPRECATED
  Retrieves the Team identified by the index parameters.

**Parameters:**

workflowIndex (long) – the parent workflow's collection identifier

containerIndex (long) – the parent container's collection identifier

teamIndex (long) – the team's collection identifier

teamName (String) – on return, contains the team's group identifier

**Return Values:** Returns a long value: EZP_SUCCESS on success or EZP_INDEX_NOT_FOUND if an index is out of range.

## Router Only Methods

These Methods should only be called when the Session Control is connected via *Connect(EZP_ACCESS_ROUTER_ONLY, false)* call. See *Connect* for details.

- ***GetServerInfo***
  GetServerInfo(lIndex as Long, ByRef hServer as Long, ByRef bIsRunning as Boolean, ByRef bIsStale as Boolean, ByRef lTotalUsersOnline as Long, ByRef strMachineName as String, ByRef strProtocol as String, ByRef strEndpoint as String, ByRef Last as Date, ByRef First as Date)

  This function is used to retrieve various information about a specific server connected to the current Documanage Router. Note that ByRef properties are return values.

  **Parameters:**

- *1Index:* Valid range from 0 to value of the property TotalServers – 1.Specifies the current numbered server to retrieve information about. Typically you would loop through all the servers (from 0 to TotalServers – 1) changing this Index property of this call accordingly.
- *hServer:* A unique handle to the server. This identifier can be used in other server management type calls. This handle is displayed on the server console itself in the root of the tree.
- *bIsRunning, bIsStale:* Both these parameters indicate the status for the specified server. IsStale indicates that a server has been marked as out-of-date and will shutdown when all users are disconnected. IsRunning indicates that a server is up and running correctly.
- *lTotalUsersOnline:* Total number of users connected to this server.
- *strMachineName, strProtocol, strEndpoint:* All three of these indicate where the server is located on the network. Protocol and Endpoint may indicate only the primary/default network information since the server may be “listening” on multiple protocols.
- *Last, First:* Both of these date fields indicate the first and last times the server reported to the router.

**Return Value:** None, but note that ByRef properties are return values.

- ***SetServerStale***
  *SetServerStale (hServer as Long)*

  Marks a server as Stale. This Server will no longer accept new connections. This Server will also shut itself down when it no longer has any users connected to it.

  **Parameters:**

- *hServer:* The handle to the Documanage Server you want to mark as Stale. This value can be obtained with a call to GetServerInfo( ).

**Return Value:** None.

- ***SpawnNewServer***
  SpawnNewServer (hServer as Long)

  Launches another server on the same machine as the server specified by the hServer parameter.

  **Parameters:**

  - *hServer:* The handle of the Documanage Server that will launch another Documanage Server.

  **Return Value:** None.

- ***GetTotalNTUsers***
  ***GetNTUser (IIndex as Long) as String***
  ***GetTotalNTGroups as Long***
  ***GetNTGroup (IIndex as Long) as String***
  ***GetTotalNTUsersinGroup (strGroup as String) as Long***
  ***GetNTUserinGroup (strGroup as String, IIndex as Long) as String***
  ***GetTotalNTGroupsforUser (strUser as String) as Long***
  ***GetNTGroupsforUser (strUser as String, IIndex as Long) as String***
  ***GetFullName (strUser as String) as String***
  ***GetTotalUsersByTech ( pluginName as String) as Long***
  ***GetTotalGroupsByTech( pluginName as String) as Long***
  ***GetTotalUsersinGroupByTech (pluginName as String, strGroup as String) as Long***
  ***GetNumGroupsOfUserByTech (pluginName as String, strUser as String) as Long***
  ***GetTotalGroupsForUserByTech (pluginName as String, strUser as String) as Long***

***GetFullNameByTech (pluginName as String, strUser as String) as String***
***GetTotalMembersInGroupByTech (pluginName as String, strGroup as String) as Long***

These functions allow communication to the Documanage user and groups system. These functions are for internal Documanage use and should not be necessary for most end-users.

## Using the Session Control

1. **Set UserName, Password and Domain using a dialog box or other means.**
2. **Set the UserName, Password and Domain properties.**
3. **Call the method Connect(EZP_ACCESS_FULL,False). The hSession property will now be set.**
4. **On application close, call Connect(EZP_ACCESS_NONE, False).**

# POVolume Control

The POVolume Control returns a list of known volumes to the server. This control is used with Document Migration.

## Properties

- ***hSession*** **Data Type:** Long (Read/Write)
  The current server login from the Session control

- ***NumVolumes*** **Data Type:** Long (Read Only)
  The number of data types known to the system.

## Methods

- ***InitializeFromSession***
  hSession as Long

  Initializes the control and fetches the volume list

- ***GetVolume***
  Item (as Long) as String

  The list of volumes is cached in the control

## Using POVolume Control

```
POVolumeCtrl1.InitializeFromSession (POSession1.hSession)

lngVolumeCount = POVolumeCtrl1.NumVolumes

For i = 0 To lngVolumeCount - 1

  strResult = String(1024, " ")

  strResult = POVolumeCtrl1.GetVolume(i)

   Next
```

# POQuery Control

The Query control is used to open a cabinet based on the user's access rights and the specified filter or sort criteria. The Query control will return a unique hCursor handle, which identifies the Documanage cabinet opened. This hCursor is valid for the lifetime of the cabinet, until another Cabinet is opened with this same control, or a CloseCursor call is made. After either of those actions the hCursor should no longer be used.

## Properties

- ***Cabinet*** **Data Type:** String (Read/Write)
  The name of the Cabinet you wish to open. Cabinets are created in the Documanage Administrator Module. A full list of available cabinets can be obtained by using GetCabinetName( ).

- ***CurrentAuthor*** **Data Type:** String (Read Only)
  The author of the last item the method GetRelativeItem() returned.

- ***CurrentItemType*** **Data Type:** Long (Read Only)
  The item type of the last item the method GetRelativehItem() returned. Typically, this will be EZP_DOCUMENT, EZP_FOLDER, or EZP_CABINET.

- ***CurrentLabel*** **Data Type:** String (Read Only)
  The item label of the last item the method **GetRelativeItem()** returned.

- ***CurrentTypeDescription*** **Data Type:** String (Read Only)
  The human readable description returned of the item, e.g., "folder" or "invoice." of the last item the method **GetRelativehItem()** returned.

- ***DocumentFilter*** **Data Type:** String (Read/Write)
  A specially formatted string that will be used to filter what document(s)

will be shown in this opened cabinet. The format of the filter is a SQL Where Clause without the SQL keyword “WHERE”. To filter on a particular Documanage Document property you will need to know the corresponding database column name of that property (shown in the chart at the end of this document). The indexes (the columns of your mapped table) of the document’s parent folder can also be used. Detailed formatting of Where clauses can be found in any SQL reference. These values may need to be fully qualified. The document properties exist in the table OT_Docs and are typically owned by the user EZPOWER. The fully qualified name of OT_Docs can be retrieved by referencing the property DocumentTable.

Changing this value after Initializing will not affect the state of the opened cabinet. This property should be set before calling initialize.

- ***DocumentOrderBy*** **Data Type:** String (Read/Write)
  A specially formatted string used to change the order of documents shown in this Cabinet. This string’s format is identical to a SQL OrderBy clause without the SQL keyword “OrderBy”. To order on a particular Documanage Document property you will need to know the corresponding database column name of that property (shown in the chart at the end of this document). Detailed formatting of OrderBy clauses can be found in any SQL reference. These values may need to be fully qualified. They exist in the table OT_Docs and are typically owned by the user EZPOWER. The fully qualified name of OT_Docs can be retrieved by referencing the property DocumentTable.

  Changing this value after Initializing will not affect the state of the opened cabinet. This property should be set before calling initialize.

- ***DocumentTable*** **Data Type:** String (Read Only)
  The fully qualified SQL table name of the Documanage table that stores the documents for this cabinet. This is useful for creating the search string for the DocumentFilter property.

- ***FolderFilter*** **Data Type:** String (Read/Write)
  A specially formatted string that will be used to filter what folders will be shown in this opened cabinet. The format is a SQL Where Clause without the SQL keyword “WHERE”. Detailed formatting of Where clauses can be found in any SQL reference. In this statement you will refer to the columns of your business tables that make up this cabinet. Since these cabinet indexes differ from table to table and cabinet to cabinet. If in your application you have no prior knowledge of the format of the Cabinet, this information can be retrieved two ways:

  1) Use the functions GetTotalLevels and its related function GetLevelInfo to retrieve the names of each table that represents each level. Then by Initializing the POFolder control with Table Indexes Only. It will retrieve the columns (or what we call Indexes) of that table/level (cabinets can be multi-leveled, each level will have a different set of Indexes based on a single table). The POFolder’s array property Qualified is the name of the Index you would use in the folder filter string.

  2) If you already have the Cabinet opened you can Initialize Indexes Only in the POFolder control with any folder hItem. The use of the POFolder control is the same as the above approach. By navigating through each level of the cabinet you can get all the indexes. The first approach is a bit more generic and does not require an open cabinet containing folders.

  Changing this value after Initializing will not affect the state of the opened cabinet. This property should be set before calling initialize.

- ***FolderOrderBy*** **Data Type:** String (Read/Write)
  A specially formatted string used to change the order of folders shown in this Cabinet. This string’s format is identical to a SQL OrderBy clause without the SQL keyword “OrderBy”. Detailed formatting of OrderBy clauses can be found in any SQL reference. If, in your application, you have no prior knowledge of the format of the Cabinet, this information can be retrieved via the two ways detailed in the FolderFilter property above.

Changing this value after Initializing will not affect the state of the opened cabinet. This property should be set before calling initialize.

- ***hCurrentItem*** **Data Type:** Long (Read Only)
  The hItem of the last item the method GetRelativeItem() returned.

- **hCursor** **Data Type:** Long (Read Only)
  The unique value returned which identifies the cabinet opened. The hCursor will change after every Initialize() call. More information about hCursors can be found in the general description section of this control.

- ***hRootItem*** **Data Type:** Long (Read Only)
  hItem for the root folder of the cabinet. This hItem is the very top of the hierarchy of folders and documents that make up a cabinet. Visually this hItem is represented (by the POTree control for example) by a picture of a cabinet. This item is nether a folder or document, it is a special object in the system that will identify itself as a “EZP_CABINET”. While this item is an “EZP_CABINET” it is not an hCursor and should not be confused with one. Its basically a place holder in the tree hierarchy to make visual representation simpler and programmatic navigation easier. The following are some facts about the item useful for GetRelative( ) calls: There is always only one hItem of this type “EZP_CABINET” in any given Cabinet. It will never contain documents. It has no siblings or parent. Its children are all folders. If it has no children this cabinet must be empty.

- ***hSession*** **Data Type:** Long (Read/Write)
  The current session with the Documanage server. hSession is a value passed from the Session control, which identifies the login to the Documanage server.

- ***LastErrorNumber*** **Data Type:** Long (Read Only)
  The Documanage error number.

- ***LastErrorSource*** **Data Type:** String (Read Only)
  The location of the last Documanage error. This is useful for reporting bugs to Oracle.

- ***MaxFolders*** **Data Type:** Integer (Read/Write)
  Maximum number of top level folders to retrieve. This is typically a constant number around 50 to 100. For large cabinets retrieving all folders may use too many resources and be slow (a cabinet could, and often will, contain thousands of folders). See MinFolder on how to get folders passed this max number.

- ***MinFolders*** **Data Type:** Integer(Read/Write)
  This property's name is somewhat misleading. It represents the point inside the opened cabinet to start retrieving top level folders. When opening a cabinet you should always start with 0 (zero). Once a Cabinet is opened you will only be able to access a sub-set of folders that are contained in the cabinet. This set will be all the folders in the cabinet up to MaxFolders. To get the next set of folders you would set this property to itself plus MaxFolders. You then would need to Initialize this control again (and all dependent controls – see "Initializing and Re-initializing the POQuery" section below.) To get the next sets of folders, repeat this process, continually adding MaxFolders to this value and Re-initializing. To go to a previous set of folders simply subtract MaxFolder instead of adding.

- ***TotalCabinets*** **Data Type:** Long (Read Only)
  The total number of cabinets in the system. Use GetCabinetName() to retrieve the names of each cabinet.

- ***TotalDocumentTypeSelections*** **Data Type:** Long (Read Only)
  Number of valid document types defined. Use in conjunction with the method GetDocumentTypeName().

- ***TotalGroups*** **Data Type:** Long (Read Only)
  The total number of groups in the system. Use GetGroupID() to retrieve the total groups.

- ***TotalVariables*** **Data Type:** Long (Read Only)
  Total number of variables that must be set to open a cabinet. This should always be checked immediately after Initializing and before using the opened cabinet in any way. See the section “Using Variable Filters” below about how variable filters work.

- ***TotalVariablesPickListItems*** **Data Type:** Long Array (Read Only)
  The number of entries in the pick list for each variable used to open a cabinet. The input index is the same index of the variable name returned from the "VariableNames(index)" property below. The items in each pick list are returned by the GetVariablePickListItem method (see below). See the section “Using Variable Filters” below about how variable filters work. Its array size is based on TotalVariables (see Using Array Properties).

- ***VariableNames*** **Data Type:** String Array (Read Only)
  The name of each variable used to open a cabinet. See the section “Using Variable Filters” below about how variable filters work. Its array size is based on TotalVariables (see Using Array Properties).

- ***VariableTypes*** **Data Type:** Long Array (Read Only)
  The SQL type of each variable used to open a cabinet. Returns the SQL data type of a given identified folder property. Types include:

  EZP_TYPE_CHAR, EZP_TYPE_NUMERIC, EZP_TYPE_DECIMAL, EZP_TYPE_INTEGER, EZP_TYPE_SMALLINT, EZP_TYPE_FLOAT, EZP_TYPE_REAL, EZP_TYPE_DOUBLE, EZP_TYPE_VARCHAR, EZP_TYPE_DATE, EZP_TYPE_TIME, EZP_TYPE_TIMESTAMP, EZP_TYPE_LONGVARCHAR, EZP_TYPE_BINARY, EZP_TYPE_VARBINARY, EZP_TYPE_LONGVARBINARY

See the section “Using Variable Filters” below about how variable filters work. Its array size is based on TotalVariables (see Using Array Properties).

- ***VariableValues*** **Data Type:** String Array (Read/Write)
  The value of each of the variables used to open the cabinet. These should be set before using this cursor in any calls or controls. See the section “Using Variable Filters” below about how variable filters work. Its array size is based on TotalVariables (see Using Array Properties).

## Methods

- ***GetCabinetName***
  GetCabinetName(Index As Long) As String

  This method returns the cabinets that are accessible to this user. To retrieve all cabinets call this function multiple times incrementing Index from 0 to TotalCabinets –1.

  **Parameters:**

  - *Index:* Numbered cabinet to retrieve. Valid Range from 0 to TotalCabinets –1.

  **Return Value:** The name of the cabinet you requested.

- ***GetDocumentTypeName***
  GetDocumentTypeName(Index As Long) As String

  This method returns the name of all document types in the system. To retrieve all document types call this function multiple times incrementing Index from 0 to TotalDocumentTypeSelections –1.

  **Parameters:**

- *Index*: Numbered document types to retrieve. Valid Range from 0 to TotalDocumentTypeSelections –1.

**Return Value:** The name of the document type you requested.

- ***GetGroupID***
  *GetGroupID(Index As Long) As String*

  This method returns the group name.

- ***GetRelativehItem***
  GetRelativehItem(hItem As Long, RelativePosition As Long)As Long

  This method allows retrieving items in a cabinet.

  **Parameters:**

  - *hItem:* The item at which to begin the search.
  - *RelativePosition:* The geometric relation of the requested item to the specific item. This can be one of the following:

    EZP_GET_SELF

    EZP_GET_FIRST_CHILD

    EZP_GET_LAST_CHILD

    EZP_GET_NEXT_SIBLING

    EZP_GET_PREV_SIBLING

    EZP_GET_PARENT

See “Navigating With the GetRelativehItem()” below for more details.

**Return Value:** Returns the hItem you are requesting. Zero (0) indicates that this item was not found. In addition many properties will be set after this call:

- hCurrentItem
- CurrentItemDescription
- CurrentAuthor
- CurrentLabel
- CurrentItemType

These properties will change to the last hItem you retrieved in this way. They will reset to empty values if the item you requested doesn’t exist.

- ***GetTemporaryFile***
  GetTemporaryFile(Path As String, Prefix As String) As String

  This method creates and returns to you a temporary file in the specified directory. This function calls the Win32 API function of the same name and is exposed only as convenience to the programmer and has very little to do with Documanage functionality. The user is responsible for cleaning this file up.

  **Parameters:**

  - *HItem Path:* The full path on a locally accessible drive to create a temporary file.
  - *Prefix:* Three character prefix for the file.

  **Return Value:** If successful, the name of the new file.

- ***GetTemporaryPath***
  GetTemporaryPath()As String

  This method returns the local path where Documanage stores its temp files. This typically will be the location of the Windows temporary folder.

  **Return Value:** If successful the name of the temp path.

- ***GetVariablePickListItem***
  GetVariablePickListItem(VariableIndex As Long, PickListItemIndex As Integer) As String

  Returns the pick list string indicated by the PickListItemIndex for the Filter Variable indicated by the VariableIndex. The complete pick list for each Filter Variable may be retrieved by iterating over the range for PickListItemIndex from 0 to the size of the list (minus one) which is returned by the TotalVariablesPickListItems count for that VariableIndex. See the section “Using Variable Filters” below about how variable filters work.

  **Parameters:**

  - *VariableIndex:* Valid range from 0 to TotalVariables –1
  - *PickListItemIndex: :* Valid range from 0 to TotalVariablePicklistItems(VariableIndex) -1

  **Return Value:** String value to place in a pick list for this filter variable.

- ***Initialize()***
  This method returns an hCursor and hRootItem. Some properties should be set before, for example hSession, Cabinet, MinFolders and MaxFolders, are set. At this point, you have opened a cabinet. See the section “Using the Query Control” below.

- ***IsActionAllowed***
  IsActionAllowed (hSession As Long, hCursor As Long, hItem As Long, Action As Long) As Boolean

  This method returns a TRUE or FALSE value, depending if an operation is available and/or valid on a particular hItem. The values for operation are listed in the Documanage Include file. The valid operations are listed below in the section Using IsActionAllowed.

  **Parameters:**

  - *hSession*
  - *hCursor*
  - *hItem:* All three of these parameters will uniquely identify a folder or document in the system that will be the target of the security query. Since we pass a cursor and session we can check security on any item in the system, not just in the current cursor this control opened.
  - *Action:* The action code. See “Using IsActionAllowed” below. A different set of actions are used for folders and documents.

  **Return Value:** True if the action is allowed, false if not.

- ***ItemType***
  ItemType(hItem As Long) As Long

  This method returns the item type for any given hItem. [NOTE: This function is obsolete. The IsFolder, IsDocument, etc. should be used.]

  **Parameters:**

  - *hItem:* Handle to item we want to know the type of.

The type of the item: EZP_DOCUMENT, EZP_FOLDER, or EZP_CABINET.

- ***IsFolder(hItem as Long) as Boolean***
  ***IsDocument(hItem as Long) as Boolean***
  ***IsCabinet(hItem as Long) as Boolean***
  ***IsPersonalFolder(hItem as Long) as Boolean***
  ***IsProjectFolder(hItem as Long) as Boolean***
  Determines the type of an hItem. These sets of functions are useful in place of ItemType( ).

  **Parameters:**

  - *HItem:* The item you are checking

  **Return Value:** True if this item is the type you are checking for, otherwise False.

  - **PublishDocument (hItem as Long) as Long**
  Reserved for future use.

- ***GetTotalLevels***
  GetTotalLevels(strCabinet as String, bIncludeHiddenLevels as boolean) as Long

  Given a Cabinet this function will return the total number of levels it contains.

  **Parameters:**

  - *strCabinet:* Cabinet to retrieve the levels from.

  - *bIncludeHiddenLevels:* If False the method will return the count of visible levels. True will return all levels, visible or not.

**Return Value:** Number of levels in provided cabinet.

- ***GetLevelInfo***
  GetLevelInfo(strCabinet As String, lLevel As Long, IncludeHiddenLevels as Boolean, ByRef strTableName As String, ByRef strTableDescription As String, ByRef bAllowDocsAtTableLevel As Boolean, ByRef Boolean, ByRef strQualifiedName As String, ByRef strDatabaseName As String, ByRef bIsVisible as Boolean)

  Retrieves assorted information about a particular level of a particular cabinet. Note that ByRef properties are return values. This call is useful if information needs to be retrieved from the cabinet prior to opening the cabinet. Otherwise, cabinet information should be obtained through equivalent properties. This should be used in conjunction with the *InitializeTableIndexes* method of the POFolder object.

  **Parameters:**

  - *strCabinet:* The Cabinet in question.
  - *lLevel:* Level of the cabinet to retrieve information about.
  - *bIncludeHiddenLevels:* Indicates if the level number specified in lLevel is counting hidden levels. If the Total Level count was retrieved by calling GetTotalLevels with bIncludeHiddenLevel set to true, then this parameter should be set to true. If this level number is from POFolder's Level property or was determined by using GetRelativeItem and counting yourself then this value should be set to False. See the Documanage Administrator documentation for information about hidden levels.
  - *strTableName*
  - *strTableDescription*
  - *bAllowDocsAtTableLevel*
  - *bAllowDocsAtPCabLevel*

- *strQualifiedName*
- *strDatabaseName*
- *bIsVisible:* The different information returned by this function.

**Return Value:** None, other than the ByRef parameters.

- ***GetTotalRelatedTables***
  GetTotalRelatedTables(strTable As String) As Long

  This method retrieves the total count of related tables for a given table. These relationships are created in the Documanage Administrator.

  **Parameters:**

  - *strTable:* The name of the table to retrieve the number of related tables for.

  **Return Value:** The total count of related tables for the given table.

- ***GetRelatedTable***
  GetRelatedTable(lIndex As Long, strTable As String) As String

  This method retrieves a the name of a related table give an existing table and the related table's index which should not exceed the number returned by *GetTotalRelatedTables*.

  **Parameters:**

  - *lIndex:* Valid range from 0 to GetTotalRelatedTables(strTable) –1. Number of related table to retrieve.

  - *strTable:* The name of the table to retrieve related tables for.

  **Return Value:** Returns the name of a requested related table.

- ***GetTotalUsersInGroup***
  GetTotalUsersInGroup(strGroup As String) As Long

- ***GetUserInGroup***
  GetUserInGroup(lIndex as Long, strGroup As String ) As String

  *Both functions are obsolete.* Users and Groups can now be retrieved by the POSession control's NT users and groups calls.

- ***Reinitialize ()***
  This method can be used to refresh all folders in a cursor. It is not recommend to be used at this time. See "Initializing and Reinitializing The POQuery" below.

- ***CloseCursor ()***
  This method forces a cursor to be closed and free the associated memory on the Server. This call is automatically called when Visual Basic disposes of the control unloading its Form or calling initialize() on a control that has been initialized previously. It is always a good idea to make this call on any initialized Query controls prior to calling Session's Connect(EZP_ACCESS_NONE) (i.e. Disconnecting from the Server).

# Using the Query Control

## *Opening a Cabinet*

1. **Set the following properties: hSession, Cabinet, MinFolders, MaxFolders**

2. **Optionally, set the following properties: FolderFilter, DocumentFilter, FolderOrderBy, DocumentOrderBy**

3. **Call Initialize().**

4 **Check TotalVariables.**
If 0, proceed to step 5. Otherwise, traverse the list of Cabinet variables (1 through TotalVariables); the name is in VariableNames[*]. The value has to be set in VariableValues[*]. All variables **must** have been set to a valid value. Some variables may have a pick list, exposed through property TotalVariablesPickListItems and method GetVariablePickListItem(). See "Using Variable Filters" below.

5 **hCursor and hRootItem are now set.**

6 **Use RelativehItem() to navigate through the cabinet or use of a visual control such as POTree Control.**

7 **Call CloseCursor() when done.**

## Navigating the Cabinet Programmatically (Using GetRelativehItem)

While our toolkit provides visual controls to show and move through cabinets, code can be written to programmatically explore the cabinet and manipulate folder and documents.

GetRelativehItem can be used to move through the internal hierarchy of folders and documents and retrieve simple properties (labels and types). By using the return results (the hItem Handle) of this function you can initialize the folder and document controls to see extended properties such as folder indexes and document properties. Various operations can be performed on these folders and documents such as checking out documents and editing them, moving documents, deleting them, etc. All operations of Documanage are accessible without using the visual controls.

Sample Code Navigating the cabinet and adding item to a user create Listbox:

```
Dim CurrHItem As Long
```

```
' This code will add all folder's Labels in the
first level of a cabinet to a ListBox

' Navigate to the first Item in the cabinet -- The
hRootItem
   CurrHItem = MyQuery.GetRelative-
HItem(MyQuery.hRootItem, EZP_GET_SELF)

Get its first child which will always be folder
CurrHItem = MyQuery.GetRelativeHItem(MyQuery.hRoo-
tItem, EZP_GET_FIRST_CHILD)
' Add its label to the listbox
List1.AddItem MyQuery.CurrentLabel()

' Get all the rest of the folders...
While CurrHItem <> 0 ' Zero indicates there are no
folders at the requested Location
   CurrHItem = MyQuery.GetRelativeHItem(CurrHItem,
EZP_GET_NEXT_SIBLING)
   List1.AddItem MyQuery.CurrentLabel()
```

## Initializing and Reinitializing the POQuery

After you open a cabinet, you may wish to change the value of a property that must be set before you initialize (like FolderFilter, MinFolder, etc). To do this you must start over, and open the cabinet again by calling Initialize. By doing this, your original cabinet is closed and your hCursor and all hItems retrieved from the cabinet are now invalid. Any other controls using the hCursor and hItems (like the POTree, PODocument, POFolder, POFld_Doc, etc) must reset all their hCursor and hItem type properties.

## Using Variable Filters

It would be a good idea to review the Administrator documentation about Variable Filters before trying to write code to open one. To get started quickly with the Documanage controls, you can keep your application from opening any cabinets that have a Variable Filter. After Initializing the POQuery check the TotalVariables property. If it is not zero do not open the cabinet.

Opening a cabinet with a Variable Filter requires a bit more work than opening a cabinet without one. Basically, a “Variable Filter” is a lot like using FolderFilter, in that it limits what folders are shown when the cabinet is opened, but the Variable Filter string is set on the server side by the Administrator. The format of the string on the server is identical to a FolderFilter string with one difference: special "substitution tags" are written in the string. These values need to be substituted or replaced each time the cabinet is opened by the POQuery control. Basically you need to “fill-in the blanks” of the filter. Commonly used replacement values can be stored locally to your application in the local POFFICE.INI file to make it convenient to build a configurable pick list. These values can be retrieved via the GetVariablePicklistItem calls, however you are not required to make use of this feature in your application. The Documanage Workstationapplication does use this feature as an example.

Without specifying the values for the Variable Filters the cabinet will be opened ignoring the filter specified in the Administrator. This is useful for opening cabinets in conjunction with the PODocument’s source document filter property and Workflow.

### *Example*

Assume a cabinet called “Authors In a State” was created. It contains an index called “au_state” which in the actual database table contains the 2-letter abbreviation of the state. One Variable Filter was setup. One variable was specified (called “Author_State”.) When we open this cabinet we will see that TotalVariables equals 1. We could call VariableName(0) and it would return

"Author_State". We would need to set VariableValues(0) equal to a state, like "NY" to open the cabinet.

### *INI File Entries*

The POQuery control can supply pick list values read from the local POFFICE.INI file. These values are configured and stored locally on the client application machine, not on the server and so are machine-specific. The POFFICE.INI section [Cabinet Filters] is reserved for this purpose.

Each variable name can have a list of any number of strings associated with it to be used as pick list choices. The number of choices is given as a value of a key named after the variable name. Each choice is given as the value of a key named after the variable name concatenated with a zero-based decimal sequence number. In general, the section would look like this for two variables named "VariableAName" and "VariableBName":

```
[Cabinet Filters]
VariableAName=4
VariableAName0=first choice for VariableAName
VariableAName1=second choice for VariableAName
VariableAName2=third choice for VariableAName
VariableAName3=fourth choice for VariableAName
VariableBName=2
VariableBName0=first choice for VariableBName
VariableBName1=second choice for VariableBName
```

If there are more than 10 choices for a variable, the key values continue to 2 digits as in:

```
...
MyVariableName9=tenth choice for MyVariableName
MyVariableName10=eleventh choice for MyVariableName
… etc. ...
```

and so on. To implement choices for the previously discussed example cabinet "Authors in a State", one could set up the following section to offer

choices of commonly selected states. Recall that one variable was specified (called “Author_State”.)

```
[Cabinet Filters]
Author_State=5
Author_State0=DC
Author_State1=MD
Author_State2=NY
Author_State3=TN
Author_State4=VA
```

Finally, note that there is no mention of "cabinet name" in this discussion. If multiple cabinets are set up with Variable Filters, unless those filters refer to data with exactly the same choices, the filter variable names should be made distinct. This will allow the definition of distinct sets of choices using the scheme described here.

### *Sample Code*

Below is a Visual BASIC function that will setup and show a form to allow user input of these filters. The implementation of the form itself is left up to the reader. The form's appearance is of a series of labels and text boxes, one set for each variable to fill in. The form has 3 functions. First is Init, which tells the form how many values you want to display. DisplayValue will insert a row containing a label and a text box. Finally, GetValue retrieves what the user entered after the form was closed.

```
Public Function SetupVarFilterForm() As Boolean

   On Error GoTo ErrHld:

   Dim x As Long
   Dim y As Long
   Dim varCount As Long

   'Find out how many variables in this cabinet filter
   varCount = POQuery1.TotalVariables

   Load QueryForm
   Call QueryForm.Init(varCount)

   'If there are variables, set up form spaces to
   'fill-in
   If varCount > 0 Then
      QueryForm.Caption = "Please Fill-In All Values to
       Open This Cabinet..."

       For x = 0 To varCount - 1
         ‘DisplayForm takes a data type to assure the
         ‘data is entered in the correct form.
         Call QueryForm.DisplayValue(x,
            POQuery1.VariableNames(x),
            POQuery1.VariableTypes(x))

         'If a pick list is defined for this variable,
         'build it
         If POQuery1.TotalVariablesPickListItems(x) > 0
            Then

            For y = 0 To
            POQuery1.TotalVariablesPickListItems(x) - 1
               Call QueryForm.AddPickListItem(x,
                  POQuery1.GetVariablePickListItem(x, y))
```

```
                Next y

             End If

          Next x

          QueryForm.Show 1

          If QueryForm.tag = "0" Then
             'User hit cancel
             GoTo ErrHdl
          End If

          'Transfer user's input to POQuery control
          For x = 0 To varCount - 1
             POQuery1.VariableValues(x) =
                QueryForm.GetValue(x)
          Next x

       End If

       Unload QueryForm
       SetupVarFilterForm = True
       Exit Function

       ErrHld:
          Unload QueryForm
          SetupVarFilterForm = False
          Exit Function
    End Function
```

# Visual Controls

## POFolder and Document Control

This control is used mainly for displaying a single folder's contents. To use this control, you pass the hItem of the parent folder. The control then displays, in a Windows ListView Control, the contents (also known as children) of the opened folder. This control is contained in the POFld_Doc.ocx file.

### Properties

- ***AllowDeletingFolders*** **Data Type:** Boolean (Read/Write)
  Indicates if the delete key can be used to delete a highlighted folder. This operation will raise the Error event of this control if the user is not authorized to delete folders and this property is set to true.

- ***Appearance*** **Data Type:** Integer (Read/Write)
  0 — normal
  1 — 3Dlook

- ***BorderStyle*** **Data Type:** Integer (Read/Write)
  0 – normal
  1 – solid border

- ***CurrentHItem*** **Data Type:** Long Array (Read Only)
  Returns one of the selected **hItems** in the control indicated by Index. 0 to Total Selected –1.

- ***DisplayItems*** **Data Type:** Long (Read/Write)
  Indicates whether to display documents, folders, or both. For documents, set to EZP_DOCUMENT. For folders, set to EZP_FOLDER. For both, set to EZP_DOCUMENT logically “or-ed” with EZP_FOLDER.

- ***EmptyMessage*** **Data Type:** String (Read/Write)
  Text to display when no items are present in the view. Default is: " This Folder Is Empty"

- ***ExtendedStylesActive*** **Data Type:** Boolean (Read Only)
  Indicates if advanced ListView styles are available. Since this control is "sub-classed" from a standard Windows "common" control much of its appearance is determined by Window itself. If a recent comctrl32.dll is installed on your system certain new properties in this control will become active and this property will return true. Microsoft is shipping new comctrl32.dlls with it Internet Explorer web browser 3.0 or greater.

- ***FullRowSelect*** **Data Type:** Boolean (Read/Write)
  Allow a row to be selected by mouse-click anywhere in the row, not just in the first column. Available only if ExtendedStylesActive = TRUE.

- ***IndicateVersionedDocs*** **Data Type*:*** Boolean (Read/Write)
  Indicates versioned documents in a folder in bold face font if it is set to true. Default is true.

- ***LastErrorNumber*** **Data Type:** Long (Read Only)
  The last Documanage error.

- ***LastErrorSource*** **Data Type:** String (Read Only)
  The location of the last Documanage error. This is useful for reporting bugs to Oracle.

- ***ParentHCursor*** **Data Type:** Long (Read/Write)

Set to the hCursor generated by this control's master Query Control.

- ***ParentHItem*** **Data Type:** Long (Read/Write)
  The hItem of the folder whose contents you wish to show in the control.

- ***ParenthSession*** **Data Type:** Long (Read/Write)
  The hSession of the folder you wish to show in the control.

- ***TotalItems*** **Data Type:** Long (Read Only)
  The total number of items shown in the control.

- ***TotalSelected*** **Data Type:** Long (Read Only)
  The total number of selected items.

- ***CurrentFocusedhItem*** **Data Type:** Long (Read Only)
  The hItem of the element in the object in the listview that currently has the focus.

- ***ShowFolderProperties*** **Data Type:** Boolean (Read/Write)
  If set to TRUE allows folders indexes to appear in the control. This must be set prior to initialization. This setting is designed to allow this control to replace the Documanage Folder List control.

- ***ShowGridLines*** **Data Type:** Boolean (Read/Write)
  Show gridlines in the control. Available only if ExtendedStylesActive = TRUE.

- ***MovableHeaders*** **Data Type:** Boolean (Read/Write)
  Allows the position of the headers of the columns to be adjusted by mouse drag. Available only if ExtendedStylesActive = TRUE.

- ***ViewMode*** **Data Type:** Long (Read/Write)
  Determines how to display items in the list. It is recommended to use

LVS_REPORT in all cases.

Can be any of the following values:

LVS_SMALLICON

LVS_LIST

LVS_REPORT

## Methods

- ***Export***
  *Export(PutOnClipboard As Boolean, Path As String).*

  This method exports to a local path all the selected documents in the control.

  **Parameters:**

  - *PutOnClipboard:* Set to TRUE will additionally place these documents on the Clipboard in addition to the disk. Typically set this to FALSE.
  - *Path:* The pathname to export the files to.

- ***Copy()***
  This method copies a document(s) to the Windows Clipboard.

  **Return Value:** None

- ***Cut()***
  This method cuts a document(s) to the Windows Clipboard.

  **Return Value:** None

- ***Delete()***

This method deletes the currently selected document(s).

**Return Value:** None

- ***Paste()***
  This method pastes a document(s) from the Windows Clipboard into the selected folder.

- ***PasteShortcut()***
  This method pastes a document's shortcut from the Windows Clipboard into the selected folder.

- ***RefreshItem***
  *RefreshItem(hItem As Long, bNotify As Boolean, bKeepSelection As Boolean)*

  This method updates the contents of the folder in this control. Refreshing is necessary when an action in a different control effect the documents in this control. See Refreshing Folders and Documents below.

  **Parameters:**

  - *hItem:* The folder to refresh. If zero the control will refresh the currently selected hItem.
  - *bNotify:* Fire a Folder Action event (EZP_ACTION_REFRESHDEPENDENTS) in response to this call. Typically send FALSE. See the FolderAction event below.
  - *bKeepSelection:* Tells control to attempt to keep selected document highlighted after refresh.

- ***GetIndexFromHitem***
  *GetIndexFromhItem(Index As Long)*

  This method, given an hItem, returns its position in the list view.

**Parameters:**

- *hItem:* The handle to the item in the control.

**Return Value:** The index position in the list view.

- ***GetHtemfromIndex***
  *GethItemfromIndex(Index As Long) As Long*

  This method given an index in the list box returns an hItem.

  **Parameters:**

  - *Index:* Item's position in control. Valid range from 0 to TotalItems -1.

  **Return Value:** The hItem for the indicated item.

- ***GetLabel***
  ***GetLabel(hItem As Long)As String***
  This method returns the Documanage label using the hItem value (first column of display).

  **Return Value:** The Documanage label.

- ***Initialize()***
  This method is called to refresh the content of the control. Any changes to the document's properties must be followed by this call. hItem (set in ParenthItem property) is the folder's hItem that will have its contents shown.

- ***RemoveItem(hItem As Long) As Boolean***
  This method removes an hItem from list box given. It does not delete hItem from system – the item is only removed from the visual list, with no effect on the server. This method is useful for manual filtering and the hiding of documents. Refreshing or initializing this will eliminate this

methods effect.

**Parameters:**

**hItem:** The hItem handle for the item to hide.

- ***SetDisplayStyle***
  *SetDisplayStyle(Style As Long, bAdd As Boolean)*

  This method adds or removes Windows styles for this control. See the Windows API documentation on ListView's styles. This is provided for advanced users familiar with the ListView Window's common control (not the Visual Basic ActiveX version).

  **Parameters:**

  - *Style:* Any LVS_* styles. Multiple styles can be combined . These styles can be found in the Windows API documentation. Below are some examples:

    LVS_ICON, LVS_REPORT, LVS_SMALLICON, LVS_LIST, LVS_TYPEMASK, LVS_SINGLESEL, LVS_SHOWSELALWAYS, LVS_SORTASCENDING, LVS_SORTDESCENDING, LVS_SHAREIMAGELISTS, LVS_NOLABELWRAP, LVS_AUTOARRANGE, LVS_EDITLABELS, LVS_OWNERDATA, LVS_NOSCROLL, LVS_TYPESTYLEMASK, LVS_ALIGNTOP, LVS_ALIGNLEFT, LVS_ALIGNMASK, LVS_OWNERDRAWFIXED, LVS_NOCOLUMNHEADER, LVS_NOSORTHEADER

  - *bAdd:* Indicated if this style or combination of styles should be added or removed.

## Events

- ***Error***
  See the “Error Handling” section at the end of this document.

- ***Folder Action***
  *FolderAction(hItem As Long, Action As Long)*

  Fired when an action has occurred on a folder. The user can take the appropriate action at this point.

  **Parameters:**

  - *hItem:* The Item (a folder) which the specified action as taken place on. If 0 then the action took place on the control, but not on an item. This is called a “whitespace” event.
  - *Action:* The code of the action which took place. Can be one of the following:

    EZP_ACTION_EXPAND, EZP_ACTION_COLLAPSE

    A folder has been expanded or collapsed in some manner. If expanding you should typically have the control “drill down” into this folder by setting ParenthItem to the hItem parameter and calling initialize.

    EZP_ACTION_SELECT

    A folder has been selected in some manner. Setting the property CurrentHItem will not fire this event.

    EZP_ACTION_GETPROPERTIES

A folder has been selected to display a context menu or property in some manner. This is the right-click event.

EZP_ACTION_REFRESH_CUTSOURCE

Fired when a document has been cut from one control and pasted in this one.

EZP_ACTION_REFRESHDEPENDENTS

Fired when the control has refreshed the given folder internally or in response to a RefreshItem call with bNotify set to TRUE.

- ***FileDrop***
  *FileDrop(Filename As String, hItem As Long)*

  Fired when the user drops or pastes Windows' files from Windows' Explorer or anywhere else from Windows. You may take any action – like importing these files.

  **Parameters:**

  - *FileName:* The filename of the file dropped.
  - *hItem:* The folder hItem the file was dropped into.

- ***DocumentAction***
  *DocumentAction(hItem As Long, Action As Long)*

  Fired when an action has occurred on a document. The user can take the appropriate action at this point, which is probably viewing the document or showing its properties.

  **Parameters:**

- *hItem:* The Item (a document) which the specified action as taken place on.
- *Action:* The code of the action which took place. Can be one of the following:

  EZP_ACTION_EXPAND

  A documents has been selected to open. Typically the user would send this document to the Viewer control. Typically this is fired when a document is double-clicked or enter is pressed.

  EZP_ACTION_SELECT

  A document has been selected in some manner. Setting the property CurrentHItem will not fire this event.

  EZP_ACTION_GETPROPERTIES

  A document has been selected to display a context menu or property sheet. This is the right-click event.

## Example

The following is an example of the Document/Folder List control. In this example, the hSession handle is returned from the Session control, assigns the hCursor from the Query control, and then displays the appropriate folders and documents.

```
POFld_Doc1.ParenthSession   = POSession1.hSession

POFld_Doc1.ParentHCursor   = POQuery1.hCursor

POFld_Doc1.DisplayDetails   = True
```

## Using the POFolder and Document Control

This example shows how to display in the control all the folders in a cabinet. Let's assume we've already opened a cabinet in the Documanage Query Control (see that control for more details on opening cabinets). Remember that hRootItem (created by the Query Control) is the "File Cabinet" that contains all the folders. HRootItem is an hItem which is type EZP_CABINET , so since the Folder and Document Control shows the contents of any hItem (that is, type EZP_FOLDER or EZP_CABINET).

We can see the folders that this cabinet contains by doing the following:

```
POFld_Doc.ParentHSession = POSession.hSession

POFld_Doc.ParentHCursor  = POQuery.hCursor

POFld_Doc.ParentHItem = POQuery.hRootItem

POFld_Doc.Initialize
```

Now all of the cabinet's folders are displayed in the Control. We can see a folder's contents by double-clicking on that folder. Folders will typically contain documents and sub-folders. We then add some code in the control's FolderAction Event sub-routine.

```
Private Sub POFld_Doc1_FolderAction(ByVal hItem As Long, ByVal
Action As Long)
 Select Case Action

    case EZP_ACTION_EXPAND:
     rem Expand is a Double-Click
     POFld_Doc1.ParentHItem = hItem ‘ hItem is the Documanage Item
        ‘ Handle to the clicked folder.
           POFld_Doc1.Initialize
```

```
                case else:
            Exit Sub

        End Select
    End Sub
```

# POFolder List Control

This control works like the Folder and Documents List control except that it shows only folders (no documents). If the Folder and Document List control is set to show only folders, and to display full folder indexes, that control will appear the same as this one. Since the functionality of this control has been merged with Folder and Documents List control, the Folders List control should be considered obsolete and is only documented here briefly for any legacy code that may still use this control.

## Properties

- ***Appearance*** **Data Type:** Integer (Read/Write)
  0 — normal
  1 — 3Dlook

- ***BorderStyle*** **Data Type:** Integer (Read/Write)
  0 – normal
  1 – solid border

- ***ColumnCount*** **Data Type:** Long (Read Only)
  The total count of columns to be shown.

- ***FolderCount*** **Data Type:** Long (Read Only)
  The total number of folders shown.

- ***Enabled*** **Data Type:** Boolean (Read/Write)
  Whether or not the control is able to react to mouse clicks and Drag and Drop.

- ***Font*** **Data Type:** stdFont (Read/Write)
  The font used in the control. See your programming language reference for details on setting font properties. See Windows API documentation for the stdFont structure.

- ***hWind*** **Data Type:** Long (Read Only)
  The windows handle of the control. See Windows API for details.

- ***LastErrorNumber*** **Data Type:** Long (Read Only)
  The last error in Documanage.

- ***LastErrorSource*** **Data Type:** String (Read Only)
  The location of the last Documanage error. This is useful for reporting bugs to Oracle.

- ***ParentHItem*** **Data Type:** Long (Read/Write)
  The hItem of the folder whose contents are to be shown in the control.

- ***ParentHCursor*** **Data Type:** Long (Read/Write)
  The hCursor of the folder you wish to show in the control.

- ***ParentHSession*** **Data Type:** Long (Read/Write)
  The hSession of the folder you wish to show in the control.

- ***TotalSelected*** **Data Type:** Long (Read Only)
  The total number of selected items.

## Methods

- ***GetLabel***
  *GetLabel(hItem As Long) As String*

  This method returns the name of an item in the tree referred to by its hItem.

- ***Initialize()***

  This method opens a folder in the control. You determine which folder by the hCursor and hItem handles.

- ***Paste()***

  This method pastes a document(s) from the Windows Clipboard into the selected folder.

- ***RefreshItem***
  *RefreshItem(hItem As Long, bNotify As Long, bKeepSelection As Long)*

  This method is called to refresh the contents of the box. Any changes to the properties must be followed by this call. hItem (set in parenthItem property) is the folder's hItem that will have its contents shown.

## Events

- ***FolderAction***
  *FolderAction(hItem As Long, Action As Long)*

  Fired when a folder is expanded/opened, most likely when a user double-clicks the mouse or presses [Enter]). Whether the user expanded or collapsed a folder can be determined by checking the action parm. The user can take the appropriate action at this point. Specific actions that can be taken include:

EZP_ACTION_EXPAND, EZP_ACTION_SELECT

- ***FileDrop***
  *FileDrop(Filename As String, hItem As Long)*

  Fired when the user drops or pastes Windows' files from Windows' Explorer or anywhere from Windows. You may take any action, e.g., importing these files.

## Using the Folder List Control

As previously noted, this control operates nearly identically to the Folder and Documents control, except that it only shows folders with all their Documanage Folder Properties.

Much like the example in the Folder and Documents control we can show all the folders in an opened cabinet (see the Query Control to see just how to open a cabinet) this way:

```
POFld_List.ParentHSession = POSession.hSession

POFld_List.ParentHCursor  = POQuery.hCursor

POFld_List.ParentHItem = POQuery.hRootItem ‘ The Documanage Cabinet Handle and contains all the folders

POFld_List.Initialize
```

You can also let the user open one of the displayed folders (by a double-click) by the following code. Remember, however, that you will only see folders in this control - so by showing the contents of a folder here we won't see any documents, just sub-folders.

```
Private Sub POFld_List1_FolderAction(ByVal hItem As Long, ByVal
```

```
Action As Long)
    Select Case Action
        case EZP_ACTION_EXPAND:
        rem Expand is a Double-Click
        POFld_List1.ParentHItem = hItem ‘ hItem is the Documanage
        Item
            ‘ Handle to the clicked folder.
        POFld_List1.Initialize
        case else:
            Exit Sub
    End Select
End Sub
```

# POTree Control

The POTree control displays, in a hierarchical form, all the folders and documents in a given cabinet.

## Properties

- ***AllowDeletingFolders*** **Data Type:** Boolean (Read/Write)
  Indicates if the delete key can be used to delete a highlighted folder. This operation will raise the Error event of this control if the user is not authorized to delete folders and this property is set to true.

- ***Appearance*** **Data Type:** Integer (Read/Write)
  *0 — normal*

  *1 — 3Dlook*

- ***BorderStyle*** **Data Type:** Integer (Read/Write)
  *0 – normal*

*1 – solid border*

- ***CurrentFolderhItem*** **Data Type:** Long (Read Only)
  If a document is selected, the hItem of its parent folder will be returned. If a folder is selected, the hItem of that folder will be returned. This is useful when documents are shown to quickly determine the current folder.

- ***CurrentHItem*** **Data Type:** Long (Read/Write)
  The hItem of the selected folder, document, or cabinet. Changing this property will change the highlighted Item in the tree. More information about this property can be found below in Manipulating the Visual Controls programmatically.

- ***Enabled*** **Data Type:** Boolean (Read/Write)
  This property determines whether or not a control is able to react to mouse operation such as clicks and Drag and Drops.

- ***GroupBy*** **Data Type:** String (Read/Write)
  Groups documents in the tree by the specified document field. Valid fields are Category, SubCategory, Author, and an empty string indicating no groups. Default is an empty string.

- ***FolderCount*** **Data Type*:*** Integer (Read Only)
  Total number of visible top-level folders. If the HideEmptyFolders property is set to true, the sum of FolderCount and TotalHiddenFolders properties values will give the number of top level folders in the current result set.

- ***Font*** **Data Type:** stdFont (Read/Write)
  The font used in the control. See your programming language reference manual for details on setting font properties. NOTE: Please see Windows API documentation for the stdFont type declaration.

- ***hCursor*** **Data Type:** Long (Read/Write)
  The current hCursor of this control.

- ***HideEmptyFolders*** **Data Type*:*** Boolean (Read/Write)
  Hides folders that do not have any documents in them if it is set to true. Default is false.

- ***hRootItem*** **Data Type:** Long (Read/Write)
  The item to be used as the root of the tree. This should always be set to the value in the POQuery's hRootItem property.

- ***hSession*** **Data Type:** Long (Read/Write)
  The current session with the Documanage server. hSession is a value passed to you from the Session control that identifies your login to the Documanage server.

- ***hWnd*** **Data Type:** Long (Read Only)
  The windows handle of the control. See the Windows API for details.

- ***IndicateVersionedDocs*** **Data Type*:*** Boolean (Read/Write)
  Indicates versioned documents in a folder in bold face font if it is set to true. Default is true.

- ***LastErrorNumber*** **Data Type:** Long (Read Only)
  The last Documanage error.

- ***LastErrorSource*** **Data Type:** String (Read Only)
  The location of the last Documanage error. This is useful for reporting bugs to Oracle.

- ***ReadyState*** **Data Type:** Long (Read/Write)
  Not in use.

- ***ShowDocuments*** **Data Type:** Boolean (Read/Write)
  Set to TRUE to show documents in this control, FALSE to hide documents in this control. Use FALSE to get an Explorer-like look where only folders appear in the tree.

- ***Style*** **Data Type:** Long (Read/Write)
  Determines how the tree displays the hierarchy. In most cases you will want to set this value to 5. Buttons refer to the plus/minus buttons that appear next to each item in the tree. Lines are the gray lines connecting all the items, full lines will include lines at the root of the tree, simple lines will not.

  0 – simple (no lines or buttons)
  1 – buttons only
  2 – simple lines only
  3 – simple lines and buttons
  4 – full lines only
  5 – full lines and buttons

- ***TotalHiddenFolders*** **Data Type:** Long (Read Only)
  Total number of hidden folders when HideEmptyFolders property is set to true.

- ***TotalSelected*** **Data Type:** Long (Read Only)
  Total number of selected items in the tree. Since the tree is single selection, this will only return values of 1 or 0. [This property was added in the event that we extend the tree to be multi-select in future versions.]

## Methods

- ***Copy()***
  This method copies a document(s) to the Windows Clipboard.

**Return Value:** None

- ***Cut()***
  This method cuts a document(s) to the Windows Clipboard.

  **Return Value:** None

- ***Delete()***
  This method deletes the currently selected document(s).

  **Return Value:** None

- ***Expand***
  *Expand(hItem As Long, Action As Integer)*

  This method can be used to programmatically collapse and expand folders in the tree. Selecting items in the tree can be accomplished through setting the CurrentHItem property.

  **Parameters**:

  - *hItem:* The item to adjust. Documents cannot be expanded or collapsed in the tree since they never have children.
  - *Action:* One can specify whether they wish to expand, collapse, or toggle the state of a folder in the tree by setting the mode appropriately. Specific actions that can be taken include:

    EZP_ACTION_EXPAND,
    EZP_ACTION_COLLAPSE
    EZP_ACTION_TOGGLE

  **Return Value:** None

- ***Export***
  *Export(PutOnClipboard As Boolean, Path As String)*

  This method exports to a local path all the selected documents in the

control.

**Parameters**:

- *PutOnClipboard:* Set to TRUE will additionally place these documents on the Clipboard in addition to the disk. Typically set this to FALSE.
- *Path:* The pathname to export the files to.

- ***GetLabel***
  *GetLabel (hItem As Long) As String*

  This method returns the name of an item in the tree referred to by its hItem.

  **Parameters:**

  - *hItem:* The item in question.

  **Return Value:** The label asked for. If an empty is returned no item is selected.

- ***Initialize()***
  This method is used to display all the folders and documents in the control based on the settings in this control's properties. Before calling this function make sure the hSession, hCursor, and hRootItems have been set.

- ***Paste()***
  This method pastes a document(s) from the Windows Clipboard into the selected folder.

- ***PasteShortcut()***
  This method pastes a document's shortcut from the Windows Clipboard

into the selected folder.

- ***RefreshItem***
  *RefreshItem(hItem As Long, bNotify As Boolean, bKeepSelection As Boolean)*

  This method updates the contents of a given folder in the tree. Useful ONLY if the tree is showing documents. Refreshing is necessary when an action in a different control effect the documents in this control. See Refreshing Folders and Documents below.

  **Parameters:**

  - *hItem:* The folder to refresh. If zero the control will refresh the currently selected hItem.
  - *bNotify:* Fire a Folder Action event (EZP_ACTION_REFRESHDEPENDENTS) in response to this call. Typically send FALSE. See the FolderAction event below.
  - *bKeepSelection:* Tells control to attempt to keep selected document highlighted after refresh.

## Events

- ***ConfirmFileOperation***
  *ConfirmFileOperation (lOperation As Long, lItemCount As Long, By Ref pbAllowOperation As Boolean)*

  This is used to determine if a file operation is can be permitted. lOperation passes an EZP_ACTION… constant. The lItemCount indicates the number of files affected. pbAllowOperation should be set programmatically prior to exiting this event. This last parameter determines whether the file operation should complete or not.

- ***Error***

See the “Error Handling” section at the end of this document.

- ***FolderAction***
  *FolderAction (hItem As Long, Action As Long)*

  Fired when some action has occurred on a folder. The user can take the appropriate action at this point. This control will automatically change the display in the Folder Tree itself; there will also be no need to invoke, for example, the Expand() method yourself. Action on the Cabinet item are also included in this event.

  **Parameters:**

  - *HItem:* The Item (a folder) which the specified action as taken place on. If 0 then the action took place on the control, but not on an item. This is common called a “whitespace” event.
  - *Action:* The code of the action which took place. Can be one of the following:

    EZP_ACTION_EXPAND, EZP_ACTION_COLLAPSE

    A folder has been expanded or collapsed in some manner. The method expand will not fire this event.

    EZP_ACTION_SELECT

    A folder has been selected in some manner. The property CurrentHItem will not fire this event.

    EZP_ACTION_GETPROPERTIES

    A folder has been selected to display a context menu or property in some manner. This is the right-click event

    EZP_ACTION_REFRESH_CUTSOURCE

Fired when a document has been cut from one control and pasted in this one.

EZP_ACTION_REFRESHDEPENDENTS

Fired when the control has refreshed the given folder internally or in response to a RefreshItem call with bNotify set to TRUE.

- ***Group Action***
  *GroupAction(hItem As Long, Action As Long)*

  Fired when some action has occurred on a group. The user can take the appropriate action at this point, which is usually to disable user interface elements that would pertain to a selected folder or document.

  **Parameters**:

  - *hItem*: The Item (a group) which the specified action as taken place on (This is only a placeholder and is not a valid hItem).
  - Action: The code of the action which took place. Can be one of the following:

    EZP_ACTION_EXPAND

    A group has been selected to open. Typically no action is necessary as the control will expand or collapse the node. Typically this is fired when a document is double-clicked or enter is pressed.

    EZP_ACTION_SELECT

    A group has been selected in some manner. User interface that assumes a folder or document is selected should be disabled.

    EZP_ACTION_GETPROPERTIES

A group has been selected to display a context menu or property sheet. This is the right-click event.

- ***FileDrop***
  *FileDrop(Filename As String, hItem As Long)*

  Fired when the user drops or pastes Windows' files from Windows' Explorer or anywhere else from Windows. You may take any action – like importing these files.

  **Parameters:**

  - *FileName:* The filename of the file dropped.
  - *hitem:* The folder hItem the file was dropped into.

- ***DocumentAction***
  *DocumentAction(hItem As Long, Action As Long)*

  Fired when some action has occurred on a document. The user can take the appropriate action at this point, which is probably viewing the document or showing its properties.

  **Parameters:**

  - *hItem:* The Item (a document) which the specified action as taken place on.
  - *Action:* The code of the action which took place. Can be one of the following:

    EZP_ACTION_EXPAND

    A document has been selected to open. Typically the user would send this document to the Viewer control. Typically this is fired when a document is double-clicked or enter is pressed.

EZP_ACTION_SELECT

A document has been selected in some manner. The property CurrentHItem will not fire this event.

EZP_ACTION_GETPROPERTIES

A document has been selected to display a context menu or property sheet. This is the right-click event.

## Example

```
POTree1.hSession        = POSession1.hSession

POTree1.hCursor         = POQuery1.hCursor

POTree1.hRootItem       = POQuery1.hRootItem

POTree1.Initialize()
```

## Using the Tree Control

1. **Obtain a valid hSession (from Session control), valid hCursor and hRootItem (from Query control).**
2. **Set the hSession, hCursor and hRootItem properties of the Tree control.**
3. **Optionally, set the ShowDocuments and Style properties.**
4. **Call Initialize().**
   This will display the Tree.

5 **Optionally, call GetRelativeHItem() in the Query control and the tree's Expand() in any desired combination to open the tree to the desired configuration.**

# DmgViewer Control

The DmgViewer control replaces the POViewer control, described on page 158; however it is not backward compatible with that control in any way. The DmgViewer control supports the server-side rendering viewing technology in the context of Documanage. It is a user control that can be drawn on a dialog. Any number of DmgViewer controls can coexist within the same application (or on the same dialog). The DmgViewer is fully functional within the control itself; there is no need for the container to implement UI to drive the control. However, the control presents interfaces and events that allow it to be driven by an external UI. This UI can get quite complex so developers are encouraged to enable the control's toolbars and let it handle the complexity. The container is responsible for initializing the control and for directing the control to save any altered annotations when the DmgViewer is to be reinitialized or cleared.

## DmgViewer interface

### Events

- ***SetZoomUI(' SET ZOOM INTERFACE***
  *ByVal iZoom As ZOOM_OPTION,' option - (see ZOOM_OPTION enum)*
  *ByVal lValue As Long)' value for certain options*

  If the container has a user interface to indicate the current scaling level of the image, it can respond to this event and set the interface accordingly.

- ***SetRotationUI(' SET IMAGE ROTATION INTERFACE***
  *ByVal iDegree As Integer)' current image rotate degree*

  If the container has a user interface to indicate the current rotation of the image, it can respond to this event and set the interface accordingly.

- ***SetModeUI(' SET MODE INTERFACE***
  *ByVal lMode As VW_MODE)' current mode (see VW_MODE enum)*

  If the container has a user interface to indicate the current viewing mode of the control (grabber mode, marquee mode, or annotation mode), it can respond to this event and set the interface accordingly.

- ***SetEditMenuUI(' SET EDIT MENU INTERFACE***
  *ByVal lMode As VW_MODE,' current mode (see VW_MODE enum)*
  *ByVal bIsRect As Boolean,' selection rect drawn (.t. | .f.)*
  *ByVal bClipboardData As Boolean, ' clipboard data present*
  *ByVal nAnnotations As Long)' current annotation count*

  If the container has an Edit Menu (Cut, Copy, Paste, Delete, Select All), it can respond to this event and set the various Edit Menu options based upon the parameters of this event).

- ***SetImageToolsUI(' SET IMAGE TOOLS INTERFACE***
  *ByVal bEnabled As Boolean)' controls should be enabled | disabled*

  If the container has an Image Tools menu or user interface, it can respond to this event to enable or disable the interface.

- ***SetTaskUI( ' SET TASK COUNT INTERFACE***
  *ByVal iTask As Long, ' current taskDoc number*
  *ByVal nTasks As Long)' current taskDoc count*

  If the container has a user interface to indicate the current task document number and count, it can respond to this event and set the interface accordingly.

- ***SetImageUI(' SET IMAGE COUNT INTERFACE***
  *ByVal iImage As Long,' current image number*
  *ByVal nImages As Long)' current image count*

  If the container has a user interface to indicate the current image number and count, it can respond to this event and set the interface accordingly.

- ***SetAnnotationUI(' SET ANNOTATION INTERFACE***
  *ByVal lOption As ANO_UI_OPTION, ' option (see ANO_UI_OPTION enum) ByVal lValue As Long)' value for certain options*

  If the container has a menu or interface indicating the annotation options, it can respond to this event and set the options accordingly.

- ***SetImageDirtyUI(' SET IMAGE ANNOTATION CHANGED INTERFACE***
  *ByVal bImageDirty As Boolean, ' current image annotations have changed*
  *ByVal bFileDirty As Boolean)' current file has changed annotations*

  If the container has a user interface that indicates that the current image and/or file annotations have changed, it can respond to this event and set the interface accordingly.

- ***ShowContextMenuUI(' PRESENT A CONTEXT MENU***
  *ByVal lOption As CONTEXT_MENU)' option (see CONTEXT_MENU enum)*

  If the container would like to present a context menu when the user right-clicks on the image, it can respond to this event and present a context menu.

- ***Error('CONTROL ACTION HAS GENERATED AN ERROR***
  *ByVal Number As Integer,' error number - usually EZP_STD_ERROR*
  *ByVal Description As String,' error description*
  *ByVal Scode As Long,' [ Not Implemented ]*
  *ByVal Source As String,' source of last error*

*ByVal HelpFile As String,' [ Not Implemented ]*
*ByVal HelpContext As Long,' [ Not implemented ]*
*ByRef CancelDisplay As Boolean) ' set to true to suppress error dialog*

This event is generated when an error is encountered by the DmgViewer control.

## Properties

### *Appearance Properties*

The following properties control the visibility and appearance of various interface elements of the DmgViewer control.

- ***AnnotationToolbarVisible('ANNOTATION TOOLBAR IS VISIBLE*** *) As Boolean' [ READ WRITE ]*

  The annotation toolbar can be visible or not. If it is visible, the container does not need to support the various annotation commands and events.

- ***AnnotationToolbarWrappable('ANNOTATION TOOLBAR WRAPS*** *) As Boolean' [ READ WRITE ]*

  This determines whether the annotation toolbar will wrap to multiple lines as the size of the DmgViewer control is changed. If this is set to false, the toolbar will truncate if the DmgViewer control is sized smaller than the toolbar length. If this is set to true, the toolbar will wrap to multiple lines if the DmgViewer control is sized smaller than the toolbar length.

- ***ImageToolbarVisible(' IMAGE TOOLBAR IS VISIBLE*** *) As Boolean' [ READ WRITE ]*

  The image toolbar can be visible or not. If it is visible, the container does not need to support the various image commands and events.

- ***ImageToolbarWrappable(' IMAGE TOOLBAR WRAPS***
  *) As Boolean' [ READ WRITE ]*

  This determines whether the image toolbar will wrap to multiple lines as the size of the DmgViewer control is changed. If this is set to false, the toolbar will truncate if the DmgViewer control is sized smaller than the toolbar length. If this is set to true, the toolbar will wrap to multiple lines if the DmgViewer control is sized smaller than the toolbar length.

- ***StatusBarVisible(' STATUSBAR IS VISIBLE***
  *) As Boolean' [ READ WRITE ]*

  The status can be visible or not. If it is visible, the container does not need to support the various image status events.

***Printing Options***

- **IgnoreDCDprintTags**(' PRINT DCD DOCUMENTS AS DISPLAYED
  *) As Boolean' [ READ WRITE ]*

  Documaker DCD file types can be printed on background stock paper. This option can be set to display the image as if it were printed on the stock paper.

- **PrintAnnotations**(' PRINT ANNOTATIONS WITH IMAGE
  *) As Boolean' [ READ WRITE ]*

  If this property is true, the document annotations will be printed with the document.

- **PrintOutputFormat**(' PRINT OUTPUT FORMAT
  *) As POFORMAT' [ READ WRITE ]*

  This option determines how the image will be printed (Actual size, fit to page, or pixel to pixel).

***Image Rendering and Scaling Options***

- ***EnableAFPMETcolorImage(' RENDER AFP & MET COLOR IMAGES***
  *) As Boolean' [ READ WRITE ]*

  This option determines whether AFP and Metacode printstreams that contain color will be rendered in color or black and white.

***Current Image and Document Properties***

- ***hSession(' DOCUMENT hSESSION***
  *) As Long' [ READ WRITE ]*

  The Documanage session handle for the current document. This property must be set prior to calling the Initialize method.

- ***hCursor(' DOCUMENT hCURSOR***
  *) As Long' [ READ WRITE ]*

  The Documanage cursor handle for the current document. This property must be set prior to calling the Initialize method.

- ***hItem(' DOCUMENT hITEM***
  *) As Long' [ READ WRITE ]*

The Documanage document handle for the current document. This property must be set prior to calling the Initialize method.

- ***DocumentName(' DOCUMENT NAME***
  *) As String' [ READ WRITE ]*

  The name of the current document. This property should be set prior to calling the Initialize method but it can be blank.

- ***AnnotationsAllowed(' USER CAN ANNOTATE IMAGE***
  *) As Boolean' [ READ ONLY ]*

  Indicates if the current user has authority to annotation the document.

- ***DirtyFile(' FILE ANNOTIONS HAVE CHANGED***
  *) As Boolean' [ READ WRITE ]*

  If the annotations on any page in the current document have been altered, this value will be true.

- ***DirtyImage('CURRENT IMAGE ANNOTATIONS HAVE CHANGED***
  *) As Boolean' [ READ WRITE ]*

  This value will be true if the annotations for the current image have been altered.

- ***iImage(' CURRENT IMAGE NUMBER***
  *) As Long' [ READ ONLY ]*

  The image number currently showing in the DmgViewer control (1 based).

- ***nImages(' CURRENT IMAGE COUNT***
  *As Long' [ READ ONLY ]*

  The number of images in the current task document. NOTE that virtually all Documanage files only have one task document.

- ***iTask(' CURRENT TASKDOC NUMBER***
  *) As Long [ READ ONLY ]*

  The current task document set for the DmgViewer control. NOTE that virtually all Documanage files only have one task document.

- ***nTasks(' CURRENT TASKDOC COUNT***
  *) As Long' [ READ ONLY ]*

  The number of task documents in the current file. NOTE that virtually all Documanage files only have one task document.

***Error Information***

- ***LastErrorAdviceID(' ADVICE CLAUSE ID [ Not Implemented ]***
  *) As Long' [ READ ONLY ]*

  This property is reserved for future use. It will always have a value of zero.

- ***LastErrorDescription(' LAST ERROR DESCRIPTION***
  *) As String' [ READ ONLY ]*

  This property will contain a description of the last error encountered by the DmgViewer control.

- ***LastErrorNumber(' LAST ERROR NUMBER***
  *) As Long' [ READ ONLY ]*

  This property will contain the number of the last error encountered by the DmgViewer control.

- ***LastErrorSource(' SOURCE OF LAST ERROR***
  *) As String' [ READ ONLY ]*

  This property will contain some information about where the last error occurred in the DmgViewer control.

# DmgViewer Methods

## Document Operations

- ***Initialize(' INITALIZE CONTROL (and show 1st image)***
  *) As DMV_ERROR (DMV_NOERROR | DMV_GENERALERROR | DMV_NOTIMAGEFILE)*

  This method initializes the control and show the first image of the first task document.

- ***Clear() ' CLEAR CONTROL***

  This clears the control and disables all of the control toolbars. If the container has implemented user interface elements to control the DmgViewer control, it should respond to the various events raised by this call and disable its user interface.

- ***ShowImage(' SHOW SELECTED IMAGE***
  *ByVal iTask As Long,' taskDoc number (1 based)*
  *ByVal iImage As Long' image number (1 based)*
  *) As DMV_ERROR (DMV_NOERROR | DMV_GENERALERROR)*
  *Show the requested image.*
  *SaveAnnotations()' SAVE ANNOTATION DATA TO Documanage*

  Save any altered annotations to permanent storage in Documanage. Any individual page annotations that have changed are saved automatically by the DmgViewer control. This call commits the changes to permanent storage. The container should check the .DirtyImage property prior to making this call. NOTE that the DmgViewer control will not automatically save the file annotations to Documanage. It is up to the container to make this call.

- ***PrintDocument() ' PRINT DOCUMENT***

  Print the current document. The user will be prompted with a Printer Setup dialog.

***Edit Operations***

- ***EditCopy() ' COPY SELECTED ITEMS TO CLIPBOARD***
  This method copies the currently selected annotations or image marquee to the clipboard. If the Copy menu option from the DmgViewer Image Toolbar is selected, the control does this operation automatically. If the container has an Edit Menu with a Copy option, it can invoke this method in response to that menu selection.

- ***EditCut() ' CUT SELECTED ITEMS TO CLIPBOARD***
  This method cuts the currently selected annotations to the clipboard. If the Cut menu option from the DmgViewer Image Toolbar is selected, the control does this operation automatically. If the container has an Edit Menu with a Cut option, it can invoke this method in response to that menu selection. NOTE that an image selection marquee cannot be cut.

- ***EditClear() ' DELETE SELECTED ITEMS***
  This method deletes the currently selected annotations. If the Delete menu option from the DmgViewer Image Toolbar is selected, the control does this operation automatically. If the container has an Edit Menu with a Clear or Delete option, it can invoke this method in response to that menu selection. NOTE that an image selection marquee cannot be deleted.

- ***EditPaste() ' PASTE CLIPBOARD CONTENTS***
  This method pastes any annotations in the clipboard onto the image. If the Paste menu option from the DmgViewer Image Toolbar is selected, the control does this operation automatically. If the container has an Edit Menu with a Paste option, it can invoke this method in response to that menu selection. NOTE that an image data cannot be pasted.

- ***EditSelectAll() ' SELECT ALL AVAILABLE ITEMS***
  This method selects all of the current annotations in the DmgViewer is in annotation mode or marquees the entire image if the DmgViewer is in

marquee mode. If the Select All menu option from the DmgViewer Image Toolbar is selected, the control does this operation automatically. If the container has an Edit Menu with a Select All option, it can invoke this method in response to that menu selection.

***User Interface Operations***

- ***ShowIndexInfo() ' SHOW PRINTSTREAM INDEX INFORMATION***
  This method is only relevant for printstream files. It will show a dialog containing the Index Information embedded in the printstream file. If the Index Information button is selected from DmgViewer Image Toolbar, the control does this operation automatically. If the container has an Menu with an Index Information option, it can invoke this method in response to that menu selection.

- ***ShowResourceInfo() ' SHOW PRINTER RESOURCE INFORMATION***
  This method is only relevant for printstream files. It will show a dialog containing the Printer Resources required to render the printstream file. If the Resource Information button is selected from DmgViewer Image Toolbar, the control does this operation automatically. If the container has an Menu with an Resource Information option, it can invoke this method in response to that menu selection.

- ***ShowAnnotations( ' SHOW / HIDE ANNOTATIONS***
  *ByVal bShow As Boolean)' show | hide*

  This option shows or hides any image annotations. If the Show/Hide menu option from the DmgViewer Image Toolbar is selected, the control does this operation automatically. If the container has an Annotation Menu with a Show/Hide option, it can invoke this method in response to that menu selection.

- ***SelectAnnotationTool( ' SELECT ANNOTATION TOOL***
  *iTool As AnnotationToolIDs)' ToolID*

  This option selects the specified annotation tool (see AnnotationToolIDs). If the tool is selected from the DmgViewer Annotation Toolbar, the control does this operation automatically. If the container has an Annotation Menu or Toolbar, it can invoke this method in response to the tool selection.

- ***SetMode( ' SET VIEW MODE***
  *ByVal lMode As VW_MODE)' mode*

  This option sets the current view mode for the DmgViewer. The modes can be Grabber mode, Marquee mode, or Annotation mode. If the mode is selected from the DmgViewer Image Toolbar, the control does this operation automatically. If the container has a Mode menu or Toolbar, it can invoke this method in response to the mode selection.

- ***SetRotation(' SET IMAGE ROTATION***
  *ByVal lOption As ROTATION_OPTION) ' rotation option*

  This method sets the current image rotation (see the ROTATION_OPTION enumeration). If the rotation is selected from the DmgViewer Image Toolbar, the control does this operation automatically. If the container has a Rotation menu or Toolbar, it can invoke this method in response to the rotation selection.

- ***SetZoom( ' SET IMAGE SCALE***
  *ByVal lOption As ZOOM_OPTION)' scale option*

  This method sets the current image scale (see the ZOOM_OPTION enumeration). If the scale is selected from the Scale menu on the DmgViewer Image Toolbar, the control does this operation automatically. If the container has a Scale Menu, it can invoke this method in response to the scale selection.

## Enumerations

```
Public Enum VW_MODE
  VW_MODE_EMPTY = 0
  VW_MODE_MARQUEE = 1
  VW_MODE_GRABBER = 2
  VW_MODE_ANNOTE8 = 3
End Enum
Public Enum ANO_UI_OPTION
  ANO_UI_OPTION_DISABLE = -1
  ANO_UI_OPTION_ENABLE = 0
  ANO_UI_OPTION_UNCHECK = 1
  ANO_UI_OPTION_TOOLSELECT = 2
  ANO_UI_OPTION_SHOWHIDE = 3
End Enum
Public Enum ROTATION_OPTION
  ROTATE_NONE = 0
  ROTATE_LEFT90 = 1
  ROTATE_RIGHT90 = 2
  ROTATE_FLIP = 3
End Enum
Public Enum ZOOM_OPTION
  ZOOM_ENABLE = -1
  ZOOM_FITTO_WINDOW = 0
  ZOOM_FITTO_WIDTH = 1
  ZOOM_FITTO_HEIGHT = 2
  ZOOM_FITTO_ACTUAL = 3
  ZOOM_SCALE_1600 = 4
  ZOOM_SCALE_800 = 5
  ZOOM_SCALE_400 = 6
  ZOOM_SCALE_200 = 7
```

```
    ZOOM_SCALE_150 = 8
    ZOOM_SCALE_125 = 9
    ZOOM_SCALE_100 = 10
    ZOOM_SCALE_75 = 11
    ZOOM_SCALE_50 = 12
    ZOOM_SCALE_25 = 13
    ZOOM_SCALE_12 = 14
    ZOOM_SCALE_6 = 15
    ZOOM_MARQUEE_DRAWN = 16
    ZOOM_MARQUEE_UP = 17
    ZOOM_MARQUEE_DOWN = 18
  End Enum
  Public Enum CONTEXT_MENU
    MNU_NAVIGATION = 0
    MNU_MODE = 1
    MNU_ZOOM = 2
    MNU_ORIENTATION = 3
    MNU_PRINTPROGRESS = 4
  End Enum
  Public Enum DMV_ERROR
    DMV_GENERALERROR = -1
    DMV_NOERROR = 0
    DMV_USERCANCEL = 1
    DMV_WORKING = 2
    DMV_WARNING = 3
    DMV_NOTIMAGEFILE = 7
  End Enum
```

***POFormat Values***
pofActualSize = 1
pofFitToPage = 2
pofPixelToPixel = 0

***AnnotationToolIDs values***
TOOLID_ATTACH_NOTE = 8
TOOLID_END = 11
TOOLID_FILLED_RECT = 6
TOOLID_FREEHAND_LINE = 2
TOOLID_HIGHLIGHTER = 3
TOOLID_HOLLOW_RECT = 5
TOOLID_NONE = 0
TOOLID_RUBBER_STAMP = 10
TOOLID_SELECT_ANNOTATIONS = 1
TOOLID_STRAIGHT_LINE = 4
TOOLID_TEXT = 7
TOOLID_TEXT_FROM_FILE = 9

# Using the DmgViewer control

## Setup

```
With ocxDmgViewer
  .AnnotationToolbarVisible = True
  .AnnotationToolbarWrappable = True
  .ImageToolbarVisible = True
  .ImageToolbarWrappable = True
  .StatusBarVisible = True
End With
```

### Initializing

```
Private Sub loadViewer(ByVal hItem As Long, ByVal sTag As String)
  '
  ' clear control (this optionally saves any changed annotations)
  '
  unloadViewer
  '
  ' initialize / reinitialize viewer
  '
  With ocxDmgViewer
    .hSession = m_hSession
    .hCursor = m_hCursor
    .hItem = hItem
    .DocumentName = sTag
    dmvRC = .Initialize()
    If (dmvRC <> DMV_NOERROR) Then
      m_TKCorigin = .LastErrorSource
      m_TKCerror = .LastErrorDescription
      m_TKCadvice = "Please check with your System Administrator."
      iOK = showMsg(vbInformation, m_TKCorigin, m_TKCerror,
m_TKCadvice)
    End If
  End With
End Sub
```

### Saving / Clearing

```
Private Sub unloadViewer()
  '
  ' offer to save any altered annotations, then clear control
  '
```

```
  With ocxDmgViewer
    If .DirtyFile Then
      m_TKCerror = "The document annotations have been altered."
      m_TKCadvice = "Do you want to save the annotation data?"
      iOK = showMsg((vbYesNo + vbQuestion + vbDefaultButton1), "Save
Annotations", m_TKCerror, m_TKCadvice)
      If (iOK = vbNo) Then
      Else
        .SaveAnnotations
      End If
    End If
    .Clear
  End With
End Sub
```

# POViewer Control

The POViewer control has been deprecated in favor of the DmgViewer control, described on .page 142. The POViewer control displays a document in the Documanage viewer. This sends the hSession, hCursor, and hItem to identify the document to be viewed, and calls Initialize() and finally View(). Annotations to the document can also be performed through the viewer using the Annotation properties and method. Note that viewing annotations is automatic when any image is viewed.

While POViewer may seem the most complex of the controls, it provides a lot of functionality with only a minimum amount of code. If creating/editing

annotations are not required, then the control is very straightforward. The properties and methods are divided into two categories:

- Imaging/Basic which detail the viewing of documents, and
- Annotation which allows you to create and edit multi-layer annotations.

## Basic Imaging Properties

- ***hCursor*** **Data Type:** Long (Read Only)
  The cursor of the cabinet in which the document being viewed exists.
- ***hItem*** **Data Type:** Long (Read Only)
  The handle of the document being viewed.
- ***hSession*** **Data Type:** Long (Read Only)
  The handle of the session obtained originally from POSession.
- ***hWnd*** **Data Type:** Long (Read Only)
  The window handle.
- ***LastError Number*** **Data Type:** Long (Read Only)
  The last Documanage error.
- ***LastErrorSource*** **Data Type:** String (Read Only)
  The location of the last Documanage error. This is useful for reporting bugs to Oracle.
- ***Rotation*** **Data Type:** Short (Read/Write)
  The current angle of rotation at which the image is displayed: 0, 90, 180, or 270.

## Basic Imaging Methods

- ***FillWindow()***
  This method fills the entire window with the image; the part of image appears in window.

- ***FitInWindow()***
  This method makes the entire image fit in the window, leaving portions of the window blank.

- ***NextPage()***
  This method gets the next page in a multi-page document.

- ***PreviousPage()***
  This method gets the previous page in a multi-page document.

- ***PrintDocument***
  ***PrintDocument(ShowDialog As Boolean) As Boolean***
  This method prints the currently viewed document. Also see SetPrinterInfo().

  **Parameters:**

  - *ShowDialog:* Whether or not to show the Windows Print dialog. Typically this should be TRUE.

- ***View***
  *View() As Long*

  This method views the document indicated in the hItem property.

  **Return Value:** Should be ignored.

- ***ViewNormalSize***
  *ViewNormalSize() As Long*

  This method removes the current zooming.

- ***ZoomIn()***
  This method zooms into a page.

- ***ZoomOut()***
  This method zooms out of a page.

- ***Initialize() As Long***
  Call before View, allows internal initialization and clean-up prior to viewing.

- ***SetPrinterInfo***
  *SetPrinterInfo(sDriverName as string, sDeviceName as string, sPortName as string) As Long*

  Sets Printer driver information if you wish to print with no dialog. This call is not needed when ShowDialog is True in the Print call.

  **Parameters:**

  - sDriverName
  - sDeviceName
  - *sPortName:* Various information to identify your printer.

- ***CanRotate() As Boolean***
  This method returns true if the image can be rotated, false otherwise.

## Annotation Properties

- ***AnnotationBGColor*** **Data Type:** OLE Color (Read/Write)
  The background color for the highlighted arrow.

- ***AnnotationCurrentLayer*** **Data Type:** Long (Read/Write)
  The current layer of an annotation.

- ***AnnotationFGColor*** **Data Type:** OLE Color (Read/Write)
  The foreground color for the highlighted arrow.

- ***AnnotationLayers*** **Data Type:** Long (Read/Write)
  The number of layers for the current document.

- ***AnnotationLineStyle*** **Data Type:** Integer (Read/Write)
  The style of the annotation line.

- ***AnnotationLineWidth*** **Data Type:** Integer (Read/Write)
  The width of an annotation line.

- ***AnnotationMode*** **Data Type:** Long (Read/Write)
  The current annotation mode:

  EZP_ANNOTATION_HIDE, EZP_ANNOTATION_EDIT, or EZP_ANNOTATION_SHOW.

- ***AnnotationObject*** **Data Type:** Integer (Read/Write)
  The current annotation object can include:

  EZP_ANNOTATION_LINEOBJ, EZP_ANNOTATION_TEXTOBJ, EZP_ANNOTATION_ARROWOBJ, EZP_ANNOTATION_RECTANGLEOBJ, EZP_ANNOTATION_ELLIPSEOBJ,

EZP_ANNOTATION_STICKYNOTEOBJ,
EZP_ANNOTATION_POLYGONOBJ,
EZP_ANNOTATION_POLYLINEOBJ,
EZP_ANNOTATION_HIGHLIGHTEROBJ,
EZP_ANNOTATION_STAMPOBJ,
EZP_ANNOTATION_REDACTIONOBJ,
EZP_ANNOTATION_FREEHANDOBJ,
EZP_ANNOTATION_SOUNDOBJ,
EZP_ANNOTATION_BUTTONOBJ,
EZP_ANNOTATION_HOTSPOTOBJ

- ***AnnotationPaintMode*** **Data Type:** Integer (Read/Write)
  The current way annotations will paint:

  EZP_ANNOTATION_PAINT_OPAQUE, EZP_ANNOTATION_PAINT TRANSPARENT, or EZP_ANNOTATION_PAINT_TINTED

- ***AnnotationPattern*** **Data Type:** Integer (Read/Write)
  The current annotation pattern can include:

  EZP_ANNOTATION_PATTERN_HORIZONTAL,
  EZP_ANNOTATION_PATTERN_VERTICAL,
  EZP_ANNOTATION_PATTERN_FDIAGONAL,
  EZP_ANNOTATION_PATTERN_BDIAGONAL,
  EZP_ANNOTATION_PATTERN_CROSS,
  EZP_ANNOTATION_PATTERN_DIAGCROSS,
  EZP_ANNOTATION_PATTERN_NONE

- ***DocTypeAnnotation*** **Data Type:** Boolean (Read/Write)
  Indicates which annotation set with which you are currently working: local (FALSE) or global (TRUE).

## Annotation Methods

- ***AnnotationCopy()***
  This method undoes the last annotation copy.

- ***AnnotationCut()***
  This method undoes the last annotation cut.

- ***AnnotationDelete()***
  This method deletes the current annotation.

- ***AnnotationLayerAdd***
  *AnnotationLayerAdd (Type As Long, Comments As String, ViewGroup As String, HideGroup As String, EditGroup As String, DeleteGroup As String) As Boolean*

  This method adds an annotation layer.

- ***AnnotationLayerDelete***
  *AnnotationLayerDelete(LayerNum As Long) As Long*

  This method deletes the given annotation layer indicated by LayerNum.

- ***AnnotationPaste()***
  This method undoes the last annotation paste.

- ***AnnotationSave()***
  This method saves the annotations back to the screen.

- ***AnnotationTypeCanChangeDelete***
  *AnnotationTypeCanChangeDelete(Type As Long) As Boolean*

  This method returns whether or not deleting a group for an indicated annotation and type is allowed to be changed by the current user.

- ***AnnotationTypeCanChageEdit***
  *AnnotationTypeCanChangeEdit(Type As Long) As Boolean*

  This method returns whether or not editing a group for an indicated annotation and type is allowed to be changed by the current user.

- ***AnnotationTypeCanChangeHide***
  *AnnotationTypeCanChangeHide(Type As Long) As Boolean*

  This method returns whether or not hiding a group for an indicated annotation and type is allowed to be changed by the current user.

- ***AnnotationTypeCanChangeView***
  *AnnotationTypeCanChangeView(Type As Long) As Boolean*

  This method returns whether or not viewing a group for an indicated annotation and type is allowed to be changed by the current user.

- ***AnnotationUndo***
  *AnnotationUndo()*

  This method undoes the last annotation edit.

- ***CanDeleteLayer***
  *CanDeleteLayer(LayerNum As Long) As Error*

  This method given a layer indicates if the current user allows deleting.

- ***CanEditLayer***
  *CanEditLayer(LayerNum As Long)As Error*

  This method given a layer indicates if the current user allows editing.

- ***CanHideLayer***
  *CanHideLayer(LayerNum As Long)*

  This method given a layer indicates if the current user allows hiding.

- ***CanViewLayer***
  *CanViewLayer(LayerNum As Long)*

  This method given a layer indicates if the current user allows viewing.

- ***GetAnnotationLayerCreatedBy***
  *GetAnnotationLayerCreatedBy(LayerNum As Long) As String*

  This method returns who created a given layer.

- ***GetAnnotationLayerText***
  *GetAnnotationLayerText(LayerNum As Long) As String*

  This method returns the name of the given layer.

- ***GetAnnotationTypeDefaultDelete***
  *GetAnnotationTypeDefaultDelete(Type As Long) As String*

  This method returns the Default group for a given annotation type for deleting.

- ***GetAnnotationTypeDefaultEdit***
  *GetAnnotationTypeDefaultEdit(Type As Long) As String*

  This method returns the Default group for a given annotation type for editing.

- ***GetAnnotationTypeDefaultHide***
  *GetAnnotationTypeDefaultHide(Type As Long) As String*

  This method returns the Default group for a given annotation type for hiding.

- ***GetAnnotationTypeDefaultView***
  *GetAnnotationTypeDefaultView(Type As Long) As String*

  This method returns the Default group for a given annotation type for viewing.

- ***GetAnnotationTypeText***
  *GetAnnotationTypeText(TypeNum As Long) As String*

  This method, given an annotation layer type, returns its name (e.g., MarkUp, BlackOut, etc.)

## Using the POViewer Control

### *Viewing A Document*

POViewer1.hSession=POSession.hSession

POViewer1.hCursor=POQuery1.hCursor

POViewer1.hItem=hItem ‘ retrieved from POTree’s DocumentAction Event

POViewer1.Initialize

POViewer1.View()

# Dmg QBE Control

## Introduction

The Query By Example (QBE) control provides a common dialog for searching in several Documanage applications, including the Documanage Workstation, Documanage DocMigration utility, and the Open Document Management API (ODMA).

The QBE control generates and outputs folder, document, extended document, and full text filters that these applications use to search the

Documanage server database through the POQuery Control or the Document Migration tools. The control outputs these search filters through the control properties, which are described in the section on Properties. The control can also store search filters in XML files for later use.

## Limitations and Requirements

The QBE control provides a Query By Example dialog, not a full query builder. It does not include Boolean search capabilities other than through its Edit SQL tab, and it does not support building structured full text queries other than through its Full Text Search edit text field.

The QBE Control uses the Spread control to display tables. Other controls used by the QBE Control include POVolume, POFolder, PO_Document, POQuery and DMDTPicker.

## User Interface

The QBE control can be displayed in a dialog or in a form. You specify the title bar text displayed by the dialog. via the container for the control.

The bottom of the dialog displays:a Case Sensitive checkbox and an Indicate Empty Folders option list; and OK. Clear, and Clear All buttons.

A property allows you to hide or show each of these checkboxes and buttons, allowing you to provide your own buttons or programmatic interface.

- Clicking OK makes the control fire an event to the container telling it that the search filters are ready.
- Clicking Cancel fires an event telling the container telling it that no action should be taken.

- Clicking Clear clears the values in the fields on the current screen and set the operators back to their defaults.

- Clicking Clear All clears all fields on all of the screens.

Clicking Save and Open Query saves the query to the XML query file, reads from the XML file, and fires events indicating that the user has clicked a button. The path to the XML query file is usually provided by the container. If it does not provide a path, the control saves to a query file defined by a default path. Properties get and set the text in the Edit SQL screen, and get and set the contents of the forms on the properties screens. The control uses XML to format the formal data for saving and restoring. The current widths of the columns in the tables shown in the tabs are stored in the XML file and are restored when the XML file is read in.

## Behaviors

The QBE control maintains a document filter string, a folder filter string, and, if required, an extended document filter string. It builds these strings from the form data when the following actions take place: clicking OK; clicking Save; and switching between tabs (that is, if you switch from, say, the Folder tab to the Edit SQL tab, it builds the data in the Folder tab into the filter string); and programmatically calling the BuildFilter() method. The filter properties contain empty strings (their defaults) before these actions take place.

When it builds the filter string, the control tries to validate field entries for each data type unless it uses the BuildFilter method to build the filter string with Validate = FALSE.

**NOTE:** The control does not validate Between ranges since a Between query can be in either order—because a database engine is not sensitive to

order in a between clause, the control sorts these when building the query.

The control validates queries in tab field order. If it finds and displays an error, it highlights the invalid field and then stops the validation and query-building process. This allows the control to find only one validation error at a time. That is, if alphabetic characters are input to a numeric field, the control displays an error message when the query string is built.

An Edit button in the error dialog takes you back to the tab with the error; an Ignore button allows the control to build the filter even if there is an error. You can use the Ignore button to save bad queries for later editing, or to correct them in the Edit SQL dialog.

Clicking Cancel returns all properties, including those that define the filters, to the states that they were in when the control was invoked. This requires buffering the original property values during the session.

When the QBE control first appears, it displays the Folder Properties tab. In subsequent displays of the control, the tab that you selected last is in front. You can use the Front Tab property to programmatically select the front-most tab.

During a session, the control remembers the settings of all of the tabs between invocations. The control also saves a set of default settings for Operators and Category by storing them in an XML file that accommodates cabinet-specific defaults. This file is specified by the container application, but the control reads from and writes to it.

A Not Equal operator and a Does Not Contain clause (labeled Omits) is available for all field types.

## Properties

Properties for the QBE control include a session handle, a document table, a cabinet table, a case sensitive setting, a show empty folders setting, assorted properties for showing or hiding dialog elements, a path to query files, and optional filters that can preset the dialog. The QBE control also has properties for the session, document table, table name, SQL filters, dialog title, the case sensitive and empty folders checkboxes, errors, and tab display settings.

Output properties define filters that can be stored in XML files for later use. If the control writes a filter to an XML file, it appears in the Edit SQL dialog and any values in the field-oriented page for that filter are cleared. The filter properties are meant to be read from, but they can also be written to.

**NOTE:** To use the QBE control, initialize the following properties: hSession, (DocumentTable, TableName) OR FiltersXMLPath with tables set in the XML file.

- ***hSession*** **Data Type:** Long (Read/Write)
  The session to the server, input to the control. Default = 0.

- ***Cabinet*** **Data Type:** String (Read/Write)
  Input to the control, the name of the current cabinet. Default = NULL.

- ***Level*** **Data Type:** Long (Read/Write)
  Input to the control, the level number that should be the focus when the control is displayed. 1-based. Default = 1.

- ***DocumentTable*** **Data Type:** String (Read/Write)
  Input to the control, for use in building DocumentFilter. Default = NULL.

- ***TableName:*** **Data Type:** String (Read/Write)

Input to the control, the fully qualified table name for the current cabinet level. Default = NULL. If TableName is set but Cabinet is not, the Folder tab presents a single level corresponding to that table.

- ***Category:*** **Data Type:** String (Read/Write)
  The document category (doctype) selected in the Category screen.

- ***FolderFilter:*** **Data Type:** String (Read/Write)
  If input, displayed in SQL Edit screen; output to caller. Default = NULL.

- ***DocumentFilter:*** **Data Type:** String (Read/Write)
  If input, displayed in SQL Edit screen; output to caller. Default = NULL.

- ***ExtendedDocFilter*** **Data Type:** String (Read/Write)
  If input, displayed in SQL Edit screen; output to caller. Default = NULL.

- ***FullTextFilter*** **Data Type:** String (Read/Write)
  If input, displayed in Full Text Search screen; output to caller. Default = NULL.

- ***FullTextHelp*** **Data Type:** String (Read/Write)
  Sets the help text displayed in the Full Text Search screen.

- ***FiltersXMLPath*** **Data Type:** String (Read/Write)
  Path to XML file. Passed in by container for read or write. Refer to “XML Files” on page 179 for the XML schema. Default = NULL.

- ***CaseSensitive*** **Data Type:** Boolean (Read/Write)
  Specifies if query should be case sensitive. Default = FALSE.

- ***DBCaseSensitive*** **Data Type:** Boolean (Read/Write)
  Specifies if the data source is case sensitive. If this is FALSE, CaseSensitive is overridden, and the case sensitive checkbox is disabled,

overriding EnableCaseSensitiveBtn. Default = TRUE.

- ***Show Empties - DEPRECATED*** **Data Type:** Boolean (Read/Write)
  Specifies if empty folders are indicated. Default = FALSE.

- ***LastErrorNumber*** **Data Type:** Long (Read)
  Documanage error number. Default = 0.

- ***LastErrorSource*** **Data Type:** String (Read)
  Source of error. Default = NULL.

**NOTE:** The Display Page properties default to TRUE.

- ***DisplayFolderPage*** **Data Type:** Boolean (Read/Write)
  If true, show Folder screen, if false, hide it. Default = TRUE.

- ***DisplayDocumentPage*** **Data Type:** Boolean (Read/Write)
  If true, show Document screen, if false, hide it. Default = TRUE.

- ***DisplayCategoryPage*** **Data Type:** Boolean (Read/Write)
  If true, show Category screen, if false, hide it. Default = TRUE.

- ***DisplayFullTextPage*** **Data Type:** Boolean (Read/Write)
  If true show full text screen, if false, hide it. Default = TRUE.

- ***DisplayXDPPage*** **Data Type:** Boolean (Read/Write)
  If true show extended document properties table on Category screen, if false, hide it. Default = TRUE.

- ***DisplayEditSQLPage*** **Data Type:** Boolean (Read/Write)
  If true show Edit SQL screen, if false, hide it. Default = TRUE.

**NOTE:** The Show Btn properties default to TRUE.

- ***ShowOKBtn*** **Data Type:** Boolean (Read/Write)
  If true show OK button, if false, hide it. Default = TRUE.

- ***ShowCancelBtn*** **Data Type:** Boolean (Read/Write)
  If true show Cancel button, if false, hide it. Default = TRUE.

- ***ShowClearBtn*** **Data Type:** Boolean (Read/Write)
  If true show Clear button, if false, hide it. Default = TRUE.

- ***ShowClearAllBtn*** **Data Type:** Boolean (Read/Write)
  If true show Clear All button, if false, hide it. Default = TRUE.

- ***ShowSaveBtn*** **Data Type:** Boolean (Read/Write)
  If true show Save Query button, if false, hide it. Default = TRUE.

- ***ShowOpenBtn*** **Data Type:** Boolean (Read/Write)
  If true show Open Query button, if false, hide it. Default = TRUE.

- ***ShowCaseSensitiveBtn*** **Data Type:** Boolean (Read/Write)
  If true show Case Sensitive button, if false, hide it. Default = TRUE.

- ***ShowEmptyFoldersBtn*** **Data Type:** Boolean (Read/Write)
  If true show Show Empty Folders button, if false, hide it. Default = TRUE.

- ***EnableCaseSensitiveBtn*** **Data Type:** Boolean (Read/Write)
  If true enable Case Sensitive button, if false, disable it. Default = TRUE.

- ***AltEnabled*** **Data Type:** Boolean (Read/Write)
  If true enable alt key processing in the control to navigate to tabs and fire buttons, if false, alt keys are ignored and underlines in button and tab names are removed; must be set before calling Initialize(). Default = FALSE.

- ***FrontTab*** **Data Type:** Long (Read/Write)
  Specifies the tab to be displayed in front (1-based). Default = 1.

- ***DefaultsXMLPath*** **Data Type:** String (Read/Write)
  Path to XML file listing defaults for Category and operator values, by cabinet. Passed in by container for read or write. Refer to “XML Files” on page 179 for the XML schema. Default = NULL.

- ***StringOpDefault*** **Data Type:** Long (Read/Write)
  Default operator for all string field types (1=Includes, 2=Equals, 3=Omits, 4=Not Equal, 5=Begins With). Default = 1.

- ***NumOpDefault*** **Data Type:** Long (Read/Write)
  Default operator for all numeric field types (2=Equals, 4=Not Equal, 5=Greater Than, 6=Less Than, 7= Between, 8=Greater Than or Equal, 9=Less Than or Equal). Default = 2.

- ***FolderLabelWidth*** **Data Type:** Long (Read/Write)
  Width in pixels of folder table Label column.

- ***FolderOpWidth*** **Data Type:** Long (Read/Write)
  Width in pixels of folder table Operator column.

- ***FolderValWidth*** **Data Type:** Long (Read/Write)
  Width in pixels of folder table Value column.

- ***DocLabelWidth*** **Data Type:** Long (Read/Write)
  Width in pixels of document table Label column.

- ***DocOpWidth*** **Data Type:** Long (Read/Write)
  Width in pixels of document table Operator column.

- ***DocValWidth*** **Data Type:** Long (Read/Write)
  Width in pixels of document table Value column.

- ***CatLabelWidth*** **Data Type:** Long (Read/Write)
  Width in pixels of category table Label column.

- ***CatOpWidth*** **Data Type:** Long (Read/Write)
  Width in pixels of category table Operator column.

- ***CatValWidth*** **Data Type:** Long (Read/Write)
  Width in pixels of category table Value column.

- ***ExtLabelWidth*** **Data Type:** Long (Read/Write)
  Width in pixels of extended document table Label column.

- ***ExtOpWidth*** **Data Type:** Long (Read/Write)
  Width in pixels of extended document table Operator column.

- ***ExtValWidth*** **Data Type:** Long (Read/Write)
  Width in pixels of extended document table Value column.

- ***EmptyFoldersOption*** **Data Type:** Long (Read/Write)
  Indicates how empty folders should be displayed by the POTree control. Valid values are 0 = no special processing of empty folders; 1 = empty folders are indicated by a special icon in the tree; 2 - empty folders are not added to the list of folders in the tree.

## Methods

- ***Initialize***
  *Initialize()*

  Called after setting all the input properties. Displays the property sheet with appropriate field labels. Returns TRUE if OK, else FALSE.

- ***Clear***
  *Clear(long Screen)*

  Clear all fields on the specified screen (1-based) and reset operators to defaults. If pass in 0, all screens are cleared, as if Clear All had been pressed.

- ***BuildFilter***
  *BuildFilter(long Filter,BOOL Validate)*

  Build the filter specified in Filter (1=Folder,2=Document,3=Extended Document) as if the OK button had been pressed. This allows the filter property to be fetched, even if the user hasn't yet performed an action to assemble the filter. If Validate = TRUE, validation occurs otherwise it does not. Returns TRUE if OK, else FALSE.

- ***Save***
  *Save()*

  Tells the control to save the filters to the XML file specified in the FiltersXMLPath property. Returns TRUE if OK, else FALSE.

- ***Open***
  *Open()*

  Tells the control to open the XML file specified in the FiltersXMLPath property and load its contents into the control. Returns TRUE if OK, else FALSE.

- ***LaunchFTHelp***
  *LaunchFTHelp()*

  Tells the control to launch the Full Text Help file, as if the user had pressed the Full Text Help button on the Full Text Search tab. Returns TRUE if OK, else FALSE.

## Events

- ***OnOK***
  *OnOK*

  Occurs when the OK button is clicked.

- ***OnCancel***
  *OnCancel*

  Occurs when the Cancel button is clicked.

- ***OnSave***
  *OnSave (BOOL Cancel, String path)*

  Occurs when the Save Query button is clicked. Parameters are passed by reference. The container can pass back a TRUE in cancel to cancel the save, and can pass in a different path (perhaps from a Save dialog). Defaults to FALSE. The control sets the path to the XML file in the FiltersXMLPath property.

- ***OnOpen***
  *OnOpen (BOOL Cancel, String path)*

  Occurs when the Open Query button is clicked. Parameters can be passed by reference. The container can pass back a TRUE in cancel to cancel the Open, and can pass in a different path, perhaps from an Open dialog. Defaults to FALSE with the path set in the FiltersXMLPath property.

**NOTE:** The QBE control does not execute the SQL queries. This is because the queries may be executed through different controls depending on the application. The Workstation uses the POQuery control to execute the query and return a cursor, while the Document Migration utility uses the migration tools or controls to execute the query. The OnCancel, OnSave, OnOK and OnOpen events tell the container that their respective buttons have been clicked. The container can then respond with actions, such as fetching filters and executing them.

If the OK, Clear, and Cancel buttons are hidden, the you use the Clear() method to clear the forms. If the equivalent of an OK button is pressed, you call the BuildFilter method for each filter, then fetch the properties.

## XML Files

The QBE control uses XML files with a common schema to save and restore queries from forms and to save and restore defaults. The filter element stores queries from the Edit SQL and Full Text Search tabs.

Clicking Save saves the query to an XML query file; clicking Open Query reads from the XML file; clicking either button fires events indicating that the user has clicked the corresponding button.

The container for the control provides a path to the XML files; if it does not provide a path, the control saves the file using a default path. Control properties for getting or setting the text in the Edit SQL screen, and for getting or setting the contents of the forms on the properties screens are available.

The XML file stores the current width of each column in each table so that these widths can be restored when the XML file is read in.

The schema for the XML files is shown here:

```
<!ELEMENT settinglist (config,rowset*,filter* >
<!ELEMENT config
(cabinet,doctable,cabtable,folderlabelwidth?,folder
opwidth?,foldervalwidth?,doclabelwidth?,docopwidth?
,docvalwidth?,catlabelwidth?,catopwidth?,catvalwidt
h?,extlabelwidth?,extopwidth?,extvalwidth?)
<!ELEMENT cabinet (#PCDATA) >
<!ELEMENT doctable (#PCDATA) >
<!ELEMENT cabtable (#PCDATA) >
<!ELEMENT rowset (table,label,operator,value?) >
<!ELEMENT table (#PCDATA) >
<!ELEMENT label (#PCDATA) >
<!ELEMENT operator (#PCDATA) >
<!ELEMENT value (#PCDATA) >
<!ELEMENT filter (type,filtertxt) >
<!ELEMENT type
("FOLDER"|"DOCUMENT"|"EXTENDED"|"FULLTEXT") >
<!ELEMENT filtertxt (#PCDATA) >
```

A `settinglist` consists of the fully qualified names of the OT_DOCS and cabinet tables, (optionally) the width in pixels of each table column, and one or more `rowsets`.

The cabinet subelement to the config element stores the cabinet name. Level table numbers in the folder tab are specified starting at 101. That is, the first level is 101, the second level is 102, and so on.

A `rowset` consists of the table number (Folder Filter table = 1, Document Filter table = 2, Category table = 3 (includes the Category field, which isn't actually in the table but controls it), Extended Document table = 4), field label, operator (as described under properties) and an optional value. The optional value allows the schema to store defaults.

The filter element stores the actual assembled filter string, which may be from the Edit SQL or Full Text screens.

## XML File Example

Here is an example of the XML generated by the QBE control:

```
<?xml version="1.0"?>
<!-- DOCTYPE settinglist SYSTEM "settinglist.dtd" --
>
<settinglist>
<cabinet>author</cabinet>
<rowset>
<table>101</table>
<label>Fname</label>
<operator>Contains</operator>
<value>Fred</value>
</rowset>
<table>101</table>
<label>Mname</label>
<operator>Contains</operator>
<value>Rocky</value>
</rowset>
<rowset>
<table>101</table>
<label>Lname</label>
<operator>Contains</operator>
<value>Flintstone</value>
</rowset>
<rowset>
<table>102</table>
<label>City</label>
<operator>Contains</operator>
<value>Bedrock</value>
```

```
</rowset>
<rowset>
<table>2</table>
<label>AU ID</label>
<operator>Contains</operator>
<value>222-22-2222</value>
</rowset>
<rowset>
<table>3</table>
<label>category</label>
<operator>Contains</operator>
<value>abnormal</value>
</rowset>
<rowset>
<table>4</table>
<label>abnormal1</label>
<operator>Contains</operator>
<value>ab1</value>
</rowset>
</settinglist>
```

# DMDTPicker Control

The DMDTPicker control provides a user interface for manipulating dates and time. It is basically a wrapper that extends the Microsoft CDateTime and CMonthCal controls. The DMDTPicker control is packaged as an ActiveX control and can be used from a variety of Microsoft development tools including C++ and Visual Basic. It inherits the standard COleControl properties and methods such as Enabled, Font, Create(), etc.

When not dropped down, it displays a date/time string. When dropped down, it displays a date picker, a time picker, or both, depending on the data type.

Primary Dispatch Interface Properties:

| Property | Description | Data Type |
|---|---|---|
| DataType | Type of date string:<br>9 = Date,<br>10 = Time,<br>11 = Timestamp | long (Read/Write) |
| Value | String representing date/time | BSTR (Read/Write) |
| bDropped | True if control is currently in the dropped-down state | boolean (Read Only) |

Event Dispatch Interface Methods:

| Method | Description |
|---|---|
| void Change() | Notifies caller that the values in the control have changed |

## Example - Visual Basic

```
Option Explicit
Private Const EZP_TYPE_DATE = 9
Private Const EZP_TYPE_TIME = 10
Private Const EZP_TYPE_TIMESTAMP = 11
Private Const DATE_FORMAT As String = "yyyy-mm-dd"
Private Const DATETIME_FORMAT As String = "yyyy-mm-dd hh:nn:ss"
Private Const TIME_FORMAT As String = "hh:nn:ss"
```

```
Private Sub DMDTPtest_Change()
    Dim sData As String
    sData = DMDTPtest.Value
    MsgBox "The value is..." & sData, vbInformation
End Sub

Private Sub Form_Load()
    DMDTPtest.Font = Me.Font
End Sub

Private Sub optDataType_Click(Index As Integer)
    Dim sData As String
    Dim lDataType As Long

     Select Case Index
        Case 0
            sData = Format(Now, DATETIME_FORMAT)
            lDataType = EZP_TYPE_TIMESTAMP
        Case 1
            sData = Format(Now, DATE_FORMAT)
            lDataType = EZP_TYPE_DATE
        Case 2
            sData = Format(Now, TIME_FORMAT)
            lDataType = EZP_TYPE_TIME
        Case Else
            MsgBox "Unknown Option", vbInformation
            Exit Sub
    End Select

  With DMDTPtest
    .Font = Me.Font
```

```
    .DataType = lDataType
    .Value = sData
  End With

End Sub
```

# General Topics

## Refreshing folders when documents change

Some coding is involved in keeping the folder and documents shown in the visual controls current. Mainly this is done via the RefreshItem method shared by both the Tree and the Folder And Document control. RefreshItem takes a given folder and updates all the documents contained in it. This is only needed when documents are manipulated by multiple controls. The Tree control contains all folders and optionally all documents in a cursor. Only when the Tree is showing document must you worry about keeping it refreshed correctly.

When Refreshing is necessary:

- When using CheckIn , CheckOut, Import, SendTo, and in the Document control—Delete (note SendTo and Delete are not recommended when using the visual controls, use cut, copy, paste, and delete in the visual controls themselves) Any Visual Control containing that document's parent folder needs RefreshItem called with hItem set to the parent folder of the document. Remember the Tree contains all folders so if it is showing documents you will have to refresh it. Folder And Document control contains only one folder so only when it ParentHItem is equal to the source or destination hItems in any of the mentioned calls is it necessary to refresh.

- When someone Cuts and then Pastes the folder where the document was cut in needs refreshing—Simply remember in your code where the last Cut was executed. The FolderAction event is fired with EZP_ACTION_REFRESHCUTSOURCE when a cut then paste has occurred (the event happens in the control who "pasted."

- When a EZP_ACTION_REFRESHDEPENDENTS action occurs in a FolderAction—This means that the control automatically refreshed itself and any other controls showing the same folder indicated in the hItem property of the event needs refreshing. If you have only one control in the same Cabinet showing documents you can ignore this message.

## The Tree and GetRelativehItem

Be aware that attempting to use the Documanage Tree control in conjunction with GetRelativeItem can cause problems since it is designed to be used interactively and GetRelativeItem is for Programmatic use. A few simple rules must be kept in mind. The Tree control only retrieves children of items when it needs to. Any calls to Expand, CurrenthItem, or RefreshItem in the Tree with hItems retrieved from this call may generate EZP_ITEM_NOT_FOUND. This means that the Tree has not retrieved that item yet. A work around is to make sure all parent hItems to the hItem in question have been Expanded in the Tree. This can be accomplished by calls to GetRelativeItem with flag EZP_GET_PARENT and the Tree's Expand.

## Visual Controls vs. Programmatic Use

While all operations are available via the various non-visual controls, a lot of user interface work can be saved by using the Documanage visual ActiveX controls (mainly the Documanage Tree Control, Documanage Folder and Documents control, and the Documanage Folder List Control). The visual controls' main strengths include:

- User can navigate through entire cabinet visually with only a few lines of code (no need for GetRelativehItem calls)
- Provides a familiar and consistent interface between user-created applications and Documanage created applications (including icons)
- Automatic Drag and Drop between controls (and operating system) without coding.
- Simple methods for Copy, Paste, and Delete operations.
- Helpful events that are geared towards Documanage clients.

# Document Specifier Controls

## Introduction

The visual and non-visual controls provide access to documents through handles, which are only valid for a particular session and open cabinet. A Document Specifier, or DocSpecifier for short, provides a way to keep a permanent reference to a document between sessions. A DocSpecifier can refer to either the latest version (whatever it is, as it continues to change over time) or a specific version of a document.

The DocSpecifier consists of four parts:

- Cabinet—the name of a cabinet that contains the specified entity. (Note that when multiple cabinets contain the same entity, any of their names will suffice. The Cabinet is used to identify which document tables should be searched and to define a security context.)

- ID—the Entity ID of the entity, or, when Major Version and Minor Version are set for a historical version of a document, this may be the Document ID.

- Major Version—the major version number of the specified entity. For example, the Major Version would be 2 for version 2.03 of a document. A specifier with a Major Version of -1 indicates that the ID portion of the DocSpecifier must be interpreted as an Entity ID rather than a Document ID. This is used to identify the latest version of a document (since the latest version's Entity ID is always the document's Document ID) or as an alternative to specifying the version numbers for other entities.

- Minor Version—the minor version number of the specified document. For example, the Minor Version would be 3 for version 2.03 of a document. The Minor Version is ignored when the Major Version is -1.

Use the Document Specifier List Control to get document specifiers for documents that match a filter you provide. You can also use the visual and non-visual controls to find documents and retrieve the data needed for a specifier.

Use the Document Access Control to get the Documanage handles you need to access a document with the visual and non-visual controls.

Because DocSpecifiers refer to documents across different Documanage sessions, keep in mind that there is no guarantee the specified document will continue to be accessible. If the session is for a different user, or your access permissions change, or the document is moved to a different cabinet or deleted, you may not be able to use the DocSpecifier. If this happens, the Document Access Control will not provide you with the Documanage handles.

# Document Specifier List Control

The Document Specifier List Control gives you one-time access to a set of document specifiers that match criteria you specify through a filter.

To use the Document Specifier List Control, you must create an instance of the control, and set its Session, Cabinet, and Filter properties. Then, call its NextDocSpecifier method, and read the Cabinet, DocID, MajorVersion, and MinorVersion properties to obtain a DocSpecifier. Continue calling NextDocSpecifier until it returns FALSE, indicating the end of the list has been reached.

This control iterates over the list of DocSpecifiers in the forward direction only; once you call NextDocSpecifier, the previous specifier is no longer available through this control. There is no way to re initialize the control to start over or use a different filter. To do any of these things, you will need to create another instance of the control.

**NOTE:** This implementation may return some DocSpecifiers that, while valid, are ultimately inaccessible due to cabinet filters or insufficient permissions. The cabinet filters and permissions are not fully evaluated until a DocSpecifier is resolved, typically with the Document Access Control.

## Properties

- ***Cabinet*** ***Data Type: String***

  The name of a cabinet containing the documents for which you want to get specifiers. A full list of cabinet names can be obtained by calling the Query Control's GetCabinetName method. Changing this property after the NextDocSpecifier method has been called will have no effect on the control.

- ***DocID*** ***Data Type: Long (Read Only)***

  The DocID of the specifier retrieved with the last call to the NextDocSpecifier method.

**NOTE:** This property is not valid before the first call to NextDocSpecifier, or after NextDocSpecifier has returned FALSE indicating there are no more specifiers that match the filter.

- ***Filter*** ***Data Type: String***

  A specially formatted string that will be used to filter what documents' DocSpecifiers will be retrieved. The format of the Filter is a SQL Where Clause without the SQL keyword "WHERE". To filter on a particular Documanage Document property, you will need to know the corresponding database column name of that property (shown in the chart at the end of this document). Detailed formatting of Where clauses can be found in any SQL reference. Changing this property after the NextDocSpecifier method has been called will have no effect on the control.

- ***DocTypeFilter*** ***DataType: String***

  The DocType of the documents on which the user want to perform a eXtended Document Attribute (XDA) query.

- ***XDAFilter*** ***DataType: String***

  A specially formatted string that will be used to filter what documents' DocSpecifiers will be retrieved. The format of the Filter is a SQL Where Clause without the SQL keyword "WHERE". To filter on a particular Documanage eXtended Document Attributes (XDA), you will need to know the corresponding database column name of that property (using Document control). Detailed formatting of Where clauses can be found in any SQL reference. Changing this property after the NextDocSpecifier method has been called will have no effect on the control.

- ***hSession*** ***Data Type: Long***

  Your current session with the Documanage Server. You can get the correct value from the Session Control. This property should not be changed once the NextDocSpecifier method has been called.

- ***MajorVersion*** ***Data Type: Short (Read Only)***

  The Major Version (e.g., for version 2.03 this would be 2) of the document. For all DocSpecifiers retrieved through this control, MajorVersion will always be set for the specific document version, even

if it is the latest version. After a new version of the document is checked in later, the specifier will still access the earlier version.

**NOTE:** This property is not valid before the first call to NextDocSpecifier, or after NextDocSpecifier has returned FALSE indicating there are no more specifiers that match the filter.

- ***MinorVersion*** ***Data Type: Short (Read Only)***

  The Minor Version (e.g., for version 2.03 this would be 3) of the document. For all DocSpecifiers retrieved through this control, MinorVersion will always be set for the specific document version, even if it is the latest version. After a new version of the document is checked in later, the specifier will still access the earlier version.

**NOTE:** This property is not valid before the first call to NextDocSpecifier, or after NextDocSpecifier has returned FALSE indicating there are no more specifiers that match the filter.

## Methods

- ***NextDocSpecifier***

  *NextDocSpecifier() As Boolean*

  This method retrieves the next specifier that matches the Filter the user specified. After this has been called, the specifier may be retrieved by accessing the DocID, Cabinet, MajorVersion, and MinorVersion properties of the control.

  ***Return Value:***

  Returns TRUE if a DocSpecifier was successfully retrieved, and FALSE if there were no more DocSpecifiers that match the filter. After this method returns FALSE, the DocID, MajorVersion, and MinorVersion

properties are no longer valid.

***Example:***

This code gets specifiers for all documents in PersonalCabinet and displays them in a simple message box. POSession1 is a Session Control that has been properly connected to a Documanage server.

```
Dim SpecList As New PODocSpecifierListCtrl

SpecList.hSession = POSession1.hSession

SpecList.Cabinet = "PersonalCabinet"

SpecList.Filter = ""

While SpecList.NextDocSpecifier <> False

   MsgBox "Specifier: " & SpecList.Cabinet & ", " _

                & SpecList.DocID & ", " _

                & SpecList.MajorVersion & ", " _

                & SpecList.MinorVersion

Wend
```

# Document Access Control

This control "converts" a DocSpecifier into Documanage handles that are usable in a particular Documanage session. To use the Doc Access Control, set its hSession property to a valid Documanage session, then call the InitializeFromDocSpecifier method with the document specifier to load the hCursor, hFolder, and hItem properties with handles that are valid in that session.

If a DocSpecifier refers to a document that does not satisfy the cabinet filters or session permissions, an error is thrown. (The Document Specifier List Control may return such inaccessible DocSpecifiers.)

A single control can be used to get handles to several documents; simply repeat the call to InitializeFromDocSpecifier with a different specifier to change the handle properties to refer to the other document.

## Properties

- ***hCursor*** ***Data Type: Long (Read Only)***
  A Documanage cursor that can be used with the visual and non-visual controls to access the document or specific version.

- ***hFolder*** ***Data Type: Long (Read Only)***
  A Documanage folder that can be used with the visual and non-visual controls to access the folder properties for the specified document.

- ***hItem*** ***Data Type: Long (Read Only)***
  A Documanage item that identifies the specified document. If the specified version was the latest version of the document, this refers to a document object; if the version was an earlier version, this refers to a document "version" object, which is similar to a document but has some

restrictions (e.g., a document version cannot be edited or checked out).

Use the hItem along with the hCursor and hSession to initialize the Document Control with this item.

- ***hSession*** ***Data Type: Long***

  Your current session with the Documanage Server. You can get the correct value from the Session Control. The hCursor, hFolder, and hItem properties are only valid within the context of this particular session handle.

## Methods

- ***InitializeFromDocSpecifier***

  InitializeFromDocSpecifier(Cabinet As String, DocID As Long, MajorVersion As Short, MinorVersion As Short) As Long

  This method initializes the hCursor, hFolder, and hItem properties for use with the other Documanage controls to access the document identified by the passed specifier. If the DocSpecifier parameters do not define an accessible document (for which the Cabinet filters and session permissions allow access), this method throws an error.

  ***Parameters***:

  - Cabinet: The Cabinet portion of the Document Specifier
  - DocID: The DocID portion of the Document Specifier
  - MajorVersion: The MajorVersion portion of the Document Specifier. To retrieve the latest version of a document, pass -1.

- MinorVersion: The MinorVersion portion of the Document Specifier. Must be between 0 and 99. This parameter is ignored if MajorVersion is -1.

  ***Return Value:*** Returns a Long value that is reserved for future use and should be ignored.

***Example***

This code displays a message box with the name and version of each document specified in a DocSpecifierList control. POSession1 is a Session Control which has already been connected to a Documanage server. SpecList is a DocSpecifierList Control which has been prepared by setting the hSession, Cabinet, and Filter properties. PODocument1 is a Document Control whose hSession property has already been set.

```
DocAccess.hSession = POSession1.hSession

While SpecList.NextDocSpecifier <> False

   DocAccess.InitializeFromDocSpecifier SpecList.Cabinet, _

                                        SpecList.DocID, _

                                        SpecList.MajorVersion, _

                                        SpecList.MinorVersion


   PODocument1.hCursor = DocAccess.hCursor

   PODocument1.hItem = DocAccess.hItem

   PODocument1.Initialize
```

```
MsgBox "Document: " & PODocument1.Name & " " _

& PODocument1.Version

Wend
```

# Custom Workflow Tasks

## Workflow Daemon

The Documanage Workflow Daemon is an application that automates workflow tasks that might normally be completed by human users. It can be run either as an NT service or as a console application. The daemon application logs into the Documanage system with a user account, periodically checks the logged-in user's inbox for pending tasks and runs custom-programmed rules to process each task. The custom rules have access to the Documanage session and can manipulate the workflow project and associated documents as desired, based on the privileges of the logged-in user account.

The daemon does not do any task processing itself, but instead makes use of one or more custom dlls to process tasks. These dlls use the Documanage API (DmgAPI) to check out, process, and check in or forward tasks.

### Installation

The workflow daemon and associated sample files are supplied on your distribution media as part of the Documanage Software Development Kit. This also kit includes the DmgAPI for Win32 which you will need to write the custom libraries you need to supply to use the daemon application. There is no installation required other than copying the files once you have the Documanage Workstation and the DmgAPI installed on your development computer. You can use the Documanage Workstation as a workbench tool to

see the data your libraries will manipulate within Documanage and to generate test data for your code.

Copy the folders with the workflow daemon and the DmgAPI from the SDK onto your hard drive.

## Notation

In this section, syntax is specified in a different typeface. Text that must be specified literally as shown is in bold. Placeholder names for syntax elements that are to be replaced by actual meaningful text are shown in italics. In many cases, parts of the syntax are optional. These optional portions are enclosed in square brackets to indicate they are optional. The square brackets are not part of the syntax itself. When brackets are nested, outer syntax is required in order to properly specify inner syntax. For example:

**name.exe** [**-x**[path]]

indicates that "name.exe" needs to be typed as shown. In order to specify an optional real "path" text, you must first specify the "-x" syntax as shown and that both path and the entire "-x" along with a path is optional. So, an actual case of this syntax might be:

**name.exe -xc**:\mydirectory\

## Running the Daemon

The Workflow Daemon command-line program may be run as a console application or configured as a Windows NT service. While you are developing your custom library routines, you should run the program from the command line for easier debugging and control. Once your libraries are debugged and ready, you should still test them running as a service if that is the target environment. The command line syntax is shown here:

**DMGWorkDaemon.exe** [options]

Options are specified as a space-separated set of the following items. When an option takes an argument, there is a space between the option and its argument. The **/p** and **/o** options are mutually exclusive and the **/i** and **/u** options should be the only options specified when they are used.

### Help and Installation

| | |
|---|---|
| **/h** or **?** | Display program command-line help |
| **/i** or **/u** | Install (**/i**) or uninstall (**/u**) program as an NT service |

### Run Mode / Connecting to Documanage

| | |
|---|---|
| **/p** *profilename* | Run as program. User profile name (see DmgAPI Help) to use to connect to Documanage. REQUIRED if run from command-line. |
| **/o** *profilename* | Run as service (default). User profile name (see DmgAPI Help) to use to connect to Documanage. |
| **/a** *address* | DNS name or IP address of Documanage router computer |
| **/r** *protocol* | Protocol to use to Documanage router |
| **/e** *endpoint* | Protocol-specific endpoint to use to Documanage router |
| **/m** *domain* | Domain of user name (/n) to use to log into Documanage |
| **/n** *username* | User name from domain (/m) to use to log into Documanage |
| **/w** *password* | Password of user name (/n) to use to log into Documanage |

### Execution Control

| | |
|---|---|
| /s *seconds* | Polling interval in seconds (time between task queue scans) |

### *Logging*

| | |
|---|---|
| **/l** *filename* | Name of log file (default = "DMWorkDaemon.log") |
| **/z** *kbytes* | Minimum log file size in K (default = 100) |
| **/x** *kbytes* | Maximum log file size in K (default = 200) |
| **/v** *loglevel* | Amount of detail to include in the log file (default = 0) |
| **/d** *setting* | If 1, include extra log detail (default 0) |
| **/b** *setting* | If 1, disable all logging (default 0) |

The following is an example of using some of these options:

```
dmworkdaemon.exe -p dprof -n bill -m acme -w pw -a mordor
-e 4000 -r ncacn_ip_tcp -s 5
```

## Run Mode

When run as a service, the daemon application makes calls to the Windows Service Control Manager (SCM) and expects control messages from it. This is the default run mode for the daemon. To run as a normal application without interfacing to the SCM, use the **/p** option. To specify a profile (next section) and still run as a service, use the **/o** option.

## Connection Parameters

The daemon application uses the DmgAPI to connect to and interact with Documanage. Refer to the DmgAPI help for additional information. In order to connect, the daemon creates or uses and existing saved DmgAPI connection profile. If you have a saved profile, you may name it with the **/p** or **/o** option and specify no other connection parameters. In order to easily create a profile, you may run the daemon with the name you want to create as the **/p**

option and specify the contents of the profile in the other connection options, **/a**, **/r**, **/e**, **/m**, **/n**, and **/w**.

If the **/o** option is omitted, the program runs as a service by default as mentioned above. Default credentials (the Windows account used to log into the computer) are used. A user profile called "Default" is created with blanks for user, password, and domain which is the key to using the Windows account credentials. The program may be run with the **/p** option referring to a like profile to use the Windows account credentials yet not run as a service.

If the Documanage router information is not specified, the daemon uses the information in the **[Router]** section of the POFFICE.INI file.

## Logging Parameters

The **/v** log level is currently not used. It may be used in the future to determine how much detail is written to the log file; the lower the level, the less logging takes place. The **/d** extra detail flag causes some extra information to be logged for certain errors.

Specifying **/b 1** will turn off all logging.

### *Setting Up Task Instructions*

The Workflow Daemon fetches the **Instructions** property for each task it finds in its queue and parses it to determine if the task can be processed by the daemon. If the Instructions don't contain the right information, the task is skipped and an error message is logged. The task instructions should be in the following format:

```
dllname[|COM control name[|entrypointname[|parameters]]]
```

The pipe **(|)** symbol is used as a delimiter, and the list is positional. For example, if the instruction contains the dllname and a parameter string, all the pipe characters up to the parameter are included:

```
dllname|||parameters
```

The dll name is the exact name of the dll including the ".dll" extension. Examples of strings which might appear in the Instructions field are:

***MyPlainDLL.dll***

***WFCom.dll|MyCOMControl.MyCOMControl.1***

***MyEMailer.dll|||user@acme.com,"Subject",file:///C:/IF.txt***

***MyWFLibrary.dll||Email|user@acme.com,"Subject",file:///C:/IF.txt***

***MyWFLibrary.dll||WaitForForms|1099,518A***

The **Instructions** property is filled in by the user who designs the workflow, using the Documanage Workflow Designer. See the documentation on that program for more details on how workflows are designed and task properties are set.

The daemon interfaces to simple Win32 Dynamically Loaded Libraries (DLL's) for its processing. Some programming tools such as Microsoft Visual Basic, are incapable of producing simple dlls but instead produce COM controls. To allow workflow processing libraries to be written as COM controls, a COM compatibility dll, **WFCom.dll**, is included with the Workflow Daemon. This allows COM controls to be called through intermediation.

**NOTE:** The Workflow Daemon expects to load an "in process" library and pass process-specific information—a Session Handle, for instance—to that library. This library must use the same instance of the

DmgAPI library as the Workflow Daemon. If a COM object is defined as "out-of-process" such that a new application is launched separately from the Workflow Daemon, the session context information that the Workflow Daemon passes to it will be unusable. WFCom.dll does not call out-of-process COM objects for this reason.

If a COM compatibility dll is being used, the first entry in the instructions list will be the name of the compatibility dll as in the second example above. The second parameter, the COM control name, will then contain the programmatical name of the COM control to load, for example, **MyCOMControl.MyCOMControl.1**.

The *entrypointname* is an optional string with an internal function name to perform. This is interpreted by the library code and is opaque to the daemon application.

Optional *parameters* can be included as a "fourth field." These parameters (anything past the third field) are not used or processed by the daemon application in any way. Your library will extract any additional parameters from the instructions field and parse this data itself - no format is imposed by the workflow daemon application. The *parameters* string may therefore be in whatever format desired by your library. The entire Instructions property is limited to 254 characters in length.

## Writing Daemon-Called Libraries

You will write one or more libraries of functions which are packaged as a DLL or COM object and installed on the computer in the same directory as the workflow daemon application. Your library is then loaded and executed on demand by the daemon application as it encounters workflow tasks that reference it. Any number of libraries may be installed as long as they are packaged with different file names. Libraries and COM objects are loaded the

first time they are encountered in a task Instructions field. Thereafter, the daemon keeps the libraries and COM objects loaded until it terminates.

Each standard DLL library to be called by the daemon must implement the following two C-language interfaces:

```
long PINotify(long hSession, long hTask, long nFlag, long nSpare);
long PIExecute(long hSession, long hTask, long hProject, char* strEntrypoint);
```

COM controls must implement these as methods which are called by the **WFCom.dll** library after making a CreateDispatch() call with the COM object identifier from the Instructions field. (This call is only made once as is the LoadLibrary() call on the standard DLL. See above.)

### *PINotify()*

**long PINotify(long** *hSession*, **long** *hTask*, **long** *nFlag*, **long** *nSpare*);

| | | |
|---|---|---|
| hSession | in | DmgAPI session handle of open Documanage server session |
| hTask | in | DmgAPI task handle to the task being processed |
| nFlag | in | Reason for call: 1 = Start Task, 2 = End Task, and 3 = Goodnight |
| nSpare | in | undefined; unused at this time |

PINotify() is called each time the Workflow Daemon starts processing a new task, and when it is finished with the task. It is also called when the Workflow Daemon is shut down, to deliver a "goodnight kiss" to the DLL. All parameters are passed in. Additional nFlag values may be introduced in the future, so your code should allow for unrecognized values and ignore them. A spare long parameter is also provided for future expansion. Its value is currently undefined. Your library can return error codes to the Workflow Daemon. A zero value indicates success.

You may use the PINotify() notification call for whatever reasons you need. There are no required responses to any of the PINotify calls. For example, you may use this call to tell the library to check for abnormal numbers of projects associated with a task, or for logging into a resource, logging/reporting, etc.

### *PIExecute()*

**long PIExecute(long** hSession, **long** hTask, **long** hProject, **char*** *strEntrypoint***);**

| | | |
|---|---|---|
| hSession | in | DmgAPI session handle of open Documanage server session |
| hTask | in | DmgAPI task handle to the task being processed |
| hProject | in | DmgAPI project handle to the project folder |
| strEntrypoint | in | opaque string interpreted by your code |

After the PINotify(start task) call, the daemon calls PIExecute() and passes in the session, the task handle, the project handle, and an optional string intended to indicate what internal function the library will execute. Your library will extract properties from the task handle, including the instructions property and determine what needs to be done to process the task. Your library may use the handles passed to it by the daemon program to access Documanage, conduct queries, check out/in documents, or whatever it needs to do. All actions take place under the user identity of the daemon application session. Once completed, you library is responsible for making the decision to advance the project to the next task, suspend it, check it back in to be processed again on the next daemon cycle, forward it to another team member or to leave the project checked out. This could involve merely advancing the project or if the current task is a Human Decision task, making the correct advance calls to implement the decision.

Once the task is processed, your library returns from the PIExecute() call. Your library can return error codes to the Workflow Daemon. A zero value indicates success.

The Documanage Workflow Daemon does not maintain state between calls to plug-ins. Any state must be maintained by the plugin. Keep in mind that the order in which dlls and their functions are called cannot be guaranteed, so any state preservation must take this into account.

For details on using the session, task and project handles, see the DmgAPI documentation.

# Workflow Daemon Plugins

The Documanage Workflow Daemon polls the logged-in user's inbox for pending workflow tasks, then processes them using custom plugin libraries. These libraries can make use of the dmg_api to check out, process, and check in or forward tasks. This provides a powerful and flexible mechanism for extending the capabilities of the Documanage workflow system and automating certain tasks.

The Workflow Daemon makes use of standard dynamic link libraries (DLLs). COM libraries can be used through the intermediary of the COM compatibility plugin, WFCom.dll.

## Workflow Daemon Calls

The Workflow Daemon plug must implement the following calls:

long PINotify(long hSession, long hTask, long nFlag, long nSpare);

long PIExecute(long hSession, long hTask, long hProject, char* strEntrypoint);

PINotify() will be called each time the Workflow Daemon starts a new task, and when it is finished with the task. It will also be called when the Workflow Daemon is shut down, to deliver a "goodnight kiss" to the dll. The session and the task handle are passed in. The nFlag parameter will be set to 1 for Start Task, 2 for End Task, and 3 for Goodnight. Future flag values may be introduced in the future, and a spare long parameter is provided for future expansion. The PINotify notification may be used to tell the library to check for abnormal numbers of projects associated with a task, or for logging into a resource, reporting, etc.

When the daemon calls PIExecute(), it passes in the session, the task handle, the project handle, and an optional string indicating what internal method the library will execute. The library will extract properties from the task handle, including the instructions property.

## Workflow Daemon Tasks

Tasks that can be processed by the Workflow Daemon maintain processing directions in the instructions property of the task. The task instructions are in the following format:

```
dllname.dll[|optional COM control name][|optional
entrypointname][| optional parameters]
```

The pipe (|) symbol is used as a delimiter, and the list is positional. For example, if the instruction contains the dllname and a parameter, all the pipe characters up to the parameter are included: dllname.dll|||parameter. The dll name is the exact name of the dll, plus the .dll extension.

The instructions property will contain any additional parameters needed by the library after the formatted specification of what library to call, and the

library will parse this data itself - no format is imposed on the additional parameter field. However, the entire instructions property is limited to 254 characters in length.

The Documanage Workflow Daemon does not maintain state between calls to plugins. Any state must be maintained by the plugin. Keep in mind that the order in which dlls and their functions are called cannot be guaranteed, so any state preservation must take this into account.

The plugin can return error codes to the Workflow Daemon. A zero value indicates success.

For details on using the session, task and project handles, see the dmg_api documentation.

Source listings for a simple sample plugin follow. This plugin checks out a project and advances it without doing anything to it. It illustrates the required entrypoints (PINotify and PIExecute) and the techniques for checking out and advancing the project.

## Sample Header File

```
#ifdef WFSAMPLE_EXPORTS
#define WFSAMPLE_API __declspec(dllexport)
#else
#define WFSAMPLE_API __declspec(dllimport)
#endif
typedef enum {
   FLAG_START=1,
   FLAG_END,
   FLAG_GOODBYE
} e_NotifyFlag;
```

```
 bool nNotified;
 #define min(a, b)  (((a) < (b)) ? (a) : (b))
 #ifdef __cplusplus
extern "C" {
#endif
 WFSAMPLE_API int PINotify(long hSession, long hTask,
e_NotifyFlag nFlag, long nSpare);
 WFSAMPLE_API int PIExecute(long hSession, long hTask,
long hProject, char* strEntryPt);
 #ifdef __cplusplus
}
#endif
```

## Sample C++ File

## Sample Plugin Code

This simple sample plugin illustrates the basics of a DMGWorkflowDaemon plugin library. The header file declares the interface and the flags that can be passed into PINotify(). The source file shows simple implementations of PINotify and PIExecute.

In this example, PINotify() simply sets an internal flag if it has received the FLAG_START flag. In an actual implementation, PINotify might respond differently to each of the three flags, perhaps setting up environment variables or checking status.

PIExecute() calls an internal routine, GetParameters(), which illustrates parsing the instructions property to fetch parameter data. It then checks out the project with the dmg_api function DmgProjectCheckOut(). It gets the

branch list with DmgProjectGetBranchList(), and gets the first handle from that list with DmgListGetFirstHandle().

It continues by getting the branch ID using DmgHandleGetPropertyByName() and DmgPropertyGetValue().

At this point in an actual working plugin, the project would be processed in whatever way is appropriate. This sample plugin does nothing to the project other than the final step, advancing it. It does that by calling DmgProjectCheckInToBranch().

```
// WFSample.cpp : Defines the entry point for the DLL.
//
#include "string.h"
#include "dmg_api.h"
#ifdef _DEBUG
#include <iostream.h>
#include <stdlib.h>
#endif
#include "WFSample.h"
#ifdef __cplusplus
extern "C" {
#endif

WFSAMPLE_API int PINotify(long hSession, long hTask,
e_NotifyFlag nFlag, long nSpare)
//
// You can use the PINotify call to check for unusual
conditions like
// too many projects associated with a task, or to do
initialization.
//
```

```
{  nNotified = nFlag == FLAG_START;// nNotified true if
start task, else false
#ifdef _DEBUG
   char nn[255];
   cout << "PINotify: nFlag = " << _itoa(nFlag,nn,10) <<
endl;
#endif
   return 0;
}
 void GetParameters(long hTask, char** strParams)
//
// Utility function to parse out the parameters from the
// Instructions property. If you don't expect parameters
in the
// Instructions property, you can skip this.
//
{
                        long result;
   long hProperty = 0;
   char* strInstructions=NULL;
   long sLen = DMG_P_TASK_INSTRUCTIONS_LEN;
   //
   // Get Instructions property using the dmg_api.
   //
   result = DmgHandleGetPropertyByName(hTask,
DMG_PROP_BASIC, DMG_P_TASK_INSTRUCTIONS, &hProperty);
   if (result == DMG_SUCCESS && hProperty) {
      result = DmgPropertyGetValue(hProperty,
DMG_DATATYPE_STRING, NULL, &sLen);
      if (result == DMG_SUCCESS) {
```

```
            strInstructions = new char[sLen];
            result = DmgPropertyGetValue(hProperty,
DMG_DATATYPE_STRING, strInstructions, &sLen);
         }
         if (result == DMG_SUCCESS &&
strlen(strInstructions) > 0) {
#ifdef _DEBUG
            cout << "Instructions: " << strInstructions <<
endl;
#endif
            //
            // Got the property, now parse the string
            // Look for first pipe, between plugin name and
OCX GUID
            //
            char* ocxOffset = strchr(strInstructions,0x7c);
// "|"
            if (ocxOffset) {
               //
               // Found pipe. If no pipe, there are no
entrypoint name or parameters.
               //
               // Parse past the plugin name (we aren't
going to use it)
               //
               ocxOffset += 1;
               //
               // Look for second pipe, between OCX GUID and
entrypoint name
               //
               char* entryptOffset = strchr(ocxOffset,0x7c);
```

```
            if (entryptOffset) {
               //
               // Found pipe
               // Parse past the OCX GUID (we aren't
going to use it)
               //
               entryptOffset += 1;
               //
               // Look for third pipe, between entrypoint
and parameters,
               //
               char* paramOffset =
strchr(entryptOffset,0x7c);
               if (paramOffset) {
                  //
                  // Copy parameters into strParams
                  //
                  paramOffset += 1;
                  sLen = strlen(paramOffset);
                  *strParams = new char[sLen+1];
                  strcpy(*strParams,paramOffset);
#ifdef _DEBUG
                  cout << "Parameters: " << *strParams <<
endl;
#endif
      //
      // Delete this when done with it
      //
      if (strInstructions)
         delete[] strInstructions;
   }
```

```
}

WFSAMPLE_API int PIExecute(long hSession, long hTask,
long hProject, char* strEntryPt)
//
// Executes the functionality of the plugin. In this
example, there are
// no additional entrypoints so strEntryPt is ignored.
//
{
   long result = 0;
   long chkResult = 0;// Checkout result
   long hList = 0;
   long hBranch = 0;
   long hProperty = 0;
   long nBranchID = 0;// Default
   char* strParams=NULL;// Optional parameters
   //
   // If you expect parameters to be passed in the
Instructions field,
   // parse them out.
   //
   GetParameters(hTask,&strParams);
   //
   // Check out the project
   //
   chkResult = DmgProjectCheckOut(hProject);
   //
   // Get the first branch so we can check the project
back in.
```

```
   // An alternative is to determine what kind of task
this is - get
   // the taskType for the project (basic property
DMG_P_PROJECT_TASKTYPE)
   // then see if the type is DMG_TASK_HUMANDECISION. If
it's not a human
   // decision, just use 0 as the branchID.
   //

   if (chkResult == DMG_SUCCESS) {
      //
      // Get list of branches
      //
#ifdef _DEBUG
      cout << "Checkout successful" << endl;
#endif
      result = DmgProjectGetBranchList(hProject,&hList);
   }
   if (result == DMG_SUCCESS && hList != 0) {
      //
      // Get first branch
      //
#ifdef _DEBUG
      cout << " Got Branch List" << endl;
#endif
      result = DmgListGetFirstHandle(hList,&hBranch);
   }
   //
   // If there's no branch, we'll use a branchID of 0
(defaulted in
   // declaration at top of this method). If there is
```

```
   // a branch, get its branchID and use it.
   //

   if (result == DMG_SUCCESS && hBranch != 0) {
      //
      // Get Id property
      //
#ifdef _DEBUG
      cout << " Got first branch" << endl;
#endif
      result =
DmgHandleGetPropertyByName(hBranch,DMG_PROP_BASIC,"Id",&
hProperty);

      if (result == DMG_SUCCESS && hProperty != 0) {
         //
         // Get the branch Id
         //
#ifdef _DEBUG
         cout << " Got Id Property" << endl;
#endif
         long len = 4;
         result =
DmgPropertyGetValue(hProperty,DMG_DATATYPE_LONG,&nBranch
ID,&len);
#ifdef _DEBUG
         char nn[255];
         cout << " Id = " << _itoa(nBranchID,nn,10) <<
endl;
#endif
      }
```

```
    }

    //
    // Do interesting things to your project here.
    //

    //
    // When finished, we need to advance or forward the
project before
    // returning. In this case we advance it.
    //

    if (chkResult == DMG_SUCCESS) {
        //
        // Check into branch identified by Id.
        //
        chkResult =
DmgProjectCheckInToBranch(hProject,nBranchID);
#ifdef _DEBUG
        char nn[255];
        cout << "Checkin result = " <<
_itoa(chkResult,nn,10) << endl;
#endif
    }
    //
    // Clean up
    //
    delete[] strParams;

    return chkResult;
```

```
}

#ifdef __cplusplus
}
#endif
```

# Workstation Automation Feature

## Introduction

The Workstation Automation Feature is an ActiveX extension to the Documanage Workstation. It allows external applications to "launch" the Workstation and programmatically open items in the Workstation.

The Workstation exposes a publicly creatable ActiveX object class. If an external application creates an instance of the public object, the workstation is automatically launched if it is not currently running. If it is running, the object is created on the running process.

The external application is then free to call methods on the public class, which can open pending or suspended projects, cabinets containing a requested folder or document, or to open a cabinet with a full QBE specification.

The OpenFolder, OpenDocument, and OpenProject interfaces are fairly simple. However, the OpenQuery interface has a complex syntax with two variations. One replicates the functionality of the QBE dialog in the workstation. A second allows for multi-level cabinet folder filter conditions, order by clauses, and OR conditions.

# Interfaces

The DmgClient exposes a publicly creatable ActiveX object. When an instance of this object is created by an external application, the Windows operating system automatically launches the DmgClient if it is not already running. If DmgClient is running the external application attaches to the running process. Thus, only one instance of the DmgClient is ever running on the user's computer.

The object has the following classID: *DmgClient.DmgQuery*

After the object is created, the external application must call its .Init() method. This launches the application if it's not already running and logs into Documanage. This is the equivalent of launching the DmgClient by double-clicking on it's executable, or launching it from a menu. If DmgClient is already running, the .Init() method simply returns a SUCCESS.

Once the DmgClient was running. It continues to run even if the external application destroys all references to the DmgQuery object.

The ActiveX interface does not affect the existing Documanage Workstation functionality in any way. If desired, it can continue to be used as a stand-alone application without ever invoking the DmgQuery interface.

## Init

The Init method call launches the Documanage Workstation if it is not already running and logs in to Documanage. If the Workstation is already running, it simply returns a SUCCESS. The external application must call this method prior to calling any of the request methods. Calling other methods without first calling .Init() will return an error.

```
DmgQuery.Init(
) As Long.' 0 - success | -1 - failure
```

## Open Project

This method call opens a pending or suspended project in an Active Project dialog. The project is checked out to the user and is available for processing.

```
DmgQuery.OpenProject(          ' open a pending or suspended project
    ByVal lTaskID as Long,     ' project task ID
    ByVal sType as String,     ' project type "P" | "S"
    ByVal sKey as string,      ' project key (see below)
    ByVal sLabel as String,    ' project label
    ByVal lAction as Long      ' reserved for future use, pass 0
) As Long ' 0 = success | -1 = error
```

The equivalent Documanage Workstation operation is as follows:

1. **From the File Room Open a Task Queue (either Pending or Suspended)**
2. **From the Task Queue window, select a project, right-click and select "Work on Project"**

## Open Document

This method call opens a cabinet which contains the document corresponding to the Document Key. The document is contained in its parent folder. If the cabinet specified is the Personal Cabinet, the Personal Cabinet is opened or brought to the front without regard to the rest of the document specification (i.e. we don't allow the Personal Cabinet to be opened with a filter). Also, only the current version of the document can be opened, so the major and minor version parameters of the DocSpecifier are ignored.

```
DmgQuery.OpenDocument(  ' open a cabinet containing the document
    ByVal sKey as string,      ' document key (see below)
    ByVal sLabel as String,    ' document label
```

```
        ByVal lAction as Long     ' see Action Parameter Bitmask, below
    ) As Long ' 0 = success | -1 = error | 1 = warning
```

The equivalent Documanage Workstation operation is as follows:

1. **From the File Room Select a Cabinet and right-click. Select Open Cabinet with Query**
2. **From the Query dialog, specify the Document query criteria (although DocID is not available on this dialog)**
3. **An instance of a Cabinet containing the requested document is opened.**

## Open Folder

This method call opens a cabinet which contains the folder corresponding to the Folder Key. It is envisioned that this call pertains to a single folder, although that is not enforced (as the Folder Key is used as a folder filter without being examined or parsed). If the cabinet specified is the Personal Cabinet, the Personal Cabinet is opened or brought to the front without regard to the rest of the folder specification (i.e. we don't allow the Personal Cabinet to be opened with a filter).

```
DmgQuery.OpenFolder(      ' open a cabinet containing the folder
    ByVal sKey as string,     ' folder key (see below)
    ByVal sLabel as String,   ' folder label
    ByVal lAction as Long     ' see Action Parameter Bitmask, below
) As Long ' 0 = success | -1 = error | 1 = warning
```

The equivalent Documanage Workstation operation is as follows:

1. **From the File Room Select a Cabinet and right-click. Select Open Cabinet with Query**
2. **From the Query dialog, specify the Folder query criteria**
3. **An instance of a Cabinet containing the requested folder is opened.**

## Open Query

This method allows a cabinet to be opened with a specified filter condition. It is envisioned that this call would pertain to multiple folders and/or documents, rather than the single instance calls detailed above (OpenDocument,. OpenFolder). However if the cabinet specified is the Personal Cabinet, the Personal Cabinet is opened or brought to the front without regard to the rest of specification (i.e. we don't allow the Personal Cabinet to be opened with a filter).

There are two specification formats proposed (see Appendix A). Additional formats may be added in the future. Format 0 is equivalent to a specification derived from the QBE dialog. Format 1 has no equivalent in the workstation and would allow multi-level cabinet queries, order by clauses, and OR terms.

```
DmgQuery.OpenQuery(                    ' open a cabinet filtered
    ByVal sQuerySpecification As String, ' query specification (see below)
) As Long ' 0 = success | -1 = error | 1 = warning
```

The equivalent Documanage Workstation operation for Format 0 is as follows:

1. **From the File Room Select a Cabinet and right-click. Select Open Cabinet with Query**
2. **From the Query dialog, specify the query criteria**
3. **An instance of a Cabinet filtered by the query conditions is opened**

There is no Documanage Workstation equivalent for Format 1. It requires that the external application construct fully qualified SQL clauses.

## Error Information

If a method call fails, the external application can retrieve the following properties to determine the circumstances of the error.

LastErrorAdvice() As String - suggests what to do about the error
LastErrorAdviceID() As Long -- [ Not Used ]
LastErrorDescription() As String - description of the error
LastErrorNumber() As Long - number of the last error (not very helpful)
LastErrorSource() As String - code module that triggered the error

## Action Parameter Bitmask

The lAction parameter of the OpenFolder and OpenDocument methods is setup as a bitmask. Documanage Workstation versions prior to 6.4.2 assume this parameter has a zero value and ignore it. Documanage Workstation version 6.4.2 exposes the following enumeration:

```
Public Enum WA_ACTION
   WA_REFRESH = 1
End Enum.
```

The initial call to OpenFolder or OpenDocument should have an lAction parameter value of zero. If your code subsequently does something that might alter the contents of the Cabinet window that was opened by the initial call, you can make a subsequent call to OpenFolder or OpenDocument with the WA_REFRESH bit in the lAction parameter turned on. This call will refresh the content of the window and bring it to the front. If the user has closed the window the call will re-open it. The other parameters of the OpenFolder or OpenDocument method should have the exact values as the initial call. This is illustrated in the example shown here.

## Example

The following routine illustrates how an external application could invoke the Documanage Workstation to open a cabinet containing the requested document.

```
Private Sub openDocumentItem(ByVal sKey As String, ByVal
sLabel As String)
' -------------------------------------------------------
' --- open cabinet containing document in DmgWorkstation
' -------------------------------------------------------
'
' sKey   -  document key
' sLabel -  document label
' -------------------------------------------------------
'
' error handling
'
Const K_Proc = "TS_DmgxTree.openDocumentItem"
m_TKCorigin = ""
m_TKCerror = ""
m_TKCadviceID = 0
On Error GoTo ErrorHandler
'
```

```
' dimension local variables
'
Dim oDmgQuery As DMgClient.DmgQuery
Dim lRC As Long
Dim lAction as Long
'
' create new DmgQuery object
'
Set oDmgQuery = New DMgClient.DmgQuery
'
' initialize the DmgQuery object (this causes a Dmg
logon)
'
Screen.MousePointer = vbHourglass
lRC = oDmgQuery.Init()
If (lRC <> 0) Then
m_TKCerror = "oDmgQuery.Init() returned..." &
oDmgQuery.LastErrorDescription
m_TKCorigin = K_Proc & "..." & oDmgQuery.LastErrorSource
Err.Raise GENERAL_ERROR, m_TKCorigin, m_TKCerror, "",
ADVICE_NONE
End If
'
' open the document
'
lAction = 0
lRC = oDmgQuery.OpenDocument(sKey, sLabel, 0)
If (lRC <> 0) Then
m_TKCerror = "oDmgQuery.OpenDocument() returned..." &
oDmgQuery.LastErrorDescription
m_TKCorigin = K_Proc & "..." & oDmgQuery.LastErrorSource
Err.Raise GENERAL_ERROR, m_TKCorigin, m_TKCerror, "",
ADVICE_NONE
Else
MsgBox "oDmgQuery.OpenQuery() succeeded.", vbInformation
```

```
End If
'
'refresh the window that was opened by the previous call
'
lAction = 0 + WA_REFRESH
   lRC = oDmgQuery.OpenDocument(sKey, sLabel, lAction)
   If (lRC <> 0) Then
     m_TKCerror = "oDmgQuery.OpenDocument() returned..."
& oDmgQuery.LastErrorDescription
      m_TKCorigin = K_Proc & "..." &
oDmgQuery.LastErrorSource
      Err.Raise GENERAL_ERROR, m_TKCorigin, m_TKCerror,
"", ADVICE_NONE
   Else
      MsgBox "oDmgQuery.OpenQuery() succeeded.",
vbInformation
   End If
   '
   '
   '
'
Cleanup:
On Error GoTo 0
Set oDmgQuery = Nothing
Screen.MousePointer = vbDefault
Exit Sub
'-------------------------------------------------------
ErrorHandler:
Select Case Err.Number
Case EZP_STD_ERROR
m_TKCerror = Err.Description
Case GENERAL_ERROR
Case Else
```

```
m_TKCerror = "ERROR: " & Err.Number & " An unanticipated
error has occurred in " & K_Proc & "..." &
Err.Description
End Select
m_TKCadviceID = Err.HelpContext
m_TKCorigin = Err.Source
m_TKCmessage = m_TKCorigin & "..." & m_TKCerror
LogIt 5, m_TKCmessage
iOK = showMsg(vbInformation, m_TKCorigin, m_TKCerror,
m_TKCadvice)
Resume Cleanup
'-------------------------------------------------------
End Sub
```

# Specification Formats

**NOTE:** If an item contains a pipe-symbol, "|" (ASCII 124), replace it with an escape sequence "%7C"

## Project Key

Cabinet|Level|ProjectKeyString

### DmgAPI

Level = DMG_P_PROJECT_LEVELNUMBER

ProjectKeyString = DMG_P_PROJECT_KEYSTRING

### POAPI

Level = 1

ProjectKeyString = POProject .GetKeyString()

### Examples

Claim Properties|1|dbo.WF_Claims.Claim_Number = 1.000000 and dbo.WF_Claims.I_TaskID = 8002

Amergen Test|1|dbo.WF_Amergen.LName = 'Lobdale' and dbo.WF_Amergen.I_TaskID = 12003

## Document Key

Cabinet|DocID|MajorVersion|MinorVersion

### DmgAPI

DocID = DMG_P_DOCUMENT_ID
MajorVersion = DMG_P_DOCUMENT_MAJORVERSION
MinorVersion = DMG_P_DOCUMENT_MINORVERSION

### POAPI

DocID = PODocument.DocID
MajorVersion = PODocument.Version ' portion of string before "."
MinorVersion = PODocument.Version ' portion of string after "."

### Examples

Amergen|4|1|0
DCIG by Claim|407|1|0

## Folder Key

Cabinet|Level|FolderKeyString

### DmgAPI

Level = DMG_P_FOLDER_LEVELNUMBER
FolderKeyString = DMG_P_FOLDER_KEYSTRING

### POAPI

Level = POFolder.LevelNumber
FolderKeyString = POFolder.KeyString

### Examples

Amergen|1|DMGSamples.dbo.Amergen.LName = 'Bradley'
DCIG by Claim|4|DMGSamples.dbo.Vehicle.Vehicle_ID = 24.000000

## Query Specification (Format 0)

?Format=0?Cabinet=CabinetDesignation?[Folder=FolderCriterion]?[Document=DocumentCriterion] …

- **Format = 0** — ' this is a constant value. The Format item must be the first item in the Query Specification
- **Cabinet** = Cabinetname|Cabinet Level|Force Case Blind (Y|N) ' the Cabinet Designation Item must be the second item in the Query Specification
- **Folder** = Fieldname|Operator|Value|[Value2]' Folder criteria can be repeated as necessary
- **Document**= Fieldname|Operator|Value|[Value2]' Document criteria can be repeated as necessary
- **XDPCategory**=Category' this term must preceded any XDP terms

- **XDP**== Fieldname|Operator|Value|[Value2]' XDP criteria can be repeated as necessary; must be preceded by XDPCategory term
- **FolderOrder** = FieldName[|ASC][|DESC]' Folder Order By criteria can be repeated as necessary, sort order is optional and defaults to ASC
- **DocumentOrder** = FieldName[|ASC][|DESC]' Document Order By criteria can be repeated as necessary, sort order is optional and defaults to ASC
- **FullTextFilter** = Full Text Filter
- **IndicateEmptyFolders**=[TRUE] [FALSE]' value must be either True or False
- **Operators**
  - = equals
  - <>not equal to
  - < less than
  - > greater than
  - <= less than or equal to | on or before
  - >= greater than or equal to | on or after
  - -- between (requires Value2)
  - ** contains
  - * begins with

**NOTE:** In all cases if an item contains a question mark "?" (ASCII 63) it should be replaced with an escape sequence "%3F"

**NOTE:** The "Force Case Blind" value in the Cabinet Designation item constructs SQL clauses that are Case-Blind for Case-Sensitive databases. This is a very expensive operation for the database. This value should be set to "N" unless a case-blind filter is truly intended. For non-Case-Sensitive databases the value is overwritten to "N" regardless of its value.

- All items are delimited with a question mark. There should be no final trailing question mark.
- Folder and Document criteria can be repeated as necessary.
- The Field names are the field names as displayed in the Documanage Workstation and Documanage Administrator (they are not the fully qualified table column names)
- This specification format is currently restricted to one cabinet level.
- This format does not allow for the specification of "OR" terms.
- This syntax does not address specifying Variable Filters (i.e. Server-Side filters)

## Query Specification (Format 1)

?Format=1?Cabinet=CabinetDesignation?[FolderFilter=FolderFilter]?[DocumentFilter=DocumentFilter]?[FolderOrderBy=FolderOrderBy]?[DocumentOrderBy=DocumentOrderBy]?[XDPFilter=XDPFilter]?[FullTextFilter=FullTextFilter]?[IndicateEmptyFolders=True|False]

- **Format = 1** ' this is a constant value. The Format item must be the first item in the Query Specification
- **Cabinet**=Cabinetname|Cabinet Level|Force Case Blind (Y|N) [ ignored ] ' the Cabinet Designation Item must be the second item in the Query Specification. The Cabinet Level parameter is ignored. The Force Case Blind parameter is ignored.
- **FolderFilter**=SQL where clause with fully qualified folder filter
- **DocumentFilter**=SQL where clause with fully qualified document filter
- **FolderOrderBy**=SQL order by clause for Folder
- **DocumentOrderBy**=SQL order by clause for Document
- **XDPFilter**=Category|SQL where clause with fully qualified XDP Filter
- **FullTextFilter**=Full Text Filter
- **IndicateEmptyFolders**=[True] [False]

**NOTE:** In all cases if an item contains a question mark "?" (ASCII 63) it should be replaced with an escape sequence "%3F"

**NOTE:** The "Force Case Blind" value in the Cabinet Designation item is ignored as the designated filters are not manipulated in any way. If the filter syntax has a Force Case Blind form, this field can be set to "Y" for the sake of completeness. This is not necessary.

- All items are delimited with a question mark. There should be no final trailing question mark.

- The various items should not be repeated. If they are the last specified of the type will be used, without an error noted.
- This specification format allows for "OR" clauses in Folder and Document filters. This syntax does not address specifying Variable Filters (i.e. Server-Side filters)

## Examples

- FolderFilter = DMGSamples.dbo.Vehicle.Customer_ID = 137 Or DMGSamples.dbo.Vehicle.Customer_ID = 145
- DocumentFilter = dbo.OT_Docs.DocID = 4
- FolderOrderBy = DMGSamples.dbo.Amergen.FName
- DocumentOrderBy = Order By dbo.OT_Docs.Tag
- XDPFilter = CLAIM|dbo.OT_CLAIM.Accident_Cause LIKE '%test%' and dbo.OT_CLAIM.Fault LIKE '%yes%'
- FullTextFilter = See Microsoft documentation for syntax
- IndicateEmptyFolders = True

**NOTE:** The DocumentOrderBy clause requires the "Order By" phrase, whereas the FolderOrderBy does not and will throw an error if it's included. The DocumentOrderBy clause will not throw an error if it's not included, but will not include any documents in the query.

# COM Invocation Interface

Programmers can give Workstation users a way to select documents and pass them to a custom component using the COM Invocation Interface. The interface provides a menu option that instantiates a COM object and passes information about the current connection, cabinet, and the documents that users have selected. The interface can run the COM object in process or out-of-process.

## DocSelectionData object

When the user selects a COM Invocation menu item, the Workstation instantiates an object by calling CreateObject on the supplied ProgID. It then calls the .Init() method of the object passing it a DocSelectionData object.The properties of that object are as follows:

### General

- ***MenuCaption()  As String***
  *The caption of the menu option used to invoke the object.*

  The same object could be used by multiple menu options and change it's behavior accordingly).

- ***Version()  As String***
  The version of the interface; currently 1.0

- ***CurrentForm()  As Object***
  *The current form.*

  This allows an in-process component to use this form as the "parent" for any dialogs it posts. Thus they will follow the parent if minimized or context is switched to a different applications, and so on. This value will be invalid for out-of-process components.

## Connection Information

The component could use the existing connection if it is in-process and based on POAPI; otherwise it could use this information to establish a session.

- ***.hSession() As Long***
  *current session*
- ***.hCursor() As Long***
  *current cursor*
- ***.hFolder() As Long***
  *current folder hItem*
- ***.Cabinet() As String***
  *cabinet name*
- ***.FolderLabel() As String***
  *current folder label*
- ***.FolderKeyString() As String***
  *current folder keystring*

## Document Information

Information about the selected document(s).

- ***.TotalDocuments() As Long***
  *number of documents*
- ***.DocumentLabel(i) As String***
  *document label (array is 1-based)*
- ***.DocumentKeyString(i) As String***
  *document keystring (array is 1-based)*
- ***.DocumentSpecifier(i) As String***
  *document specifier (array is 1-based)*
- ***.DocumenthItem(i) As Long***
  *hItem for document (array is 1-based)*

# Com Object

Various COM objects that support the COM Invocation interface can be defined. Depending upon how they are compiled, they can run in-process or out-of-process. They could conceivably run on a separate machine through DCOM. If the object uses POAPI and is running in-process, it can use the existing session, cursor and hItems. In this case, the component would act as if it were part of the Workstation. Out-of-process components or those built on DmgAPI would need to establish a session and use the folderkey to get a cursor, or use the docSpecifiers to get document hItems. A COM object supporting the COM Invocation interface has to expose the following properties and methods:

- ***.Init(oDocSelectionData as DmgClient.DocSelectionData) As Long***
  *The passed parameter could also be defined "As Object" and then would be late bound. The method should return a zero for success, greater than zero for a user cancellation, less than zero for an error.*

- ***.Version() As String***
  *The version of the COM Invocation interface that the component supports, currently 1.0.*

- ***.RefreshContent() As Boolean***
  *A flag indicating whether the Workstation should refresh its UI content.*

  For example, the component may have altered folder contents, document attributes, and so on.

- ***.LastErrorDescription() As String***
  *A description of the last error encountered.*

  The workstation reports the error to its log and to the user if the .Init method returns a negative error code.

- ***.LastErrorSource() As String***
  *Source of the last error.*

- ***.LastErrorNumber() As Long***
  *Number of the last error.*

- ***.LastErrorAdvice() As String***
  *Suggestion to the user regarding the error.*

# DmgComObjects.ini File

Programmers can give Workstation users a way to select documents and pass them to a custom component using the COM Invocation Interface. The

interface provides a menu option that instantiates a COM object and passes information about the current connection, cabinet, and the documents that users have selected.

The DmgComObjects.ini filet adds one or more menu items to the Documanage Workstation Document Menu.The following paragraphs describe the entries in the file and include a name/value table, and a sample file entry.

# Version

The Version section contains the version information for the COM Invocation Interface.

| Name | Possible Values | Default | Description |
|---|---|---|---|
| Version | 1.0 | "1.0” | Version information. NOTE this section must be present with a value of 1.0 |

### Sample ini file entry:

```
[Version]
Version=1.0
```

# Document COM Objects

This section contains a count of the number of COM objects that can be defined. In the future, COM objects on the Folder menu or some other extension might be supported

| Name | Possible Values | Default | Description |
|---|---|---|---|
| Count | 2 | No Default | The number of Document COM objects defined. |

### Sample ini file entry:

```
[DocumentCOMObjects]
Count=2
```

# Document COM Object

This section contains information for the nth COM object. One of these sections exists for each COM object.

| Name | Possible Values | Default | Description |
|---|---|---|---|
| ProgID | TestDmgCOM.TestObject | No Default | The ProgID of the object. The Workstation instantiates the object with a CreateObject call on this ProgID |
| MenuCaption | Test C&OM Object | If a menu caption is not provided, the ProgID is used. | The menu caption added to the Document Menu; accelerator keys may be defined but should not be in conflict with existing accelerator keys on the menu. |

| Name | Possible Values | Default | Description |
|---|---|---|---|
| ShowOnCabinet | True, False | False | A flag that determines whether the menu option appears on the Cabinet Window |
| ShowOnProject | True, False | False | A flag that determines whether the menu option appears on the ActiveProject window. |
| EnableOnMultiselect | True, False | True | A flag whether the menu option should be enabled when multiple documents are selected. |
| EnableMask | 0, or mask setting | 0 | A bit mask that enables the menu options corresponding to user authorities (download, view, edit, and so on) . A value of zero enables the menu option whenever a document is selected. |

## Sample DocCOMObject.ini file entry:

```
[DocCOMObject_1]
ProgID=TestDmgCOM.TestObject
MenuCaption=Test C&OM Object
ShowOnCabinet=True
ShowOnProject=True
EnableOnMultiselect=True
EnableMask=0
```

# Enable Mask

The EnableMask value reflects which of its bits should be ON to enable different menu options. The bits are evaluated based upon the user authorities for the last selected document.

**NOTE:** All of the specified Enable Mask bits must be ON to enable the menu item.

An EnableMask value of zero indicates that the menu option should be enabled whenever a document is selected. The DocSelectionData object exposes the following enumeration, which holds bit values that control the enabled state of the COM object menu item(s).

```
Public Enum COM_ENABLE_BITS
   CEB_ISCHECKEDOUTBYCURRENTUSER = 2
   CEB_CHECKOUT = 4
   CEB_VIEW = 8
   CEB_DOWNLOAD = 16
   CEB_EDITCONTENTS = 32
   CEB_EDITDATA = 64
   CEB_CHECKINSAME = 128
   CEB_CHECKINMINOR = 256
   CEB_CHECKINMAJOR = 512
   CEB_DELETE = 1024
   CEB_VIEWHISTORY = 2048
   CEB_MOVE = 4096
   CEB_COPY = 8192
End Enum
```

**Enable Mask Constant Definitions**

| Constant | Definition |
|---|---|
| CEB_ISCHECKEDOUTBYCURRENTUSER | The document is checked out by the current user. |
| CEB_CHECKOUT | The document can be checked out. |

| Constant | Definition |
|---|---|
| CEB_VIEW | The document can be viewed. |
| CEB_DOWNLOAD | The document file can be downloaded. |
| CEB_EDITCONTENTS | The user can edit the document file. |
| CEB_EDITDATA | The user can edit the document information. |
| CEB_CHECKINSAME | The document can be checked in as the same version. NOTE that the document must be checked out for this authority to evaluate to true. |
| CEB_CHECKINMINOR | The document can be checked in as a minor version. NOTE that the document must be checked out for this authority to evaluate to true. |
| CEB_CHECKINMAJOR | The document can be checked in as a major version. NOTE that the document must be checked out for this authority to evaluate to true. |
| CEB_DELETE | The document can be deleted. |
| CEB_VIEWHISTORY | The version history of the document can be viewed. |
| CEB_MOVE | The document can be moved to another folder. |
| CEB_COPY | The document can be copied to another folder. |

## Enable Mask Example

- To specify that a COM menu option should be enabled if the user has the authority to download and edit the document file contents, the EnableMask should be set to 48 (16 + 32).
- To specify that a COM menu option should be enabled if the user has the authority to copy and delete a document, the EnableMask should be set to 9216 (1024 + 1892).

- To specify that a COM menu option should be enabled if the user has the authority to checkout a document, edit the document information, and download and edit the document file, the EnableMask should be set to 116 (4 + 16 + 32 + 64).

- To specify that a COM menu option should be enabled whenever a document is selected, the EnableMask should be set to zero.

# Base Functionality COM Replacement Interface

A COM component can replace the existing basic functions of the Workstation. The following .ini file entries allow you to replace these functions with custom COM components.

- Edit
- View
- Scan
- Import
- Export
- Mail
- Print

The Workstation reads the DmgCOMObjects.ini file and determines if a replacement is defined for any of these base document procedures. If present the Workstation invokes the COM component designated in the .ini file rather than the procedure ordinarily called by the Workstation. It passes a

DocSelectionData object as discussed above. The technical details of the COM interface are identical to that discussed above.

## Edit

Edit replaces the Document Edit procedure

| Name | Possible Values | Default | Description |
|---|---|---|---|
| ProgID | COM ProgID of component to replace Document Edit Procedure | No Default | The ProgID of the object; the Workstation will instantiate the object with a CreateObject call on this ProgID |
| ShowOnCabinet | True\|False | False | A flag whether the procedure should be replaced on the Cabinet Window. |
| ShowOnProject | True\|False | False | A flag whether the procedure should be replaced on the Project Window. |
| EnableOnMultiselect | True\|False | False | A flag whether the procedure should be replaced on the Cabinet Window. |
| MenuCaption | blank\|Replacement Menu Caption | "" which preserves the existing Menu Caption. | replacement Menu Caption for the procedure. NOTE it is recommended this item be left blank or omitted |

| Name | Possible Values | Default | Description |
|---|---|---|---|
| ToolTip | blank\|Replacement ToolTip | "" which preserves the existing ToolTip | replacement ToolTip for the procedure button on the Document Toolbar. NOTE that this is ignored on the ActiveProject Window which does not have a Document Toolbar |

## View

View replaces the Document View procedure.

| Name | Possible Values | Default | Description |
|---|---|---|---|
| ProgID | COM ProgID of component to replace Document View Procedure | No Default | The ProgID of the object; the Workstation will instantiate the object with a CreateObject call on this ProgID |
| ShowOnCabinet | True\|False | False | A flag whether the procedure should be replaced on the Cabinet Window. |
| ShowOnProject | True\|False | False | A flag whether the procedure should be replaced on the Project Window. |
| EnableOnMultiselect | True\|False | False | A flag whether the procedure should be replaced on the Cabinet Window. |

| Name | Possible Values | Default | Description |
|---|---|---|---|
| MenuCaption | blank\|Replacement Menu Caption | "" which preserves the existing Menu Caption. | replacement Menu Caption for the procedure. NOTE it is recommended this item be left blank or omitted |
| ToolTip | blank\|Replacement ToolTip | "" which preserves the existing ToolTip | replacement ToolTip for the procedure button on the Document Toolbar. NOTE that this is ignored on the ActiveProject Window which does not have a Document Toolbar |

## Scan

Scan replaces the Scan procedure.

| Name | Possible Values | Default | Description |
|---|---|---|---|
| ProgID | COM ProgID of component to replace Scan Procedure | No Default | The ProgID of the object; the Workstation will instantiate the object with a CreateObject call on this ProgID |
| ShowOnCabinet | True\|False | False | A flag whether the procedure should be replaced on the Cabinet Window. |
| ShowOnProject | True\|False | False | A flag whether the procedure should be replaced on the Project Window. |
| EnableOnMultiselect | NA | NA | Does not apply. Can be present but will be ignored. |

| Name | Possible Values | Default | Description |
|---|---|---|---|
| MenuCaption | blank\|Replacement Menu Caption | "" which preserves the existing Menu Caption. | replacement Menu Caption for the procedure. NOTE it is recommended this item be left blank or omitted |
| ToolTip | NA | NA | Does not apply. Can be present but will be ignored. |

## Import

Import replaces the Document Import procedure.

| Name | Possible Values | Default | Description |
|---|---|---|---|
| ProgID | COM ProgID of component to replace Document Import Procedure | No Default | The ProgID of the object; the Workstation will instantiate the object with a CreateObject call on this ProgID |
| ShowOnCabinet | True\|False | False | A flag whether the procedure should be replaced on the Cabinet Window. |
| ShowOnProject | True\|False | False | A flag whether the procedure should be replaced on the Project Window. |
| EnableOnMultiselect | NA | NA | Does not apply. Can be present but will be ignored. |
| MenuCaption | blank\|Replacement Menu Caption | "" which preserves the existing Menu Caption. | replacement Menu Caption for the procedure. NOTE it is recommended this item be left blank or omitted |

| Name | Possible Values | Default | Description |
|---|---|---|---|
| ToolTip | blank\|Replacement ToolTip | "" which preserves the existing ToolTip | replacement ToolTip for the procedure button on the Document Toolbar. NOTE that this is ignored on the ActiveProject Window which does not have a Document Toolbar |

## Export

Export replaces the Document Export procedure.

| Name | Possible Values | Default | Description |
|---|---|---|---|
| ProgID | COM ProgID of component to replace Document Export Procedure | No Default | The ProgID of the object; the Workstation will instantiate the object with a CreateObject call on this ProgID |
| ShowOnCabinet | True\|False | False | A flag whether the procedure should be replaced on the Cabinet Window. |
| ShowOnProject | True\|False | False | A flag whether the procedure should be replaced on the Project Window. |
| EnableOnMultiselect | True\|False | False | A flag whether the procedure should be replaced on the Cabinet Window. |

| Name | Possible Values | Default | Description |
|---|---|---|---|
| MenuCaption | blank\|Replacement Menu Caption | "" which preserves the existing Menu Caption. | replacement Menu Caption for the procedure. NOTE it is recommended this item be left blank or omitted |
| ToolTip | blank\|Replacement ToolTip | "" which preserves the existing ToolTip | replacement ToolTip for the procedure button on the Document Toolbar. NOTE that this is ignored on the ActiveProject Window which does not have a Document Toolbar |

## Mail

Mail replaces the Document Mail procedure.

| Name | Possible Values | Default | Description |
|---|---|---|---|
| ProgID | COM ProgID of component to replace Document Mail Procedure | No Default | The ProgID of the object; the Workstation will instantiate the object with a CreateObject call on this ProgID |
| ShowOnCabinet | True\|False | False | A flag whether the procedure should be replaced on the Cabinet Window. |
| ShowOnProject | True\|False | False | A flag whether the procedure should be replaced on the Project Window. |

| Name | Possible Values | Default | Description |
|---|---|---|---|
| EnableOnMultiselect | True\|False | False | A flag whether the procedure should be replaced on the Cabinet Window. |
| MenuCaption | blank\|Replacement Menu Caption | "" which preserves the existing Menu Caption. | replacement Menu Caption for the procedure. NOTE it is recommended this item be left blank or omitted |
| ToolTip | blank\|Replacement ToolTip | "" which preserves the existing ToolTip | replacement ToolTip for the procedure button on the Document Toolbar. NOTE that this is ignored on the ActiveProject Window which does not have a Document Toolbar |

## Print

Print replaces the Document Print procedure.

| Name | Possible Values | Default | Description |
|---|---|---|---|
| ProgID | COM ProgID of component to replace Document Print Procedure | No Default | The ProgID of the object; the Workstation will instantiate the object with a CreateObject call on this ProgID |
| ShowOnCabinet | True\|False | False | A flag whether the procedure should be replaced on the Cabinet Window. |

| Name | Possible Values | Default | Description |
|---|---|---|---|
| ShowOnProject | True\|False | False | A flag whether the procedure should be replaced on the Project Window. |
| EnableOnMultiselect | True\|False | False | A flag whether the procedure should be replaced on the Cabinet Window. |
| MenuCaption | blank\|Replacement Menu Caption | "" which preserves the existing Menu Caption. | replacement Menu Caption for the procedure. NOTE it is recommended this item be left blank or omitted |
| ToolTip | blank\|Replacement ToolTip | "" which preserves the existing ToolTip | replacement ToolTip for the procedure button on the Document Toolbar. NOTE that this is ignored on the ActiveProject Window which does not have a Document Toolbar |

## Some Considerations

The User Interface does not change, except for Document menu items and Document Toolbar button ToolTips that can be overridden. The enabled/ disabled state of the items follows the same evaluation of user authorities as the base functionality except that most of these procedures are currently enabled only when a single document is selected, as follows:

| Procedure | Selection |
|---|---|
| **Edit** | single select only |
| **View** | single select only |

| Procedure | Selection |
|---|---|
| **Scan** | NA |
| **Import** | NA |
| **Export** | multiselect OK |
| **Mail** | multiselect OK |
| **Print** | multiselect OK |

The syntax allows this to be overridden with the EnableOnMultiselect flag.

# Sample DmgCOMObjects.ini file

A typical DmgCOMObjects.ini file is shown here.

```
[Version]
Version=1.0

[DocumentCOMObjects]
Count=2

[DocCOMObject_1]
ProgID=TestDmgCOM.TestObject
MenuCaption=Test C&OM Object
ShowOnCabinet=True
ShowOnProject=True
EnableOnMultiselect=True
EnableMask=0

[DocCOMObject_2]
ProgID=TestDmgCOMoop.TestObject
MenuCaption=Test CO&M Object Out-of-Process
ShowOnCabinet=True
ShowOnProject=True
EnableOnMultiselect=False
EnableMask=0

[Mail]
ProgID=TestDmgCOM.ReplaceMail
ShowOnCabinet =True
ShowOnProject =True
EnableOnMultiselect=True
```

```
MenuCaption=Mail document(s) via specialized component…
ToolTip=Mail document(s) via Acme SuperMail

[Print]
ProgID= TestDmgCOM.ReplacePrint
ShowOnCabinet =True
ShowOnProject = False
EnableOnMultiselect= False
MenuCaption=
ToolTip=
```

# Index

## V

# Index