HotDocs API

Table of Contents

Help Topics for the HotDocs Application Programming Interface (API) 1

Organization of the Help File 1

Find Topics in the Help File 1

Other Help Resources 1

Copyright Information 3

Copyright 3

Warranty Information 3

Government Use 3

Trademark Information 3

New Features and Enhancements 5

New and Enhanced Features in the HotDocs 11 API 5

HotDocs 11 5

New and Enhanced Features in the HotDocs 10 API 7

HotDocs 10 7

HotDocs 10.1 7

New and Enhanced Features in the HotDocs 2009 API 7

HotDocs 2009 7

New and Enhanced Features in the HotDocs 2008 API 7

HotDocs 2008 7

New and Enhanced Features in the HotDocs 2007 API 8

HotDocs 2007 8

New and Enhanced Features in the HotDocs 2006 API 8

HotDocs 2006 8

New and Enhanced Features in the HotDocs 2005 API 9

HotDocs 2005 9

HotDocs 2005 SP2 9

New and Enhanced Features in the HotDocs 6.2 API 9

HotDocs 6.2 SP1 9

New and Enhanced Features in the HotDocs 6.1 API 10

HotDocs 6.1 10

HotDocs 6.1 SP1 10

About the HotDocs API 11

What is the HotDocs API? 11

Using Command-Line Options 13

Introduction: Command-Line Options 13

To use command-line options when starting HotDocs 13

To use command-line options when using a shortcut to start HotDocs 13

To use command-line options within ASSEMBLE instructions 14

To add command-line options to a library item 14

Full List of Command-Line Options 14

Installation Switches 14

Application Control Switches 15

Template Type Switches 16

Answer Initialization Switches 16

Interview Behavior Switches 17

Answer Disposition Switches 18

Document Disposition Switches 19

Automator/Filler Command Line Switches 19
Use Command-Line Options When Starting HotDocs 20
To use command-line options when starting HotDocs 20
Use Command-Line Options when Using a Shortcut to Start HotDocs 21
To use command-line options when using a shortcut to start HotDocs 21
Use Command-Line Options within ASSEMBLE Instructions 21
To use command-line options within ASSEMBLE instructions 21
Use Command-Line Options at File Properties 21
To add command-line options to a library item 21
Answer File 22
Answer Summary 22
Clause Name 23
Default Answer File 23
Discard Answers 23
Don't Brag 24
Edit Template 24
Exit HotDocs 24
Finish Interview Action 25
Hide Library 25
HotDocs Auto-Assemble File 25
HotDocs Auto-Install File 26
HotDocs Model 26
Installation File 26
Interview Scope 26

Keep Interview Group 27
Library File 27
Lock Answer File 28
Lock Library 28
New Answer File 28
No Assembly Window 29
No Exit 29
No Interview 29
Output File 30
Overlay Answer File 30
Paper Size 31
Paper Tray 31
Print 32
Print Answers Only 32
Print Both 33
Print Copies 33
Print Duplex 33
Print Form Only 34
Print Without Dialogs 34
Question Summary 34
Register 35
Save Answers 35
Save Answers Prompt 35
Send to Plugin 36

Send to Word Processor 36
Show Library 37
Start Interview Group 37
Suggest Save 37
Suggest Save New 38
Suppress Installation 38
Suppress Unanswered Warning 38
Template File 39
Unregister 39
COM API 41
About the HotDocs COM API 41
About the HotDocs COM API 41
How do I use the HotDocs COM API in .NET? 42
How do I use the HotDocs Variable Mapping API? 42
Understand COM Events 44
How do I program a COM Event in Visual C#? 47
Enumerations 48
DependencyType Enumeration 48
HDAFFORMAT Enumeration 49
HDAIMENU Enumeration 49
HDANSWERUPLOADFORMAT Enumeration 50
HDASSEMBLYSTATUS Enumeration 50
HDAUI Enumeration 51
HDDirectory Enumeration 57

HDLIMENU Enumeration ... 58
HDLUI Enumeration ... 58
HDMappingBackfill Enumeration ... 62
HDOUTPUTTYPE Enumeration ... 63
HDPRODUCTFLAVOR Enumeration ... 63
HDServerFileType Enumeration ... 64
HDVARTYPE Enumeration ... 64
HotDocs.Answer Object ... 65
HotDocs.Answer Object ... 65
Answer.AddMultipleChoiceValue Method ... 68
Answer.ClearAskedFlag Method ... 69
Answer.Create Method ... 70
Answer.GetRepeatCount Method ... 71
Answer.GetRepeatIndex Method ... 72
Answer.IsMultipleChoiceValueSet Method ... 72
Answer.IterateValues Method ... 73
Answer.SetRepeatIndex Method ... 75
Answer.Application Property ... 76
Answer.Name Property ... 77
Answer.RepeatCount Property ... 77
Answer.Type Property ... 77
Answer.Unanswered Property ... 78
Answer.Value Property ... 78
Answer.OnValueFoundEvent Event ... 80

HotDocs.AnswerCollection Object 82
HotDocs.AnswerCollection Object 82
AnswerCollection.Add Method 85
AnswerCollection.Close Method 86
AnswerCollection.Create Method 87
AnswerCollection.Item Method 87
AnswerCollection.Overlay Method 88
AnswerCollection.Save Method 89
AnswerCollection.UploadAnswerCollection Method 90
AnswerCollection.Application Property 90
AnswerCollection.Count Property 90
AnswerCollection.DefaultAnswerFile Property 91
AnswerCollection.Description Property 91
AnswerCollection.FileFormat Property 92
AnswerCollection.FileName Property 93
AnswerCollection.Modified Property 94
AnswerCollection.Title Property 95
AnswerCollection.XML Property 95
HotDocs.Application Object 96
HotDocs.Application Object 96
Application.AddUserMenuItem Method 101
Application.AddUserMenuItem2 Method 102
Application.ConvertModelToTemplate Method 104
Application.ConvertTemplateToModel Method 105

Application.CreateTemplatePackage Method 105
Application.DeleteUserMenuItem Method 106
Application.getDefaultPath Method 107
Application.GetHotDocsSetting Method 108
Application.OpenLibrary Method 109
Application.PrintDocument Method 110
Application.PublishOnlineFiles Method 111
Application.PublishOnlineFiles2 Method 112
Application.ResolveReferencePath Method 113
Application.RetrieveUrlFile Method 113
Application.SaveDocAsPDF Method 114
Application.SelectMultipleTemplates Method 115
Application.SelectMultipleTemplates2 Method 117
Application.SelectTemplate Method 118
Application.SelectTemplate2 Method 120
Application.SendToWordProcessor Method 121
Application.SetUserInterfaceItem Method 122
Application.ActiveAssembly Property 122
Application.Assemblies Property 123
Application.AssemblyQueueVisible Property 124
Application.CanAssembleAll Property 125
Application.CanEditTemplates Property 125
Application.CommandLine Property 125
Application.CurrentLibraryPath Property 126

Application.Flavor Property 126
Application.Hwnd Property 127
Application.Plugins Property 128
Application.Version Property 128
Application.Visible Property 129
Application.AssemblyCompleteEvent Event 129
Application.OnAssemblyCompleteEvent Event 130
Application.OnAssemblyStartEvent Event 131
Application.OnErrorEvent Event 131
Application.OnLibraryInterfaceCloseEvent Event 131
Application.OnLibraryOpenEvent Event 132
Application.OnTemplateSelectedEvent Event 132
Application.OnUserInterfaceEvent Event 132
Application.OnUserMenuItemClickedEvent Event 133
HotDocs.Assembly Object 133
HotDocs.Assembly Object 133
Assembly.AddUserMenuItem Method 138
Assembly.DeleteUserMenuItem Method 139
Assembly.GetSaveAsExtDlg Method 140
Assembly.LocalBrowseDlg Method 141
Assembly.OpenAnswerFileDlg Method 141
Assembly.SelectOpenAnswerFileDlg Method 142
Assembly.SendToWordProcessor Method 143
Assembly.SetUserInterfaceItem Method 143

Assembly.UseAnswerFile Method 144
Assembly.AnswerCollection Property 144
Assembly.AnswerSummaryPath Property 145
Assembly.Application Property 145
Assembly.AssemblyHandle Property 145
Assembly.CommandLine Property 146
Assembly.DocumentPath Property 146
Assembly.Hwnd Property 147
Assembly.KeepInQueue Property 147
Assembly.Map Property 147
Assembly.PrintwhenComplete Property 148
Assembly.PromptToSaveDocument Property 148
Assembly.QuestionSummaryPath Property 148
Assembly.ShowAnswerFileDialog Property 149
Assembly.Status Property 149
Assembly.SuppressUnansweredWarning Property 150
Assembly.TemplateDesc Property 150
Assembly.TemplatePath Property 150
Assembly.TemplateTitle Property 151
Assembly.Visible Property 151
Assembly.OnAssemblyCompleteEvent Event 151
Assembly.OnAssemblyStartEvent Event 151
Assembly.OnCanOpenFile Event 152
Assembly.OnCloseAssemblyInterfaceEvent Event 152

Assembly.OnErrorEvent Event 153
Assembly.OnFileOpen Event 153
Assembly.OnFileSave Event 154
Assembly.OnFileSelectEvent Event 154
Assembly.OnGetAnswerFileDisplayName Event 155
Assembly.OnGetMRUInfo Event 155
Assembly.OnNeedAnswerEvent Event 156
Assembly.OnPostCloseAnswerFile Event 156
Assembly.OnPostSaveDocumentEvent Event 157
Assembly.OnPreCloseAnswerFile Event 157
Assembly.OnPreSaveDocumentEvent Event 158
Assembly.OnUserInterfaceEvent Event 158
Assembly.OnUserMenuItemClickedEvent Event 159
Assembly.PostSaveAnswersEvent Event 159
Assembly.PreSaveAnswersEvent Event 160
HotDocs.AssemblyCollectionClass Object 160
HotDocs.AssemblyCollectionClass Object 160
AssemblyCollectionClass.Add Method 162
AssemblyCollectionClass.AddToQueue Method 162
AssemblyCollectionClass.Clear Method 163
AssemblyCollectionClass.FindByHandle Method 163
AssemblyCollectionClass.Insert Method 164
AssemblyCollectionClass.Item Method 164
AssemblyCollectionClass.Move Method 164

AssemblyCollectionClass.Remove Method 165
AssemblyCollectionClass.Application Property 165
AssemblyCollectionClass.Count Property 166
HotDocs.Component Object 166
HotDocs.Component Object 166
Component.DisplayEditor Method 167
Component.Application Property 168
Component.DBName Property 168
Component.DialogName Property 169
Component.HelpText Property 170
Component.Name Property 171
Component.Prompt Property 171
Component.Properties Property 172
Component.Type Property 173
Component.Title Property 173
HotDocs.ComponentCollection Object 174
HotDocs.ComponentCollection Object 174
ComponentCollection.Create Method 176
ComponentCollection.CreateComponent Method 177
ComponentCollection.CreateVariable Method 178
ComponentCollection.Item Method 179
ComponentCollection.Open Method 180
ComponentCollection.OpenBase Method 180
ComponentCollection.OpenForEdit Method 181

ComponentCollection.Application Property 181
ComponentCollection.Count Property 182
ComponentCollection.FileName Property 183
ComponentCollection.OnlyVariables Property 183
ComponentCollection.ReadOnly Property 183
HotDocs.ComponentProperties Object 184
HotDocs.ComponentProperties Object 184
ComponentProperties.Add Method 185
ComponentProperties.Item Method 185
ComponentProperties.Count Property 186
HotDocs.ComponentProperty Object 186
HotDocs.ComponentProperty Object 186
ComponentProperty.Name Property 187
ComponentProperty.ReadOnly Property 187
ComponentProperty.Value Property 188
ComponentProperty.VariantType Property 188
HotDocs.Dependency Object 188
HotDocs.Dependency Object 189
Dependency.Dependencies Property 189
Dependency.DependencyType Property 190
Dependency.Target Property 190
HotDocs.DependencyCollection Object 190
HotDocs.DependencyCollection Object 190
DependencyCollection.GetEnumerator Method 191

DependencyCollection.Item Method 191
DependencyCollection.Count Property 192
HotDocs.Icon Object 192
HotDocs.Icon Object 192
Icon.LoadBitmap Method 193
Icon.LoadIcon Method 193
Icon.HBITMAP Property 194
Icon.HICON Property 194
Icon.index Property 195
Icon.maskColor Property 195
HotDocs.Library Object 195
HotDocs.Library Object 195
Library.Close Method 197
Library.New Method 197
Library.Open Method 197
Library.Save Method 198
Library.Application Property 198
Library.Description Property 199
Library.MainFolder Property 199
Library.Redraw Property 199
Library.Title Property 200
HotDocs.LibraryEntity Object 200
HotDocs.LibraryEntity Object 200
LibraryEntity.AddFolder Method 202

LibraryEntity.AddTemplate Method 203
LibraryEntity.Item Method 203
LibraryEntity.Remove Method 204
LibraryEntity.Application Property 204
LibraryEntity.Count Property 204
LibraryEntity.Description Property 204
LibraryEntity.ID Property 205
LibraryEntity.IsFolder Property 205
LibraryEntity.OverlayIndex Property 205
LibraryEntity.Parent Property 205
LibraryEntity.TemplateFullPath Property 206
LibraryEntity.TemplatePath Property 206
LibraryEntity.Title Property 206
HotDocs.Plugin Object 207
HotDocs.Plugin Object 207
Plugin.CLSID Property 208
Plugin.Description Property 208
Plugin.priorityClass Property 209
Plugin.priorityIndex Property 210
HotDocs.PluginsClass Object 211
HotDocs.PluginsClass Object 211
PluginsClass.Item Method 212
PluginsClass.Register Method 212
PluginsClass.Unregister Method 213

PluginsClass.Count Property 214
HotDocs.TemplateInfo Object 215
HotDocs.TemplateInfo Object 215
TemplateInfo.Close Method 216
TemplateInfo.Open Method 216
TemplateInfo.ComponentCollection Property 217
TemplateInfo.Dependencies Property 217
TemplateInfo.EffectiveComponentFile Property 217
TemplateInfo.PointedToComponentFile Property 218
TemplateInfo.PrimaryComponentFile Property 218
TemplateInfo.RecursiveDependencies Property 218
HotDocs.VarMap Object 218
HotDocs.VarMap Object 219
VarMap.HDVariablesAdd Method 222
VarMap.HDVariablesItem Method 222
VarMap.MappingAdd Method 223
VarMap.MappingAdd2 Method 223
VarMap.MappingAddEx2 Method 224
VarMap.MappingItem Method 224
VarMap.MappingItem2 Method 225
VarMap.MappingRemove Method 226
VarMap.OpenComponentFile Method 226
VarMap.OpenMapFile Method 226
VarMap.SaveMapFile Method 227

VarMap.ShowUserInterface Method 227
VarMap.SourceNamesAdd Method 228
VarMap.SourceNamesAdd2 Method 229
VarMap.SourceNamesItem Method 229
VarMap.SourceNamesItem2 Method 230
VarMap.SourceNamesRemove Method 231
VarMap.Application Property 231
VarMap.DefaultBackfill Property 232
VarMap.HDVariablesCount Property 232
VarMap.MappingCount Property 232
VarMap.MapTextAndMultipleChoice Property 233
VarMap.ShowBackfillColumn Property 233
VarMap.SourceNamesCount Property 233
Answer Source API 235
About the HotDocs Answer Source API 235
What is an answer source integration? 235
Answer Source Integration Example 235
How do I create an answer source integration? 241
HotDocs Answer Source API 243
HotDocs Answer Source API 244
BeginUpdateBatch Function 246
ChooseMultipleRecords Function 246
ChooseRecord Function 247
CloseRecord Function 248

CommitUpdates Function 249
EndUpdateBatch Function 250
GetChosenRecords Function 250
GetField Function 251
GetFieldW Function 252
GetFieldAccess Function 254
GetFieldAccessW Function 255
GetFieldName Function 256
GetFieldNameW Function 257
IsAvailable Function 258
OpenRecord Function 259
SetField Function 260
SetFieldW Function 261
SupportsBackfill Function 263
Plug-in API 265
About the HotDocs Plug-in API 265
What is a HotDocs plug-in? 265
How do I create a HotDocs plug-in? 266
How do I create a HotDocs plug-in using Visual C#? 267
ILibraryWindowContextMenuExtension Interface 270
ILibraryWindowContextMenuExtension Interface 270
ContextCommand Function 272
ContextGetMenuEntry Function 273
ContextGetMenuTitle Function 274

ContextInitialize Function 275
ContextLibraryInitialized Function 276
ILibraryWindowFileHandlerExtension Interface 276
ILibraryWindowFileHandlerExtension Interface 276
Assemble Function 279
Edit Function 280
Initialize Function 280
LibraryInitialized Function 281
RegisterFileType Function 282
ILibraryWindowIconProvider Interface 283
ILibraryWindowIconProvider Interface 283
Initialize Function 285
LibraryInitialized Function 285
UpdateLibraryEntry Function 286
ILibraryWindowMenuExtension Interface 286
ILibraryWindowMenuExtension Interface 286
Command Function 289
DisplayMenuInitialize Function 290
GetMenuEntry Function 291
GetMenuTitle Function 292
Initialize Function 292
LibraryInitialized Function 293
IOutputPlugin Interface 294
IOutputPlugin Interface 294

DocumentAssembled Function 297
GetPlugInfo Function 298
Initialize Function 298
LibraryInitialized Function 299
CommandId Property 300
IPluginPreferences Interface 300
IPluginPreferences Interface 300
IPluginPreferences Function 302
Contact HotDocs Sales and Support 303
HotDocs Technical Support 303
Outside the European Union: 303
Inside the European Union: 303
HotDocs Sales Support 303
Outside the European Union: 304
Inside the European Union: 304
Documentation Feedback 305
Glossary 307
Index 323

Help Topics for the HotDocs Application Programming Interface (API)

This Help file is a technical guide to integrating HotDocs with other software applications. It is written with the assumption that you have experience in software development and an understanding of a high-level programming language such as Visual Basic, C++, or C#. Because the type of application you integrate with HotDocs and the extent of that integration will vary from project to project, this file provides only broad guidelines on how to accomplish your task. You must adapt this information to the requirements of your project.

Organization of the Help File

Content within the help file is categorized into several areas:

1. **New Features and Enhancements:** What's new in HotDocs 11 (and prior versions)
2. **About HotDocs API:** Conceptual information about the role HotDocs API plays.
3. **Using Command Line Option:** A reference guide to the Command Line Options.
4. **COM API:** A reference guide to the COM APIs, arranged by class.
5. **Answer Source API:** A reference guide to the Answer Source APIs.
6. **Plug-in API:** A reference guide to the Plug-in API.

Find Topics in the Help File

To use the help file, you have several options:

- Click the **Index** tab to view a listing of all the topics in the help file, referenced by index keyword.
- Enter a search term or phrase in the **Search** box and click **Search** to view topics that contain your search phrase. If the search phrase is found, a **Search Results** list is displayed, showing the different topics that meet your criteria.

See Form a Help Search Query for detailed instructions on creating a search query.

Other Help Resources

There are four buttons in the navigation bar where you can access the following options

- 

Hide: Hide the left hand pane, the button then changes to the 

Show button.

- 

Back: Go back to the previous page.
- 

Print: View the **Print** dialog box to print a copy of the current page.
- 

Options: Show the options drop-down menu where you can choose from: **Hide Tabs**, **Back**, **Forward**, **Home**, **Stop**, **Refresh**, **Internet Options**, **Print**, and **Search Highlights Off**.

For additional help using HotDocs, you can view the online HotDocs Wiki. See http://wiki.hotdocs.com.

When you view a topic in a search results list, the help system highlights your search terms. If you want to remove these highlights, select **Search Highlights Off** from the **Options** menu and refresh the page. To turn them back on again select **Search Highlights On** and refresh the page..

Copyright Information

Copyright

Warranty Information

HotDocs makes no representations or warranties with respect to the contents or use of this product and specifically disclaims any express or implied warranties of merchantability or fitness for any particular purpose. Information in this document is subject to change without notice and does not represent a commitment on the part of HotDocs.

Government Use

Use, duplication, or disclosure by the Federal Government is subject to restrictions as set forth in FAR clauses 52.227--14, "Rights in Data--General"; 52.227--19, "Commercial Computer Software--Restricted Rights"; and subdivision (c)(1)(ii) of the Rights in Technical Data and Computer Software clause DFAR 252.227--7013; and the limitations set forth in the standard commercial license agreement for this software. Unpublished rights are reserved under the copyright laws of the United States.

Trademark Information

HotDocs is a registered trademark of HotDocs Limited. Other product names may be trademarks or registered trademarks of their respective companies.

New Features and Enhancements

New and Enhanced Features in the HotDocs 11 API

HotDocs 11

For HotDocs 11 this help file has undergone multiple format changes, see Get Help Using HotDocs Desktop API for information on navigating the new interface.

Examples within the help file now use Visual C# as the default programming language.

New Features

Command-Line options

The following two command-line options have been added:

- Don't brag (/db)
- Send to plugin (/sto)

Model Document is now called HotDocs Models. This has changed the name of the command-line option from Model Document to HotDocs Model but the option itself remains /mo.

COM API

HDServerFileType Enumumeration was added.

Dependency Type Enumeration: The following directories were added:

- MissingFileDependency
- AdditionalTemplateDependency

HDAUI Enumeration: The following values were added:

- AUIDLGRESOURCEPANE
- AUIFORMRESOURCEPANE
- AUIVARSHEETRESOURCEPANE
- AUIEOISENDTOOUTPUT
- AUIFILESELECTOPENANSWERS
- AUIVIEWDIALOGNAVIGATIONBAR
- AUIVIEWENDOFINTERVIEWDIALOG

- AUIVIEWEXPANDALL
- AUIVIEWCOLLAPSEALL
- AUIVIEWPREVIEWTAB
- AUIASMQUEUE

HDLUI Enumeration: The following values were added:

- LUIVIEWTITLES
- LUIVIEWFILENAMES
- LUIVIEWEXPANDALL
- LUIVIEWCOLLAPSEALL
- LUIVIEWTABSATTOP
- LUIVIEWMARKUPVIEW
- LUITEMPLATEPRINT
- LUITEMPLATECONVERTTOMODEL
- LUITEMPLATECONVERTFROMMODEL
- LUITOOLSTEMPLATEMANAGER
- LUITOOLSAUTOMATOR
- LUIHIDDENDATAREMOVER

HotDocs.Application Object: The following properties and events were added:

- *CreateTemplatePackage* Method
- *PublishOnlineFiles2* Method
- *AssemblyCompleteEvent* Event
- *OnLibraryOpenEvent* Event

HotDocs.Assembly Object: The following events were added:

- *PostSaveAnswersEvent* Event (replaces *OnPostSaveAnswersEvent* Event)
- *PreSaveAnswersEvent* Event (replaces *OnPreSaveAnswersEvent* Event)

Plugins

New HotDocs Plug-in APIs were added:

- *IOutputPlugin* interface
- *IPluginPreferences* Interface

New and Enhanced Features in the HotDocs 10 API

HotDocs 10

- **Default folders and registry keys:** The API documentation has been updated to reflect the new default file folders and registry keys used by HotDocs 10.

- **New HotDocs product names:** The HDPRODUCTFLAVOR Enumeration has been updated to reflect the new HotDocs product names. Also, the *Application.CanAssembleAll* and *Application.CanEditTemplates* properties were added to help integrations determine what features are available in the instance of HotDocs in use.

HotDocs 10.1

- **Template Dependency API:** The new *TemplateInfo* object lets you determine which files are required by a given template. For example, if you create a source control integration, you could use this new API to make sure that when you check a file out of source control, you also get all of its dependent files. The *Dependency* and *DependencyCollection* objects complete the set of new objects that make up the template dependency API.

New and Enhanced Features in the HotDocs 2009 API

HotDocs 2009

- **Answer Source API:** New functions have been added to support Unicode strings in the answer source API: *GetFieldW* , *GetFieldAccessW* , *GetFieldNameW* , *and SetFieldW*.

- **HDAFFORMAT Enumeration:** New values (*PreHD2009Format* and *HD2009Format*) have been added, which correspond to the two answer file formats that HotDocs 2009 can write.

New and Enhanced Features in the HotDocs 2008 API

HotDocs 2008

- **Model Document Support:** New functions have been added to the *Application* object to enable converting templates to Model Documents, and Model Documents to templates. (See ConvertModelToTemplate and ConvertTemplateToModel.)

New and Enhanced Features in the HotDocs 2007 API

HotDocs 2007

- **Command-Line Options:** HotDocs now includes two new command-line options—Start Interview Group (/sig) and Keep Interview Group (/kig). These options are used to control which questions are asked when assembling a group of related documents; specifically, they keep questions that are already answered in one interview from being asked in subsequent interviews.

New and Enhanced Features in the HotDocs 2006 API

HotDocs 2006

- **ComponentCollection Object:** The *_ComponentCollection4 interface was added, which includes methods and properties for creating and editing various types of HotDocs components. This interface, along with the new ComponentProperties object, allows you to create and customize virtually any HotDocs component.*
- ***ComponentProperties Object:** This new object is a collection of ComponentProperty objects, which allows you to view or make changes to settings of components in an open component file. (Click here for a list of properties for various HotDocs component types.)*
- ***ComponentProperty Object:** This new object represents a single property (setting) of a HotDocs component, such as a variable name or prompt.*
- ***Component Object** The _Component3 interface was added, which includes a new property (Properties) to view or change the properties of the component. It also includes a method (DisplayEditor) for displaying a component editor.*
- ***HDVARTYPE Enumeration:** The HDVARTYPE enumeration includes a number of additional types for use when creating and interacting with HotDocs components.*
- ***Command-Line Options:** A new command-line option, Edit Template (/ed), allows you to more easily open a HotDocs template for editing.*

New and Enhanced Features in the HotDocs 2005 API

HotDocs 2005

- ***Application object:*** *The following two methods were added:*
  - *RetrieveUrlFile: Retrieves a file from a given URL.*
  - *AddUserMenuItem2: Allows you to add custom menu items to the HotDocs library menus.*
- ***Library object:*** *The following property was added:*
  - *Redraw: Refreshes the library window.*

HotDocs 2005 SP2

- ***Answer Source Integrations:*** *New functions have been added to improve answer source integrations. These functions allow answers to be back-filled (written back to the answer source) and they also allow users to select multiple records in spreadsheet dialogs. (See What is an answer source integration?.)*
- ***HotDocs Plug-ins:*** *A number of changes were made to the HotDocs plug-in architecture. These changes include different parameters in existing interfaces and new methods for registering plug-ins. Although the interfaces still have the same names, the IIDs have changed and plug-ins created for use with earlier versions of HotDocs will not work with HotDocs 2005 SP2 unless they are modified and registered with HotDocs 2005 SP2. (See What is a HotDocs plug-in?)*
- ***Variable Mapping:*** *New methods were added to the HotDocs Variable Mapping interface to allow mapping of Text variables to Multiple Choice variables. The mapping interface can also now allow answers to be written back (or back-filled) to the original answer source. (See HotDocs.VarMap Object.)*

New and Enhanced Features in the HotDocs 6.2 API

HotDocs 6.2 SP1

- ***HotDocs Plug-ins:*** *You can now create HotDocs plug-ins to extend the library window user interface. Plug-ins can be used to add menus and items to the library menu, create shortcut menus, and customize the way HotDocs assembles and edits files. (See How do I create a HotDocs plug-in? for more information.)*

New and Enhanced Features in the HotDocs 6.1 API

HotDocs 6.1

- **ComponentCollection Object:** The *_ComponentCollection2* interface was added to the *ComponentCollection* object. This interface includes a method for creating a new, empty component file (*Create*) and a method for creating a new variable in a component file (*CreateVariable*).

HotDocs 6.1 SP1

- **Application Object:** The *_Application2* interface was added to the *Application* object. This interface includes a new method for saving documents as PDF files: *SaveDocAsPDF*.
- **Application Object:** A new method was added to the *Application* object for publishing files for use with HotDocs Online Server (HotDocs Server): *PublishOnlineFiles*.
- **Assembly Object:** The *_Assembly2* interface was added to the *Assembly* object. This interface includes a new property that controls whether HotDocs will prompt to save a copy of the assembled document at the end of the assembly: *PromptToSaveDocument*.

About the HotDocs API

What is the HotDocs API?

The HotDocs Application Programming Interface (API) lets you integrate HotDocs document assembly software functionality into your own application. For example, by integrating your application with HotDocs, you could allow your users to assemble HotDocs text and form documents using your application's data. Although HotDocs can be integrated with virtually any kind of software, it is most often used with database applications, case managers, litigation support systems, infobases, address books, and other similar applications.

There are three types of people involved in the integration process:

- **Integrator:** The most basic role of the integrator is to make data from your application available to HotDocs using the API. In addition, the integrator may also provide a way for HotDocs to be launched (visible or hidden) from within your application to assemble documents.

- **Template Developer:** The template developer's role is to create one or more HotDocs templates, which might use information from your application. If information will be pulled from your application, the template developer can use the HotDocs Variable Mapping dialog box to map fields in your application to HotDocs variables in each template.

- **End User:** The role of the end user is to assemble a text or form document using information from your application or answers entered during an interview. Depending on the type of integration you create, the user may or may not need to be actively involved.

This help file focuses primarily on the role of the integrator, or software developer. The roles of template developers and end users are discussed only to show how the integrator's work helps them perform their tasks.

If you are unfamiliar with HotDocs and its terminology, please refer to the Glossary to see a list of concepts and terms associated with HotDocs.

Integration Types

The HotDocs API provides methods for creating several types of integrations. Using the COM API, your application can act as the main user interface for your users. For example, it can launch HotDocs, tell it which template to use, and provide it with data it needs to assemble a document.

Another type of integration you can create is an answer source integration, which is a special DLL file that HotDocs can use to query your application for the data it needs to assemble a document. When using an answer source integration, the user is more involved in selecting the template and determining what information from your application should be used.

HotDocs also provides a plug-in API, which lets you to create a plug-in that adds a new menu to the HotDocs library window, overlays custom icons on top of template icons in the library, or otherwise extends the menus and functionality of the HotDocs library window.

Finally, the Variable Mapping API allows you to create associations between data fields in your application and HotDocs variables.

Using Command-Line Options

Introduction: Command-Line Options

You can use command-line options to control the way HotDocs assembles text and form documents. Most options can be placed on any command line that causes HotDocs to run. They are case-sensitive and must be typed in lowercase letters. If the option requires you to include a full file path, you must enclose the file path with quotation marks.

You can use command-line options in different ways, including specifying the options at the command line, at the **Properties** dialog box of a given template, and for the program file's shortcut.

Most command-line options control certain aspects of document assembly. For example, you can specify an option that always prints an assembled document once it has been sent to the word processor.

When HotDocs is started from the command line, it will continue to run, even after the processing of the command-line request is completed. To tell HotDocs to close after processing the command-line request, use the **Exit HotDocs** option.

If you are using multiple options on a single command line, you must separate each option with a space character. If the command line includes space characters, you must enclose the path in quotation marks.

To use command-line options when starting HotDocs

1. Choose **Run** from the **Start** menu. The **Run** dialog box appears.
2. Type **"C:\Program Files\hotdocs.exe"** (including the quotation marks) followed by a space and the options you want in the **Open** field. For example:

   ```
   "C:\Program Files\HotDocs\Hotdocs.exe" /tf=demoempl.docx
   ```

3. Click **OK**. If HotDocs is not already running, it loads and then performs the command-line instructions. If HotDocs is already running, it performs the command-line instructions.

To use command-line options when using a shortcut to start HotDocs

1. Locate the HotDocs program file shortcut. (A shortcut is an icon on the desktop or **Start** menu that a user can click to quickly access a program.)
2. Right-click the icon and select **Properties** from the shortcut menu. The program item's **Properties** dialog box appears.
3. Select the **Shortcut** tab.
4. In the **Target** field, enter a space after the executable (.EXE) file and type the options you want.

To use command-line options within ASSEMBLE instructions

1. Open the library you need at the HotDocs library window.
2. Select the template with the ASSEMBLE instruction you wish to add a command-line option to, and click **Edit**.
3. Double click on the ASSEMBLE instruction.
4. The **Other Field** dialog box will open.
5. In the **Template to assemble** field, following the file name, type a space and the command-line options you want to use. For example:

```
Collection Letter.docx /sa
```

To add command-line options to a library item

1. Open a library at the HotDocs library window.
2. Select the library item and click the **Properties** button to open the **Item Properties** dialog box.
3. In the **File name** field, following the file name, type a space and the command-line options you want to use. For example:

```
/af="C:\Documents and Settings\Username\My
Documents\HotDocs\Answers\jalvey.anx"
```

Full List of Command-Line Options

Below is a list of all the available command-line options grouped by use. For more information on any option follow the link to its full description.

Installation Switches

These switches can be used to modify HotDocs installation and registration.

Option		Description
Mutually Exclusive*	/if=""	The **Installation File** option specifies an alternate installation script to execute in lieu of Hotdocs.inx , which sets up a workstation for individual HotDocs users. This file must conform to the HotDocs Installation Script DTD

	/si	The **Suppress Installation** option keeps HotDocs from installing itself on a per-user basis. (Normally, HotDocs runs a per-user installation the first time a user starts HotDocs.)
Mutually Exclusive*	/regserver	The **Register** option registers the HotDocs COM servers in the registry.
	/unregserver	The **Unregister** option unregisters the HotDocs COM servers from the registry.

Application Control Switches

These switches are used to feed instructions to the HotDocs executable from the command line.

Option		Description
Mutually Exclusive*	/tf=""	The **Template File** option causes HotDocs to assemble a document using the specified template or clause library.
	/ed=""	The **Edit Template** option causes HotDocs to edit a template using the specified template or clause library file.
	/ha=""	The **HotDocs Auto-Assemble File** option causes a document to be assembled using the specified auto-assemble (.HDA) file.
	/hi=""	The **HotDocs Auto-Install File** option causes the template set in the specified auto-install (.HDI) file to be installed. During this process, HotDocs prompts the user for any required information it needs to install the template library to the correct location.
/lf=""		The **Library File** option allows you to start HotDocs and open a specific library. If HotDocs is already running, it opens the library specified by the path and file name.
/ll		The **Lock Library** option locks the current library and prevents the user from editing the library or its contents. Specifically, when HotDocs is launched and the library appears, users can select templates and assemble documents from them. They can also view the answer library and change user preferences at the **HotDocs Options** dialog box. All other options are unavailable.
/ex		The **Exit HotDocs** option closes HotDocs when both of the following conditions are met: 1) there are no documents waiting to be assembled, and 2) all other programs are finished using HotDocs.
/nx		The **No Exit** option allows the integration to keep an instance of HotDocs running without displaying any interface.
/db		The **Don't Brag** option stops HotDocs from displaying the splash screen when it opens.
Mutually Exclusive*	/hl	The **Hide Library** option causes HotDocs to assemble a document without first displaying the HotDocs template library. The user will not see the template library window at all during assembly. It is most commonly used by integrators who are using HotDocs with a third-party program and want to start an assembly without first displaying the template library window.

	/sl	The **Show Library** option forces HotDocs to display the template library if it is currently not showing. This is useful if you are integrating HotDocs with another program and you have hidden the library using the **Hide Library** option.

Template Type Switches

When a template is referred to by HotDocs, the template type is inferred from the file name extension. However, in some cases additional information is required so HotDocs knows how to process the requested template. These **mutually exclusive** switches (usable in combination with the /tf="" switch, on library items, or in INSERT or ASSEMBLE instructions) help with that. This switch (usable in combination with the /tf="" switch, on library items) can help with that.

Option		Description
Mutually Exclusive*	/mo	The **HotDocs Model** command-line option indicates that the file referenced in the library is a HotDocs Model. When you select the document in the library and click **Assemble**, HotDocs will create an interview for the model.
	/cl="name"	The **Clause Name** option is used by HotDocs to identify which clause component is associated with an item in a clause library. It is also used by HotDocs to process INSERT instructions during the assembly process. Generally speaking, developers should never have to modify this option unless they are converting clauses from one file format to another. Likewise, end users may see the **Clause Name** option while working with clauses at a clause library or during assembly, but should not modify it.

Answer Initialization Switches

These switches are used to initialize & define the answer set that will be used for an interview or assembly. They can be used in combination with the /tf="" switch and on library items.

Option		Description
/df=""		The **Default Answer File** option specifies a default answer file that is used to "seed" any answer file created during assembly. When a new answer file is created, it is automatically loaded with answers from the default answer file.
Mutually Exclusive*	/af=""	The **Answer File** option is useful if you want to use a specific answer file when you assemble a document. The option does two things: 1) when a template is selected for assembly, it immediately opens the specified answer file without displaying the **Answer File** dialog box, and 2) it sets the value for path and file name as the current answer file name to be used when answers are saved. If the specified answer file doesn't exist, it will be created when the user saves the answers

	/na[=""]	The **New Answer File** option specifies a new, untitled answer file to be used when assembling a given document. This option causes HotDocs to suppress the **Answer File** dialog box, which normally appears before assembly. Specifying a path and file name is optional. If a file name is specified, it will be used for the new answer file. If an answer file with that same name already exists, HotDocs overwrites the existing file with the new one. If no file name is specified, HotDocs displays a **Save Answer File** dialog box at the end of assembly.
/ov=""		The **Overlay Answer File** option causes HotDocs to take answers from a specific answer file and overlay them in the current answer file. For example, if you have specific information about a client that can be used in assembling multiple documents, you can save just that information in an overlay answer file and then use the **Overlay Answer File** option to force HotDocs to use those answers when assembling a document. All answers entered during assembly (including overlaid answers) are saved to the current answer file—not the overlay answer file—thus maintaining the integrity of the overlay answer file. An overlay answer file is loaded after the regular answer file so that the answers contained therein can overlay existing answers.

Interview Behavior Switches

These switches are used to modify the default behavior of the assembly window. They can be used in combination with the /tf="" switch and on library items.

Option		Description
Mutually Exclusive*	/nw	The **No Assembly Window** option causes HotDocs to assemble a document without displaying the assembly window.
	/ni	The **No Interview** option removes the **Interview** tab from the assembly window, and,by default, displays the assembled document in the **Document Preview** or **Form Document** tab (depending on whether you are assembling a text or form document). To present a correctly assembled document, you should specify an answer file using the **Answer File** option. Otherwise, the document will be assembled without any answers.
	/fia	When a user starts assembling a template that has the **Finish Interview Action** command-line option applied, HotDocs will complete the action defined in HotDocs Options—either display the assembled document at the **Document** tab of the assembly window or send the document to the word processor or HotDocs Filler.
/la		The **Lock Answer File** option prevents users from opening, closing, and saving answer files during document assembly. If it is the only option used, however, users can choose an answer file before assembly and save any answers they have entered after assembly.
/sig		The **Start Interview Group** option is used to control which questions are asked when assembling a group of related documents, specifically, it keeps questions that are already answered in one interview from being asked in subsequent interviews. It must be used with the **Keep Interview Group** option, which must be assigned to each subsequent template within the group.

/kig	The **Keep Interview Group** option is used to control which questions are asked when assembling a group of related documents, specifically, it keeps questions that are already answered in one interview from being asked in subsequent interviews. It must be used with the **Start Interview Group** option, which must be assigned to each subsequent template within the group.
/is=u	The **Interview Scope** option allows you to ask only those dialogs that contain questions not answered by an existing answer file. This may be useful, for example, if you have some answers you are retrieving from a database that you don't want the user to change. Using this option will ask only those questions that don't have answers. Cannot be used with the Start Interview Group option or the Keep Interview Group option.
/sw	The **Suppress Unanswered Warning** option keeps HotDocs from displaying the warning dialog box that appears when the user attempts to either print, save, or send the assembled document to the word processor and the assembled document still contains unanswered questions.

Answer Disposition Switches

These **mutually exclusive** switches can be used to dictate what happens to answers that were modified while the assembly window was open or during assembly. They can be used in combination with the /tf="" switch and on library items.

Option		Description
Mutually Exclusive*	/sa	The **Save Answers** option forces an answer file to be saved at the end of an assembly. If using an existing answer file, any answers entered during the interview will be saved automatically. If using a new, untitled answer file, HotDocs will force the user to specify an answer file name.
	/sap	The **Save Answers Prompt** option, which is used in connection with an ASSEMBLE instruction, prompts the user to save an answer file after completing an interview. Regardless of whether the user uses an existing answer file during assembly, when the user finishes that assembly, HotDocs prompts to save the answers in a different file.
	/ss	The **Suggest Save** option, which is used in connection with an ASSEMBLE instruction, causes HotDocs to ask users after assembly of a document has finished if they want to save answers entered during the interview in an answer file. Specifically, if the user has assembled a document and made changes to an existing answer file, HotDocs prompts to save the answers to that file. If saving a new, untitled file, HotDocs allows the user to specify the new answer file name.
	/ssn	The **Suggest Save New** option, which is used in connection with an ASSEMBLE instruction, causes HotDocs to ask if answers should be saved in a new answer file after assembly of a document has finished. Regardless of whether the user is using an existing answer file during assembly, when the user finishes that assembly, HotDocs gives the user the option of saving the answers in a new answer file.
	/da	The **Discard Answers** option prevents the user from saving answers after the document has been assembled. This option is useful when you know you will never want to save the answers you use with a particular template (for example, a fax

		cover sheet), and you don't want HotDocs to ask about saving the answers when you close the assembly window. However, the user can save the answer file during the interview.

Document Disposition Switches

These switches can be used to dictate what happens to a document after its assembly is complete. They can be used in combination with the /tf="" switch and on library items.

Option	Description
/of=""	The **Output File** option causes HotDocs to assemble the document and save it using the file name specified. If you are using the **Answer Summary** or **Question Summary** options, the **Output File** option specifies the name for either of those generated documents. This is useful if you know you want to save an assembled document every time assembly of that document finishes.
/as	The **Answer Summary** option is used with the **Output File** option to specify the path and file name for saving an answer summary. It is useful if you want a certain template to always generate an answer summary document.
/qs	The **Question Summary** option is used with the **Output File** option to specify the path and file name for saving a question summary. It is useful if you want a certain template to always generate a question summary document.
/sto	The **Send to Plugin** option sends the assembled document to a specified output plugin.
/stw	The **Send to Word Processor** option sends the assembled document to the word processor once the user closes the assembly window. This is useful if you know you always want to view the assembled document using the word processing program.

Automator/Filler Command Line Switches

These switches can be used to dictate how HotDocs will print form templates from HotDocs Automator/Filler.

Option	Description
/pr	The **Print** option causes HotDocs to print a copy of the assembled text or form document once the user closes the assembly window. This is useful if you know you will always need to print a copy of a specific assembled document.
/pw	The **Print Without Dialogs** option causes HotDocs to bypass the **Print** dialog box and print the form using the current printer. The form is printed when the user clicks the **Print Document** button at the assembly window.
/ps=""	The **Paper Size** option selects the specified paper size when the user prints a copy of the form template or document. The effect is the same as manually setting the page size from the **Print** dialog box. This option works with form templates and documents only.
/pt=""	The **Paper Tray** option causes a specified printer paper tray or manual feed option

		to be used when printing a form document from HotDocs Filler. Paper tray values that can be used include **manual**, **upper**, **lower**, and so forth. For a complete list of acceptable values, either at the assembly window or at the HotDocs Filler window, click **Document Properties > Printing** (**File** menu) and click the **Paper Source** drop-down button.
/pc=n		The **Print Copies** option specifies the number of copies that should be printed when the user prints the form document (type in the number of copies needed instead of the letter n). This number should appear in the **Number of Copies** field at the **Print** dialog box.
/pd		The **Print Duplex** option sets the duplex printing option for a given form document. It prints the document **Double-Sided**, **Side-to-Side**, as if that option were selected at the **Printing Properties** dialog box (which you can access by clicking **Document Properties > Printing** (**File** menu).) When the user prints the form document, it is printed using this option.
Mutually Exclusive*	/pa	The **Print Answers Only** option selects the **Answers Only (Use Preprinted Form)** option at the **Print** dialog box. Then, when the user prints the assembled form document, it prints only the form's answers and not the underlying static text. This allows you to use preprinted forms.
	/po	The **Print Form Only** option selects the **Form Only (Blank Form)** option at the **Print** dialog box. Then, when the user prints the form document, it prints a blank copy of the form without answers.
	/pb	The **Print Both** option selects the **Form with Answers** option at the **Print** dialog box. Then, when the user prints the form document, the current form and its answers are printed.

*Mutually exclusive switches will not show an error if used in the same command-line but will cause unpredictable behavior. We recommend you do not use these switches together.

Use Command-Line Options When Starting HotDocs

To use command-line options when starting HotDocs

1. Choose **Run** from the **Start** menu. The **Run** dialog box appears.
2. Type **"C:\Program Files\hotdocs.exe"** (including the quotation marks) followed by a space and the options you want in the **Open** field. For example:

```
"C:\Program Files\HotDocs\Hotdocs.exe" /tf=demoempl.docx
```

3. Click **OK**. If HotDocs is not already running, it loads and then performs the command-line instructions. If HotDocs is already running, it performs the command-line instructions.

Use Command-Line Options when Using a Shortcut to Start HotDocs

To use command-line options when using a shortcut to start HotDocs

1. Locate the HotDocs program file shortcut. (A shortcut is an icon on the desktop or **Start** menu that a user can click to quickly access a program.)
2. Right-click the icon and select **Properties** from the shortcut menu. The program item's **Properties** dialog box appears.
3. Select the **Shortcut** tab.
4. In the **Target** field, enter a space after the executable (.EXE) file and type the options you want.

Use Command-Line Options within ASSEMBLE Instructions

To use command-line options within ASSEMBLE instructions

1. Open the library you need at the HotDocs library window.
2. Select the template with the ASSEMBLE instruction you wish to add a command-line option to, and click  **Edit**.
3. Double click on the ASSEMBLE instruction.
4. The **Other Field** dialog box will open.
5. In the **Template to assemble** field, following the file name, type a space and the command-line options you want to use. For example:

```
Collection Letter.docx /sa
```

Use Command-Line Options at File Properties

To add command-line options to a library item

1. Open a library at the HotDocs library window.
2. Select the library item and click the  **Properties** button to open the **Item Properties** dialog box.
3. In the **File name** field, following the file name, type a space and the command-line options you want to use. For example:

```
/af="C:\Documents and Settings\Username\My
Documents\HotDocs\Answers\jalvey.anx"
```

Answer File

```
/af="path and file name"
```

The **Answer File** option is useful if you want to use a specific answer file when you assemble a document. The option does two things: 1) when a template is selected for assembly, it immediately opens the specified answer file without displaying the **Answer File** dialog box, and 2) it sets the value for path and file name as the current answer file name to be used when answers are saved. If the specified answer file doesn't exist, it will be created when the user saves the answers.

When using an existing answer file, you can retrieve an answer file from a location on a Web server by specifying a URL for the path and file name (for example, */af=http://www.yoursite.com/answers.anx*). (You cannot, however, save an answer file back to the server.)

You cannot assign the **Answer File (/af)** option to a HotDocs Auto-Assemble file.

If using this option at the command line, include the **Template File (/tf)** option.

Answer Summary

```
/as
```

The **Answer Summary** option is used with the **Output File** option to specify the path and file name for saving an answer summary. It is useful if you want a certain template to always generate an answer summary document.

The **Answer Summary** option is normally used with the **No Assembly Window** and **Answer File** options, which cause HotDocs to automatically create and save the answer summary document without displaying the assembly window.

Answer summaries are saved in HTML format.

If using this option at the command line, include the **Template File (/tf)** and **Output File (/of)** options. If using this option at the library properties, include the **Output File (/of)** option.

Clause Name

```
/cl=clausename
```

The **Clause Name** option is used by HotDocs to identify which clause component is associated with an item in a clause library. It is also used by HotDocs to process INSERT instructions during the assembly process. Generally speaking, developers should never have to modify this option unless they are converting clauses from one file format to another. Likewise, end users may see the **Clause Name** option while working with clauses at a clause library or during assembly, but should not modify it.

Default Answer File

```
/df="path and file name"
```

The **Default Answer File** option specifies a default answer file that is used to "seed" any answer file created during assembly. When a new answer file is created, it is automatically loaded with answers from the default answer file.

When specified, it does not need to have the same file name as the template's component file, nor does it need to be saved in the same folder as the component file. However, the default answer file name should be different from the current answer file name. Also, when using an existing default answer file, you can retrieve it from a location on a Web server by specifying a URL for the path and file name (for example, */df=http://www.yoursite.com/defaultanswers.anx*).

If using this option at the command line, include the **Template File (/tf)** option.

Discard Answers

```
/da
```

The **Discard Answers** option prevents the user from saving answers after the document has been assembled. This option is useful when you know you will never want to save the answers you use with a particular template (for example, a fax cover sheet), and you don't want HotDocs to ask about saving the

answers when you close the assembly window. However, the user can save the answer file during the interview.

To disable all answer file usage (saving, selecting new, and so forth) during a given assembly, use the **Lock Answer File (/la)** option.

If using this option at the command line, include the **Template File (/tf)** option.

Don't Brag

```
/db
```

The **Don't Brag** option stops HotDocs from displaying the splash screen when it opens.

Edit Template

This option was introduced with the release of HotDocs 2006.

```
/ed="path and file name"
```

The **Edit Template** option causes HotDocs to edit a template using the specified template or clause library file.

Exit HotDocs

```
/ex
```

The **Exit HotDocs** option closes HotDocs when both of the following conditions are met: 1) there are no documents waiting to be assembled, and 2) all other programs are finished using HotDocs.

Finish Interview Action

```
/fia
```

When a user starts assembling a template that has the **Finish Interview Action** command-line option applied, HotDocs will complete the action defined in HotDocs Options—either display the assembled document at the **Document** tab of the assembly window or send the document to the word processor or HotDocs Filler.

If using this option at the command line, include the **Template File (/tf)** and **Answer File (/af)** options. If using this option at the library properties, include the **Answer File (/af)** option.

Hide Library

```
/hl
```

The **Hide Library** option causes HotDocs to assemble a document without first displaying the HotDocs template library. The user will not see the template library window at all during assembly. It is most commonly used by integrators who are using HotDocs with a third-party program and want to start an assembly without first displaying the template library window. (See also Show Library.)

If using this option at the command line, include the **Template File (/tf)** option.

HotDocs Auto-Assemble File

```
/ha="path and file name"
```

The **HotDocs Auto-Assemble File** option causes a document to be assembled using the specified auto-assemble (.HDA) file.

You can also specify a URL for the path and file name (for example, */ha=http://www.yoursite.com/hdafile.hda*).

HotDocs Auto-Install File

```
/hi="path and file name"
```

The **HotDocs Auto-Install File** option causes the template set in the specified auto-install (.HDI) file to be installed. During this process, HotDocs prompts the user for any required information it needs to install the template library to the correct location.

You can also specify a URL for the path and file name (for example, */hi=http://www.yoursite.com/hdifile.hdi*). When this command is passed, HotDocs downloads the file and prompts the user for the information needed to install the template library.

HotDocs Model

```
/mo
```

The **HotDocs Model** command-line option indicates that the file referenced in the library is a HotDocs Model. When you select the document in the library and click **Assemble**, HotDocs will create an interview for the model.

See Overview: Create HotDocs Models for more information.

Installation File

```
/if=path and file name
```

The **Installation File** option specifies an alternate installation script to execute in lieu of Hotdocs.inx , which sets up a workstation for individual HotDocs users. This file must conform to the HotDocs Installation Script DTD (http://support.hotdocs.com/dtd/hotdocs6_inx.dtd).

Interview Scope

```
/is=u
```

The **Interview Scope** option allows you to ask only those dialogs that contain questions not answered by an existing answer file. This may be useful, for example, if you have some answers you are retrieving from a database that you don't want the user to change. Using this option will ask only those questions that don't have answers.

If you do not want certain variables to appear in an interview, do not include them in an explicit ASK instruction. Otherwise, HotDocs will present the variables to the user.

Cannot be used with the **Start Interview Group (/sig)** option or the **Keep Interview Group (/kig)** option.

Keep Interview Group

This option was introduced with the release of HotDocs 2007.

```
/kig
```

The **Keep Interview Group** option is used to control which questions are asked when assembling a group of related documents, specifically, it keeps questions that are already answered in one interview from being asked in subsequent interviews. It must be used with the **Start Interview Group** option, which must be assigned to each subsequent template within the group.

For example, you have three related templates that will be added to the assembly queue (*Template A*, *Template B*, and *Template C*). Each of these templates uses *Variable A*. To keep *Variable A* from being asked in all three interviews, you would assign the **Start Interview Group** option to *Template A*. Then you would assign the **Keep Interview Group** option to *Templates B* and *C*. Once the user answers *Variable A*, it will not be asked in any subsequent interviews.

If a template is added to the assembly queue that doesn't use either of these options, it and any subsequent templates will not be included in the interview group.

Library File

```
/lf="path and file name"
```

The **Library File** option allows you to start HotDocs and open a specific library. If HotDocs is already running, it opens the library specified by the path and file name.

Lock Answer File

```
/la
```

The **Lock Answer File** option prevents users from opening, closing, and saving answer files during document assembly. If it is the only option used, however, users can choose an answer file before assembly and save any answers they have entered after assembly.

To keep users from choosing an answer file before assembly, specify an answer file using the **Answer File** option. To keep them from manually saving their answers, use either the **Discard Answers** option or the **Save Answers** option.

If using this option at the command line, include the **Template File (/tf)** option.

Does not work in conjunction with the **No Assembly Window (/nw)** option.

Lock Library

```
/ll
```

The **Lock Library** option locks the current library and prevents the user from editing the library or its contents. Specifically, when HotDocs is launched and the library appears, users can select templates and assemble documents from them. They can also view the answer library and change user preferences at the **HotDocs Options** dialog box. All other options are unavailable.

New Answer File

```
/na[="path and file name"]
```

The **New Answer File** option specifies a new, untitled answer file to be used when assembling a given document. This option causes HotDocs to suppress the **Answer File** dialog box, which normally appears before assembly. Specifying a path and file name is optional. If a file name is specified, it will be used for the new answer file. If an answer file with that same name already exists, HotDocs overwrites the existing

file with the new one. If no file name is specified, HotDocs displays a **Save Answer File** dialog box at the end of assembly.

When specifying an answer file name, you must include the file name extension .ANX.

If using this option at the command line, include the **Template File (/tf)** option.

No Assembly Window

```
/nw
```

The **No Assembly Window** option causes HotDocs to assemble a document without displaying the assembly window.

If using this option at the command line, include the **Template File (/tf)** and **Answer File (/af)** options. If using this option at the library properties, include the **Answer File (/af)** option.

You should also use the **Output File (/of)** or **Send to Word Processor (/stw)** option if you want HotDocs to produce a document at the end of the assembly.

No Exit

```
/nx
```

The **No Exit** option allows the integration to keep an instance of HotDocs running without displaying any interface.

No Interview

```
/ni
```

The **No Interview** option removes the **Interview** tab from the assembly window, and, by default, displays the assembled document in the **Document Preview** or **Form Document** tab (depending on whether you

are assembling a text or form document). To present a correctly assembled document, you should specify an answer file using the **Answer File** option. Otherwise, the document will be assembled without any answers.

While viewing an assembled document that was generated using this command-line option, you cannot edit answers while viewing the **Document** tab.

If using this option at the command line, include the **Template File (/tf)** and **Answer File (/af)** options. If using this option at the library properties, include the **Answer File (/af)** option.

Output File

```
/of="path and file name"
```

The **Output File** option causes HotDocs to assemble the document and save it using the file name you specify. If you are using the **Answer Summary** or **Question Summary** options, the **Output File** option specifies the name for either of those generated documents. This is useful if you know you want to save an assembled document every time that HotDocs finishes assembling that document.

Changing the file extension on the file specified in the Output File option instructs HotDocs to convert the output file to the new file type. For example, adding the command line option **`/of="C:\templatename.pdf"`** to a DOCX template results in an PDF format output file.

Since HotDocs cannot convert to all format types, we recommend that you manually test HotDocs conversion capablities for a desired format before adding the command-line option. Be aware as well that converting output files to alternate formats can sometimes cause variations in formatting and user experience.

If you use this option with the command line, be sure to include the **Template File (/tf)** option.

Overlay Answer File

```
/ov="path and file name"
```

The **Overlay Answer File** option causes HotDocs to take answers from a specific answer file and overlay them in the current answer file. For example, if you have specific information about a client that can be used in assembling multiple documents, you can save just that information in an overlay answer file and then use the **Overlay Answer File** option to force HotDocs to use those answers when assembling a

document. All answers entered during assembly (including overlaid answers) are saved to the current answer file—not the overlay answer file—thus maintaining the integrity of the overlay answer file. An overlay answer file is loaded after the regular answer file so that the answers contained therein can overlay existing answers.

If you do not include a full path on the command-line, HotDocs will first look for the answer file in the same folder as the template. If it's not located there, HotDocs will look in the *Answers* folder.

You can retrieve an overlay answer file from a location on a Web server by specifying a URL for the path and file name (for example, */ov=http://www.yoursite.com/overlayanswers.anx*).

If you are saving the overlay answer file to the same folder as the template, do not use the same name as the template. Otherwise, HotDocs will think the answer file is a default answer file. (See Create a Default Answer File.)

If using this option at the command line, include the **Template File (/tf)** option.

Paper Size

```
/ps=paper size
```

The **Paper Size** option selects the specified paper size when the user prints a copy of the form template or document. The effect is the same as manually setting the page size from the **Print** dialog box. This option works with form templates and documents only.

Paper size values that can be used include **letter**, **legal**, and so forth. For a complete list of acceptable values, either at the assembly window or at the HotDocs Filler window, click **Document Properties > Printing** (**File** menu) and click the **Paper Size** drop-down button. Values that include a space character must be placed inside quotation marks. You can shorten the values as long as the shortened form matches only one paper size. Paper size values are not case-sensitive.

If a paper size is specified at the **Printing Properties** dialog box and the **Paper Size (/ps)** command-line option is also used, the command-line option takes precedence.

If using this option at the command line, include the **Template File (/tf)** option

Paper Tray

```
/pt=paper tray
```

This option is used with form documents only.

The **Paper Tray** option causes a specified printer paper tray or manual feed option to be used when printing a form document from HotDocs Filler. Paper tray values that can be used include **manual**, **upper**, **lower**, and so forth. For a complete list of acceptable values, either at the assembly window or at the HotDocs Filler window, click **Document Properties > Printing** (**File** menu) and click the **Paper Source** drop-down button.

If a paper source is specified at the **Printing Properties** dialog box and the **Paper Tray (/pt)** command-line option is also used, the command-line option takes precedence.

If using this option at the command line, include the **Template File (/tf)** option.

Print

```
/pr
```

This option is used with form documents only

The **Print** option causes HotDocs to print a copy of the assembled text or form document once the user closes the assembly window. This is useful if you know you will always need to print a copy of a specific assembled document.

If using this option at the command line, include the **Template File (/tf)** option.

Print Answers Only

```
/pa
```

This option is used with form documents only

The **Print Answers Only** option selects the **Answers Only (Use Preprinted Form)** option at the **Print** dialog box. Then, when the user prints the assembled form document, it prints only the form's answers and not the underlying static text. This allows you to use preprinted forms.

If using this option at the command line, include the **Template File (/tf)** option.

Print Both

```
/pb
```

This option is used with form documents only

The **Print Both** option selects the **Form with Answers** option at the **Print** dialog box. Then, when the user prints the form document, the current form and its answers are printed.

If using this option at the command line, include the **Template File (/tf)** option.

Print Copies

```
/pc=numberofcopies
```

This option is used with form documents only

The **Print Copies** option specifies the number of copies that should be printed when the user prints the form document. This number should appear in the **Number of Copies** field at the **Print** dialog box

If using this option at the command line, include the **Template File (/tf)** option.

Print Duplex

```
/pd
```

This option is used with form documents only

The **Print Duplex** option sets the duplex printing option for a given form document. It prints the document **Double-Sided, Side-to-Side**, as if that option were selected at the **Printing Properties** dialog

box (which you can access by clicking **Document Properties > Printing** (**File** menu).) When the user prints the form document, it is printed using this option.

If using this option at the command line, include the **Template File (/tf)** option.

Print Form Only

```
/po
```

This option is used with form documents only

The **Print Form Only** option selects the **Form Only (Blank Form)** option at the **Print** dialog box. Then, when the user prints the form document, it prints a blank copy of the form without answers.

If using this option at the command line, include the **Template File (/tf)** option.

Print Without Dialogs

```
/pw
```

This option is used with form documents only

The **Print Without Dialog** option causes HotDocs to bypass the **Print** dialog box and print the form using the current printer. The form is printed when the user clicks the **Print Document** button at the assembly window. .

If using this option at the command line, include the **Template File (/tf)** option.

Question Summary

```
/qs
```

The **Question Summary** option is used with the **Output File** option to specify the path and file name for saving a question summary. It is useful if you want a certain template to always generate a question summary document.

The **Question Summary** option is normally used with the **No Assembly Window** option, which causes HotDocs to automatically create and save the question summary document without displaying the assembly window.

Question summaries are saved in HTML format.

If using this option at the command line, include the **Template File (/tf)** option.

Register

```
/regserver
```

The **Register** option registers the HotDocs COM servers in the registry.

Save Answers

```
/sa
```

The **Save Answers** option forces an answer file to be saved at the end of an assembly. If using an existing answer file, any answers entered during the interview will be saved automatically. If using a new, untitled answer file, HotDocs will force the user to specify an answer file name.

To always force the user to save a new answer file after entering answers in an interview—even if using an existing answer file—use the **Save Answers Prompt (/sap)** option.

To give users the option of saving an answer file, rather than forcing them to save, use either the **Suggest Save (/ss)** or the **Suggest Save New (/ssn)** option.

Save Answers Prompt

```
/sap
```

The **Save Answers Prompt** option, which is used in connection with an ASSEMBLE instruction, prompts the user to save an answer file after completing an interview. Regardless of whether the user uses an existing answer file during assembly, when the user finishes that assembly, HotDocs prompts to save the answers in a different file.

To always save an answer file without prompting the user for an answer file name (unless the user is using a new answer file) use the **Save Answers (/sa)** option.

To give users the option of saving an answer file, rather than forcing them to save, use either the **Suggest Save (/ss)** or the **Suggest Save New (/ssn)** option.

Send to Plugin

```
/sto="Name of Plugin"
```

The **Send to Plugin** option sends the assembled document to a specified output plugin. The class name of the plugin is the plugin name. For example, to use a google upload plugin the command line switch would be:

```
/sto="HDGoogleDriveOutputPlugin"
```

If using this option at the command line, include the **Template File (/tf)** option

Send to Word Processor

```
/stw
```

The **Send to Word Processor** option sends the assembled document to the word processor once the user closes the assembly window. This is useful if you know you always want to view the assembled document using the word processing program.

If using this option at the command line, include the **Template File (/tf)** option.

Show Library

```
/sl
```

The **Show Library** option forces HotDocs to display the template library if it is currently not showing. This is useful if you are integrating HotDocs with another program and you have hidden the library using the **Hide Library** option.

Start Interview Group

This option was introduced with the release of HotDocs 2007.

```
/sig
```

The **Start Interview Group** option is used to control which questions are asked when assembling a group of related documents, specifically, it keeps questions that are already answered in one interview from being asked in subsequent interviews. It must be used with the **Keep Interview Group** option, which must be assigned to each subsequent template within the group.

For example, you have three related templates that will be added to the assembly queue (*Template A*, *Template B*, and *Template C*). Each of these templates uses *Variable A*. To keep *Variable A* from being asked in all three interviews, you would assign the **Start Interview Group** option to *Template A*. Then you would assign the **Keep Interview Group** option to *Templates B* and *C*. Once the user answers *Variable A*, it will not be asked in any subsequent interviews.

If a template is added to the assembly queue that doesn't use either of these options, it and any subsequent templates will not be included in the interview group.

Suggest Save

```
/ss
```

The **Suggest Save** option, which is used in connection with an ASSEMBLE instruction, causes HotDocs to ask users after assembly of a document has finished if they want to save answers entered during the interview in an answer file. Specifically, if the user has assembled a document and made changes to an existing answer file, HotDocs prompts to save the answers to that file. If saving a new, untitled file, HotDocs allows the user to specify the new answer file name.

If your user is using an existing answer file but you want to give the user the option of saving the answers in a new, different answer file, use the **Suggest Save New (/ssn)** option.

If you want to force users to save their answers after an assembly is finished, rather than give them the option, use either the **Save Answers (/sa)** or the **Save Answers Prompt (/sap)** options.

Suggest Save New

```
/ssn
```

The **Suggest Save New** option, which is used in connection with an ASSEMBLE instruction, causes HotDocs to ask if answers should be saved in a new answer file after assembly of a document has finished. Regardless of whether the user is using an existing answer file during assembly, when the user finishes that assembly, HotDocs gives the user the option of saving the answers in a new answer file.

If your users are using an existing answer file and you want them to save answers they have entered in that file instead of a new one, use the **Suggest Save (/ss)** option.

If you want to force users to save their answers after an assembly is finished, rather than give them the option, use either the **Save Answers (/sa)** or the **Save Answers Prompt (/sap)** options.

Suppress Installation

```
/si
```

The **Suppress Installation** option keeps HotDocs from installing itself on a per-user basis. (Normally, HotDocs runs a per-user installation the first time a user starts HotDocs.)

If the per-user installation never runs for a user, that user may not be able to edit templates.

Suppress Unanswered Warning

```
/sw
```

The **Suppress Unanswered Warning** option keeps HotDocs from displaying the warning dialog box that appears when the user attempts to either print, save, or send the assembled document to the word processor and the assembled document still contains unanswered questions.

Template File

```
/tf="path and file name"
```

The **Template File** option causes HotDocs to assemble a document using the specified template or clause library.

If you want an interview template (component file) started from the **Template File** command line, the component file must have an INTERVIEW or STARTUP computation in it or assembly will fail.

Unregister

```
/unregserver
```

The **Unregister** option unregisters the HotDocs COM servers from the registry.

COM API

About the HotDocs COM API

About the HotDocs COM API

HotDocs includes a COM API that you can use in your own application to control HotDocs. At its root is the *Application* object, which represents the HotDocs library window and the main functional uses of HotDocs. In order to control HotDocs or perform most other tasks using the API, your application must get a reference to the *Application* object. The following code sample show how you can do this:

Example

Visual C#

```
HotDocs.Application app = new HotDocs.Application();
```

The HotDocs *Application* object is a singleton. This means that there will never be more than one HotDocs *Application* object per Windows desktop at a time. If two or more applications get a reference to the HotDocs *Application* object at the same time, they all hold references to the same object, even though they are all separate processes. This also means that every resource the HotDocs *Application* object has is shared among all the applications that hold references to it. This includes the Assembly Queue, application visibility, and so forth.

COM Interface

The HotDocs COM interface includes the following objects, which you can use when developing your HotDocs integration:

- *HotDocs.Answer* Object
- *HotDocs.AnswerCollection* Object
- *HotDocs.Application* Object
- *HotDocs.Assembly* Object
- *HotDocs.AssemblyCollectionClass* Object
- *HotDocs.Component* Object
- *HotDocs.ComponentCollection* Object
- *HotDocs.ComponentProperties* Object
- *HotDocs.ComponentProperty* Object
- *HotDocs.Dependency* Object
- *HotDocs.DependencyCollection* Object
- *HotDocs.Icon* Object

- *HotDocs.Library* Object
- *HotDocs.LibraryEntity* Object
- *HotDocs.Plugin* Object
- *HotDocs.PluginsClass* Object
- *HotDocs.TemplateInfo* Object
- *HotDocs.VarMap* Object

Using the HotDocs COM API

The following topics contain information about using various features of the HotDocs COM API:

- Use the HotDocs COM API in .NET
- Use the HotDocs Variable Mapping API
- Understand COM Events
- How do I program a COM Event in Visual C#?

How do I use the HotDocs COM API in .NET?

If you use the .NET Framework to develop a HotDocs integration, you may run into a problem with .NET not cleaning references to COM objects immediately. This may cause HotDocs to continue running even after you would expect it to quit. Although .NET will eventually clean up references to the COM objects automatically, and cause HotDocs to close, you can avoid this delay by forcing a garbage collection call.

To force a garbage collection call

1. Subscribe to the *Application.OnAssemblyCompleteEvent* event.
2. When *OnAssemblyCompleteEvent* is fired, call *System.GC.Collect()*. This is a time-consuming API call, but it causes all the runtime callable wrappers (RCWs) that are not referenced to be cleaned up, releasing HotDocs and letting it shut down.

To avoid leaving files open longer than desired, you should also force the runtime to close objects when you are finished with them. For example, after you are finished with a *ComponentCollection*, you should call *Marshal.ReleaseComObject()* on the object.

How do I use the HotDocs Variable Mapping API?

The HotDocs Variable Mapping API lets you create associations between data fields in your application and HotDocs variables in your templates. For example, if your application contains a data store with information about customers, you can map the fields in your data store to variables in a HotDocs template. Then when a user assembles a document from the template, your application can pre-populate the *AnswerCollection* with appropriate data from the data store, such as a customer name or address.

Ideally, if the names of your application's data fields matched the names of your HotDocs variables, then variable mapping would not be required. Likewise, if the variable names and your data store's field names were guaranteed never to change, you might choose to hard-code mappings in your application. However, the reality is that while data fields and variables represent the same information, they often do not use the same name. The HotDocs Variable Mapping API addresses both of these problems--you can match data with different names, and you can easily change mappings whenever required by changes to the template or your data store.

The associations (mappings) you set up between HotDocs variables and your application's data fields are saved in a separate HotDocs map (.HMF) file for each template. When a template with an associated map file is assembled, HotDocs loads the information from the map file into the *Assembly*. However, once HotDocs loads the map file, it doesn't actually do anything to retrieve values from your application's data store; it is up to your application to process the mappings found in the map file and send values from your data store to the *AnswerCollection* associated with the current assembly.

HotDocs allows mapping to flow both ways, which means that in addition to sending answers from your application to HotDocs, HotDocs can also send changed answers back to your application.

A HotDocs Variable Map contains three collections of information:

Collection	Description
HDVariables	A list of HotDocs variables that can be mapped to fields in your data store. This collection is usually populated by copying the list of variables (and their types) from a HotDocs component file.
SourceNames	A list of fields in your data store.
Mapping	A list of mappings between variables and source names (fields) in your data store. Each item in this collection maps one variable name to one source name.

Mapping files created using HotDocs 2009 (or later) are saved in an XML file format, which cannot be read by earlier versions of HotDocs. HotDocs 2009-11 can still read existing binary mapping files created by earlier versions, however.

The following topics explain how you can use the HotDocs Variable Mapping API to create and use mapping files:

- Create a new HotDocs Map (.HMF) file
- Import one map file into another

- Use an existing variable mapping

Understand COM Events

COM events are similar to functions, except events allow two-way communication between a COM server and a client application. The basic idea of COM events is similar in principle to C-style callback functions. You give the COM server an interface to call when an event happens (subscribing), and the COM server calls a function on that interface when the event happens. With the events mechanism, a COM server can call back to a client to alert it of something asynchronously.

The COM event architecture consists of two interfaces, or parts:

- **Connection point:** This is the interface on the server that the client uses to tell the server that it is interested in receiving event notifications.

- **Event sink:** This is the interface the client implements that the server uses to send an event notification. An event sink is simply an IDispatch interface.

The following is an example of the logical sequence of calls for configuring these interfaces and receiving an event:

1. The client application gets a reference to the COM server (HotDocs) and asks it for a reference to the connection point that implements the event.

2. The server responds with a reference to the correct connection point.

3. The client calls a method on the connection point interface to "subscribe" to the event and passes a reference to the event sink interface as a parameter through which the server will call back.

4. The server stores the event sink interface until it needs to fire the event, in which case it calls the appropriate method on the event sink interface.

5. The client handles the method call.

6. When the client no longer needs to receive event notifications, it calls another method on the server to "unsubscribe" from the event and the server releases its reference to the event sink.

In order to use the HotDocs API, you must use COM events. For example, when an *Assembly* object is added to the *Assembly* collection, it is not assembled immediately. Instead, it is placed in a queue and the function call returns to the caller. Later, when HotDocs decides to run the assembly, HotDocs calls back (using a COM event) to the client to tell it that the assembly is ready to run.

The following HotDocs objects implement connection points that can be used to catch events:

Answer Events

Event	Description
OnValueFoundEvent	This event is fired for every value found for the particular answer. This event is fired during the processing of the *Answer.IterateValues* method.

Application Events

Event	Description
AssemblyCompleteEvent	This event is fired when assembly completes.
OnAssemblyCompleteEvent	This event has been deprecated for HotDocs Desktop 11. *AssemblyCompleteEvent* is recommended for use instead. This event is fired when an assembly is completed. It returns the name and path of the template that was used to assemble the document, the path to the assembled document, a pointer to the *AnswerCollection* object used in the assembly, and the assembly handle which was given when the assembly was added to the queue.
OnAssemblyStartEvent	This event is fired when an assembly starts. It returns a reference to the *Assembly* object that represents the assembly session.
OnErrorEvent	This event is fired when an error occurs. By returning true for the override parameter, the integration can tell HotDocs not to display any user interface indicating that an error occurred, which allows the integration to either display its own error message or silently handle the error.
OnLibraryInterfaceCloseEvent	This event is fired when the user closes the HotDocs library user interface.
OnLibraryOpenEvent	This event is fired when a library is opened.
OnTemplateSelectedEvent	This event is fired when the user selects a template in the library to assemble, or selects a template at the **SelectTemplate** or **SelectMultipleTemplate** dialogs. By returning *override == true, the integration can cancel the selection of the template.
OnUserInterfaceEvent	This event is fired when the user selects items in the library user interface.
OnUserMenuItemClickedEvent	This event is fired when the user selects an integration-defined menu item.

Assembly Events

Event	Description
OnAssemblyCompleteEvent	This event is fired when the assembly completes.
OnAssemblyStartEvent	This event is fired when the assembly starts.
OnCanOpenFile	This event is fired when a file can be opened.
OnCloseAssemblyInterfaceEvent	This event is fired when assembly interface actually closes.
OnErrorEvent	This event is fired when an error occurs. By returning true for the override parameter, the integration can tell HotDocs not to display any error messages, allowing the integration to either display its own error message or deal with the error silently.
OnFileOpen	This event is fired when a file is opened.
OnFileSave	This event is fired when a file is saved.
OnFileSelectEvent	This event is fired when a file is selected.
OnGetAnswerFileDisplayName	This event is fired when HotDocs gets the name of an answer file to display.
OnGetMRUInfo	This event is fired when HotDocs gets information from the most recently used (MRU) list.
OnNeedAnswerEvent	This event is fired when an answer value is needed by the assembly, but not found in the *AnswerCollection* answer set. It allows the integration to provide answers as they are needed, rather than trying to provide all the answers before the assembly starts.
OnPostCloseAnswerFile	This event is fired after the answer file is closed.
OnPostSaveDocumentEvent	This event is fired after a document is saved. The document can be an assembled document, a question summary document, or an answer summary document.
OnPreCloseAnswerFile	This event is fired when HotDocs prepares to close an answer file.
OnPreSaveDocumentEvent	This event is fired prior to saving the document. By setting *showui = true, the integration can prevent the user interface relating to saving the document from showing. By setting *override = true, the integration can prevent the save from happening.
OnUserInterfaceEvent	This event is fired when the user selects certain options at the assembly interface. This can be useful if the integration wants to override a particular HotDocs feature or command.
OnUserMenuItemClickedEvent	This event is fired when the user selects an integration-defined menu item. (See *Assembly.AddUserMenuItem* Method.)
PostSaveAnswersEvent	This event is fired after an answer file is saved.
PreSaveAnswersEvent	This event is fired after the user has indicated he or she wants to

save the answers, but before the actual save occurs. By setting *override == true, the integration can prevent the save from happening.

How do I program a COM Event in Visual C#?

1. Create a new project with a reference to the HotDocs Type Library in Visual Studio:
    1. Create a new Visual C# **Console Application** project.
    2. Right click on **References** from the Solution Explorer and **Add Reference**. The Add Reference dialog box appears.
    3. From the **COM type** libraries, select the **HotDocs 11 Type** Library, then click **OK**.
    4. Add a **`using HotDocs;`** statement to the project.
    5. Add a **`using System.Runtime.InteropServices;`** statement so you can release the COM object when finished with it.
2. Create a new Application object:
    - Within the static void Main method add:

```
Application app = new Application();
```

3. Create an event handler that will be called when the event is fired:
    1. Directly below the new Application object type **`app.OnAssemblyStartEvent +=`** and then **tab twice** to generate the method stub for displaying a message when the event fires. This will add the following to the static void Main method.

```
app.OnAssemblyStartEvent += app_OnAssemblyStartEvent;
```

    2. Comment out **`throw new NotImplementedException();`**
    3. Directly below that, add the following code to the newly created event:

```
Console.WriteLine("OnAssemblyStart");
```

4. Add code to start the assembly so you can see your event:

- Add the following code to the static void Main method, below the new application object:

```
string tplPath, tplTitle, tplDesc;
app.SelectTemplate("", true, out tplPath, out tplTitle, out
tplDesc);

if (tplPath != "")
    app.Assemblies.AddToQueue(tplPath);

Console.ReadLine();
```

5. Release the COM object:

- Add the following code to the end of the static void Main method, after the above code:

```
Marshal.ReleaseComObject(app);
```

6. Run the application.
7. Result:

The HotDocs SelectTemplate interface appears, allowing you to select a template from the library. When you select a template, it is added to the assembly queue. Then when the assembly starts, the *app.OnAssemblyStart* event fires and the console displays the message telling you that it fired.

Enumerations

DependencyType Enumeration

This enumeration is a set of values that determines what type of file is represented by a *Dependency* object.

HotDocs Directories

Name	Value	Description
NoDependency	0	
BaseCmpFileDependency	1	Base component file
PointedToCmpFileDependency	2	Pointed-to component file
TemplateInsertDependency	3	Template insertion
ClauseInsertDependency	4	Clause insertion
Clause library insertion	5	ClauseLibraryInsertDependency

Image insertion (INSERT \"filename\" /IMAGE)	6	ImageInsertDependency
InterviewImageDependency	7	An image included via a component
VariableTemplateInsertDependency	8	Variable template insertion (INSERT varname)
VariableImageInsertDependency	9	An image inserted via a variable (INSERT varname /IMAGE)
MissingVariableDependency	10	A variable was referenced for insertion, but does not exist.
MissingFileDependency	11	A file was referenced, but does not exist.
AssembleDependency	12	Template inclusion via ASSEMBLE statement
PublisherMapFileDependency	13	A publisher map file (.hdpmx)
UserMapFileDependency	14	A user map file (.hdumx)
AdditionalTemplateDependency	15	Additional template file specified in component properties.

HDAFFORMAT Enumeration

This enumeration is a set of values that determine which format HotDocs uses when saving an answer file.

All answer files are written in XML format by HotDocs 2009 and later. The difference between a "HotDocs 2009-11 Format" XML answer file and a "Pre-HotDocs 2009 Format" XML answer file is that HotDocs 2009-11 answer files use UTF-8 encoding and they do not contain an embedded DTD. If your answer files must be read by versions of HotDocs earlier than 2009, use the *PreHD2009Format*.

Answer File Formats

Name	Value	Description
HD_ANSFORMAT	1	Pre-HotDocs 2009 Format (Binary) - Deprecated
HD_ANXFORMAT	2	Pre-HotDocs 2009 Format (XML) - Deprecated
PreHD2009Format	2	Pre-HotDocs 2009 Format (XML)
HD2009Format	3	HotDocs 2009-11 Format (XML)

HDAIMENU Enumeration

This enumeration is a set of values for manipulating the menus in the HotDocs assembly window user interface.

Assembly Window Menus

Name	Value	Description
AI_FILE	1	The **File** menu in the assembly window.
AI_EDIT	2	The **Edit** menu in the assembly window.
AI_VIEW	3	The **View** menu in the assembly window.
AI_TOOLS	4	The **Tools** menu in the assembly window.
AI_FIELD	5	The **Field** menu in the assembly window.
AI_HELP	6	The **Help** menu in the assembly window.

HDANSWERUPLOADFORMAT Enumeration

This enumeration is a set of values to determine which format to use for answer files uploaded to a URL. It is used with the *AnswerCollection.UploadAnswerCollection* method.

Uploaded Answer Formats

Name	Value	Description
HD_DEFAULTFORMAT	1	Default binary file format
HD_XMLFORMAT	2	XML format used by XML (.ANX) answer files and HotDocs Server interviews (preferred because it is much easier to work with once it gets to the server)

HDASSEMBLYSTATUS Enumeration

This enumeration is a set of values for determining the status of an *Assembly* object.

Status

Name	Value	Description

HDASMSTATUSASSEMBLING	4	The *Assembly* object is being assembled.
HDASMSTATUSAUTOSELECTED	16	The *Assembly* object was automatically selected.
HDASMSTATUSCOMPLETED	8	The *Assembly* object has been assembled and the assembly completed with no errors.
HDASMSTATUSCONFIRMED	2	The *Assembly* object is awaiting assembly.
HDASMSTATUSERROR	256	Assembly is complete, but an error occurred during assembly and it did not complete successfully.
HDASMSTATUSPENDING	1	The *Assembly* object is pending.
HDASMSTATUSUNDEFINED	0	Undefined.
HDASMSTATUSUSERSELECTED	34	The *Assembly* object was selected by the user.

HDAUI Enumeration

This enumeration is a set of values for manipulating the HotDocs assembly window user interface.

Menus

Name	Value	Description
AUIFILEMENU	0	Enables (true) or disables (false) the **File** menu.
AUIEDITMENU	1	Enables (true) or disables (false) the **Edit** menu.
AUIVIEWMENU	2	Enables (true) or disables (false) the **View** menu.
AUITOOLSMENU	3	Enables (true) or disables (false) the **Tools** menu.
AUIFIELDMENU	4	Enables (true) or disables (false) the **Field** menu when assembling a form template with the form document tab selected.
AUIHELPMENU	5	Enables (true) or disables (false) the **Help** menu.
AUINAVIGATEMENU	63	Enables (true) or disables (false) the **Navigate** menu.

Tabs and Toolbars

Name	Value	Description
AUIINTERVIEWTAB	6	Shows (true) or hides (false) the **Interview** tab.
AUIPREVIEWTAB/AUIDOCUMENTTAB	7	Shows (true) or hides (false) the **Document Preview** (text

		templates) or **Form Document** (form templates) tab.
AUIVARIABLESHEETTAB	8	Shows (true) or hides (false) the **Variable Sheet** tab.
AUIQUESTIONSUMMARYTAB	9	Shows (true) or hides (false) the **Question Summary** tab.
AUIANSWERSUMMARYTAB	10	Shows (true) or hides (false) the **Answer Summary** tab.
AUIRESOURCEPANE	11	Reserved for future use.
AUIINTERVIEWOUTLINE	12	Shows (true) or hides (false) the interview outline.
AUIANSWERFILEDROPDOWN	13	Shows (true) or hides (false) the answer file drop-down list on the toolbar.
AUITOOLBAR	14	Shows (true) or hides (false) the standard toolbar.
AUISTATUSBAR	15	Shows (true) or hides (false) the status bar.
AUIEDITTOOLBAR	51	Shows (true) or hides (false) the edit toolbar.
AUIDIALOGTOOLBAR	52	Reserved for future use.
AUIFORMSTOOLBAR	53	Shows (true) or hides (false) the document (text templates) or forms (form templates) toolbar.
AUIDOCCOMPARETAB	61	Shows (true) or hides (false) the **Comparison** tab. This only applies when HotDocs Compare is installed, which is no longer available starting with the release of HotDocs 2009.
AUIDLGRESOURCEPANE	65	Shows (true) or hides (false) the resource pane from the Interview toolbar.
AUIFORMRESOURCEPANE	66	Shows (true) or hides (false) the resource pane from the Form Document toolbar.
AUIVARSHEETRESOURCEPANE	67	Shows (true) or hides (false) the resource pane from the Variable Sheet toolbar.

End of Interview Dialog

Name	Value	Description
AUIEOIGOTOFIRSTUNANSWERED	55	Shows (true) or hides (false) the **Go to the first unanswered question in the interview** command, when applicable, at the **End of Interview** dialog.
AUIEOIPASTETOWP	56	Shows (true) or hides (false) the **Paste the assembled document into the open word processor document** command (for text templates) at the **End of Interview** dialog.
AUIEOISENDTOCLIPBOARD	57	Shows (true) or hides (false) the **Copy the assembled document to the Clipboard** command at the **End of**

		Interview dialog. (This command is only available when assembling Microsoft Word documents.)
AUIEOIOPTIONS	58	Shows (true) or hides (false) the **Choose which buttons are displayed on this dialog** command at the **End of Interview** dialog.
AUIEOISENDTOFILLER	59	Shows (true) or hides (false) the **View the assembled form document at the Form Document tab** command (for form templates) at the **End of Interview** dialog.
AUIEOICLOSENOSAVE	60	Shows (true) or hides (false) the **Close this window without saving the assembled document** command at the **End of Interview** dialog.
AUIEOISENDTOOUTPUT	61	Shows (true) or hides (flase) the **Send to Output** command, when applicable, at the **End of Interview** dialog.

File Menu

Name	Value	Description
AUIFILENEWANSWERS	16	Enables (true) or disables (false) the **New Answers** item in the **File** menu, as well as the **New Answers** button on the toolbar.
AUIFILEOPENANSWERS	17	Enables (true) or disables (false) the **Open Answers** item in the **File** menu, as well as the **Open Answers** button and answer file drop-down list on the toolbar.
AUIFILEOVERLAYANSWERS	18	Enables (true) or disables (false) the **Overlay Answers** item in the **File** menu.
AUIFILESAVEANSWERS	19	Enables (true) or disables (false) the **Save Answers** item in the **File** menu, as well as the **Save Answers** button on the toolbar.
AUIFILESAVEANSWERSAS	20	Enables (true) or disables (false) the **Save Answers As** item in the **File** menu.
AUIFILESENDANSWERSTO	21	Enables (true) or disables (false) the **Send Answers To** item in the **File** menu.
AUIFILESAVEDOCUMENTAS	22	Enables (true) or disables (false) the **Save Document As** item in the **File** menu. It also shows or hides the **Save the assembled document in a file** command at the **End of Interview** dialog.
AUIFILESAVEQUESTIONSUMMARYAS	23	Enables (true) or disables (false) the **Save Question Summary As** item in the **File** menu (at the **Question Summary** tab).
AUIFILESAVEANSWERSUMMARYAS	24	Enables (true) or disables (false) the **Save Answer**

		Summary As item in the **File** menu (at the **Answer Summary** tab).
AUIFILESAVEVARIABLESHEETAS	25	Enables (true) or disables (false) the **Save Variable Sheet As** item in the **File** menu (at the **Variable Sheet** tab).
AUIFILESENDDOCUMENTTO	26	Shows (true) or hides (false) the **Send Document To** item in the **File** menu and the **Send Document** command (for text templates) at the **End of Interview** dialog. It also enables or disables the **Send Document** button on the toolbar. Finally, if the **End of Interview** action selected at **HotDocs Options** includes sending the assembled document to the word processor or HotDocs Filler before closing, HotDocs will simply close the assembly without sending the document.
AUIFILESENDQUESTIONSUMMARYTO	27	Shows (true) or hides (false) the **Send Question Summary To** item in the **File** menu (at the **Question Summary** tab).
AUIFILESENDANSWERSUMMARYTO	28	Shows (true) or hides (false) the **Send Answer Summary To** item in the **File** menu (at the **Answer Summary** tab).
AUIFILESENDVARIABLESHEETTO	29	Shows (true) or hides (false) the **Send Variable Sheet To** item in the **File** menu (at the **Variable Sheet** tab).
AUIFILESENDADDENDUMTO	30	Shows (true) or hides (false) the **Send Addendum To** item in the **File** menu (at the **Form Document** tab).
AUIFILEPAGESETUP	31	Enables (true) or disables (false) the **Page Setup** item in the **File** menu (at the **Question Summary** and **Answer Summary** tabs).
AUIFILEPRINTPREVIEW	32	Enables (true) or disables (false) the **Print Preview** item in the **File** menu (at the **Question Summary** and **Answer Summary** tabs).
AUIFILEPRINTDOCUMENT	33	Enables (true) or disables (false) the **Print Document** item in the **File** menu, as well as the **Print Document** button on the toolbar (at the **Form Document** tab).
AUIFILEPRINTQUESTIONSUMMARY	34	Enables (true) or disables (false) the **Print Question Summary** item in the **File** menu, as well as the **Print Question Summary** button on the toolbar (at the **Question Summary** tab).
AUIFILEPRINTANSWERSUMMARY	35	Enables (true) or disables (false) the **Print Answer Summary** item in the **File** menu, as well as the **Print Answer Summary** button on the toolbar (at the **Answer Summary** tab).
AUIFILEDOCUMENTPROPERTIES	36	Shows (true) or hides (false) the **Document Properties** item in the **File** menu (at the **Form Document** tab).

AUIFILECLOSE	37	Enables (true) or disables (false) the **Close** item in the **File** menu.
AUIFILESAVEDOCUMENTASPDF	54	Shows (true) or hides (false) the **Save the assembled document as a PDF** command at the **End of Interview** dialog.
AUIFILESENDCOMPARISONTO	64	This only applies when HotDocs Compare is installed, which is no longer available starting with the release of HotDocs 2009. Shows (true) or hides (false) the **Send Comparison To** item in the **File** menu (at the **Comparison** tab).
AUIFILEPRINTCOMPARISON	68	This only applies when HotDocs Compare is installed, which is no longer available starting with the release of HotDocs 2009. Enables (true) or disables (false) the **Print Comparison** item in the **File** menu, as well as the **Print Comparison** button on the toolbar (at the **Comparison** tab).
AUIFILESAVECOMPARISONAS	69	This only applies when HotDocs Compare is installed, which is no longer available starting with the release of HotDocs 2009. Enables (true) or disables (false) the **Save Comparison As** item in the **File** menu (at the **Comparison** tab).
AUIFILESELECTOPENANSWERS	71	Enables (true) or disables (false) the **Open Answer File** item in the **File** Menu.

View Menu

Name	Value	Description
AUIVIEWTOOLBARS	39	Enables (true) or disables (false) the **Toolbar** items (Standard, Edit, Form) in the **View** menu.
AUIVIEWSTATUSBAR	40	Enables (true) or disables (false) the **Status Bar** item in the **View** menu.
AUIVIEWQUESTIONSUMMARYTAB	41	Enables (true) or disables (false) the **Question Summary Tab** item in the **View** menu.
AUIVIEWANSWERSUMMARYTAB	42	Enables (true) or disables (false) the **Answer Summary Tab** item in the **View** menu.

AUIVIEWVARIABLESHEETTAB	43	Enables (true) or disables (false) the **Variable Sheet Tab** item in the **View** menu.
AUIVIEWINTERVIEWOUTLINE	44	Enables (true) or disables (false) the **Interview Outline** item in the **View** menu.
AUIVIEWRESOURCEPANE	45	Enables (true) or disables (false) the **Resource Pane** item in the **View** menu.
AUIVIEWDOCCOMPARETAB	62	Enables (true) or disables (false) the **Comparison** Tab item in the **View** menu. This only applies when HotDocs Compare is installed, which is no longer available starting with the release of HotDocs 2009.
AUIVIEWDIALOGNAVIGATIONBAR	74	Enables (true) or disables (false) the **Interview Navigation Bar** in the **View** menu.
AUIVIEWENDOFINTERVIEWDIALOG	75	Enables (true) or disables (false) the **End of Interview Dialog** in the **View** menu.
AUIVIEWEXPANDALL	77	Enables (true) or disables (false) the **Expand All** in the **View** menu.
AUIVIEWCOLLAPSEALL	78	Enables (true) or disables (false) the **Collapse All** in the **View** menu.
AUIVIEWPREVIEWTAB	80	Enables (true) or disables (false) the **Document Preview tab** in the **View** menu.

Tools Menu

Name	Value	Description
AUITOOLSOPTIONS	47	Enables (true) or disables (false) the **Options** item in the **Tools** menu, as well as the **HotDocs Options** button on the toolbar.
AUIASMQUEUE	76	Enables (true) or disables (false) the **Assembly Queue** in the **Tools** menu.

Help Menu

Name	Value	Description
AUIHELPHOTDOCSHELP	48	Enables (true) or disables (false) the **HotDocs Help** item in the **Help** menu.

AUIHELPONLINESUPPORT	49	Enables (true) or disables (false) the **Online Support** item in the **Help** menu.
AUIHELPABOUTHOTDOCS	50	Enables (true) or disables (false) the **About HotDocs** item in the **Help** menu.
AUIHELPONLINEREGISTRATION	70	Enables (true) or disables (false) the **Online Registration** item in the **Help** menu.

HDDirectory Enumeration

This enumeration is a set of folders for various HotDocs file types. It is used with the *Application.getDefaultPath* method.

HotDocs Directories

Name	Value	Description
DirAllUsersDocuments	12	Shared documents folder
DirAnswerFiles	7	Answer file folder (e.g., My Documents\HotDocs\Answers)
DirCatalogFiles	13	Catalog file folder (e.g., Public Documents\HotDocs\Catalogs)
DirFormDocuments	6	Form document folder (e.g., My Documents)
DirHelp	2	Help folder
DirJavaScript	1	JavaScript folder
DirLibraries	4	Library files folder (e.g., My Documents\HotDocs\Libraries)
DirMyDocuments	11	My Documents folder (e.g., My Documents)
DirProgramFiles	0	HotDocs Program Files folder (e.g., C:\Program Files\HotDocs 6)
DirPublishSettings	10	Publish settings folder (e.g., My Documents\HotDocs\Publish)
DirSpellingDictionary	3	Spelling dictionary folder
DirTemplateFiles	9	Template files folder (e.g., My Documents\HotDocs\Templates)
DirTemplateSets	5	Template sets folder (e.g., Public Documents\HotDocs\Templates)
DirUploadAnswerFiles	8	Upload answer files folder

HDLIMENU Enumeration

This enumeration is a set of values for manipulating the menus in the HotDocs library window user interface.

Name	Value	Description
LI_FILE	0	The **File** menu in the library window.
LI_EDIT	1	The **Edit** menu in the library window.
LI_VIEW	2	The **View** menu in the library window.
LI_TEMPLATE	3	The **Template** menu in the library window.
LI_TOOLS	4	The **Tools** menu in the library window.
LI_HELP	5	The **Help** menu in the library window.

HDLUI Enumeration

This enumeration is a set of values for manipulating the HotDocs library window user interface.

Tabs, Toolbars, and Context Menu

Name	Value	Description
LUIPROPERTYTAB	9	Shows (true) or hides (false) the **Properties** tab.
LUIPREVIEWTAB	10	Shows (true) or hides (false) the **Preview** tab.
LUITOOLBAR	7	Shows (true) or hides (false) the toolbar.
LUISTATUSBAR	8	Shows (true) or hides (false) the status bar.
LUICONTEXTMENU	11	Enables (true) or disables (false) the shortcut menu displayed when users right-click on items in the library.

File Menu

Name	Value	Description
LUIFILEMENU	1	Enables (true) or disables (false) the **File** menu.

LUIFILENEWLIBRARY	57600	Enables (true) or disables (false) the **New Library** item in the **File** menu, as well as the  **New Library** button on the toolbar.
LUIFILEOPENLIBRARY	57601	Enables (true) or disables (false) the **Open Library** item in the **File** menu, as well as the  **Open Library** button on the toolbar.
LUIFILESAVELIBRARY	57603	Enables (true) or disables (false) the **Save Library** item in the **File** menu.
LUIFILESAVELIBRARYAS	57604	Enables (true) or disables (false) the **Save Library As** item in the **File** menu.
LUIFILEPRINTPREVIEW	57609	Enables (true) or disables (false) the **Print Preview** item in the **File** menu.
LUIFILEPRINTLIBRARY	57607	Enables (true) or disables (false) the **Print Library** item in the **File** menu, as well as the  **Print Library** button on the toolbar.
LUIFILEINSTALLTEMPLATE	43059	Enables (true) or disables (false) the **Install Templates** item in the **File** menu.
LUIFILEIMPORTLIBRARY	43046	Enables (true) or disables (false) the **Import Library** item in the **File** menu.
LUIFILEEXPORTLIBRARY	12	Enables (true) or disables (false) the **Export Library To** items (HotDocs Library File, Plain Text File) in the **File** menu.
LUIFILEMRULIST	13	Shows (true) or hides (false) the list of Most Recently Used (MRU) library files in the **File** menu.
LUIFILEEXIT	57665	Enables (true) or disables (false) the **Exit** item in the **File** menu.

Edit Menu

Name	Value	Description
LUIEDITMENU	2	Enables (true) or disables (false) the **Edit** menu.
LUIEDITCUT	57635	Enables (true) or disables (false) the **Cut** item in the **Edit** menu, as well as the **Cut** button on the toolbar.
LUIEDITCOPY	57634	Enables (true) or disables (false) the **Copy** item in the **Edit** menu, as well as the  **Copy** button on the toolbar.
LUIEDITPASTE	57637	Enables (true) or disables (false) the **Paste** item in the **Edit** menu, as well as the  **Paste** button on the toolbar.
LUIEDITADD	43043	Enables (true) or disables (false) the **Add Item** and **Add Folder** items in the **Edit** menu, as well as the  **Add** button

		on the toolbar.
LUIEDITDELETE	43042	Enables (true) or disables (false) the **Delete** item in the **Edit** menu, as well as the **Delete** button on the toolbar.
LUIEDITSORT	43044	Enables (true) or disables (false) the **Sort** item in the **Edit** menu, as well as the **Sort** button on the toolbar.
LUIEDITPROPERTIES	43048	Enables (true) or disables (false) the **Properties** item in the **Edit** menu, as well as the **Properties** button on the toolbar.
LUIEDITMULTIPLEPROPERTIES	46209	Enables (true) or disables (false) the **Multiple** item in the **Edit** menu.

View Menu

Name	Value	Description
LUIVIEWMENU	3	Enables (true) or disables (false) the **View** menu.
LUIVIEWTOOLBAR	59392	Enables (true) or disables (false) the **Toolbar** item in the **View** menu.
LUIVIEWSTATUSBAR	59393	Enables (true) or disables (false) the **Status Bar** item in the **View** menu.
LUIVIEWPROPERTIESTAB	43068	Enables (true) or disables (false) the **Properties Tab** item in the **View** menu.
LUIVIEWPREVIEWTAB	43069	Enables (true) or disables (false) the **Preview Tab** item in the **View** menu.
LUIVIEWTITLES	46399	Enables (true) or disables (false) the **Template Titles** item in the **View** menu.
LUIVIEWFILENAMES	46398	Enables (true) or disables (false) the **File Names** item in the **View** menu.
LUIVIEWEXPANDALL	46400	Enables (true) or disables (false) the **Expand All** item in the **View** menu.
LUIVIEWCOLLAPSEALL	46401	Enables (true) or disables (false) the **Collapse All** item in the **View** menu.
LUIVIEWTABSATTOP	46349	Enables (true) or disables (false) the **Tabs at Top** item in the **View** menu.
LUIVIEWMARKUPVIEW	46431	Enables (true) or disables (false) the **Markup View** item in the **View** menu.

Template Menu

Name	Value	Description
LUITEMPLATEMENU	4	Enables (true) or disables (false) the **Template** menu.
LUITEMPLATEASSEMBLE	43052	Enables (true) or disables (false) the **Assemble** item in the **Template** menu, as well as the **Assemble** button on the toolbar.
LUITEMPLATEONLINETEST	46207	Enables (true) or disables (false) the **Test in Browser** item in the **Template** menu.
LUITEMPLATECREATE	43054	Enables (true) or disables (false) the **New** item in the **Template** menu, as well as the **New Template** button on the toolbar.
LUITEMPLATEEDIT	43053	Enables (true) or disables (false) the **Edit** item in the **Template** menu, as well as the **Edit** button on the toolbar.
LUITEMPLATECOPY	46300	Enables (true) or disables (false) the **Copy** item in the **Template** menu.
LUITEMPLATEMOVE	43047	Enables (true) or disables (false) the **Move** item in the **Template** menu.
LUITEMPLATEPRINT	46396	Enables true) or disables (false) the **Print item** in the **Template** menu.
LUITEMPLATECONVERTTOMODEL	46480	Enables (true) or disables (false) the **Create Model from Template...** item in the **Template** menu.
LUITEMPLATECONVERTFROMMODEL	46479	Enables (true) or disables (false) the **Create Template from Model...** item in the **Template** menu.

Tools Menu

Name	Value	Description
LUITOOLSMENU	5	Enables (true) or disables (false) the **Tools** menu.
LUITOOLSANSWERFILEMANAGER	43070	Enables (true) or disables (false) the **Answer File Manager** item in the **Tools** menu, as well as the **Answer File Manager** button on the toolbar.
LUITOOLSCOMPONENTMANAGER	43100	Enables (true) or disables (false) the **Component Manager** item in the **Tools** menu, as well as the **Component Manager** button on the toolbar.
LUITOOLSCOMPONENTEXPLORER	43063	Enables (true) or disables (false) the **Template Manager** item in the **Tools** menu, as well as the **Template Manager** button on the toolbar. (HotDocs Developer only.)
LUITOOLSPUBLISHINGWIZARD	43051	Enables (true) or disables (false) the **Publishing Wizard**

		item in the **Tools** menu, as well as the  **Publishing Wizard** button on the toolbar. (HotDocs Developer only.)
LUITOOLSUPLOADANSWERS	43060	Enables (true) or disables (false) the **Upload Answers** item in the **Tools** menu.
LUITOOLSREFRESHHDACACHE	43050	Enables (true) or disables (false) the **Refresh Cache** item in the **Tools** menu.
LUIVIEWASSEMBLYQUEUE	14	Enables (true) or disables (false) the **Assembly Queue** item in the **Tools** menu, as well as the  **Assembly Queue** button on the toolbar.
LUITEMPLATESETUPDATE	46417	Enables (true) or disables (false) the **Update Template Sets** item in the **Tools** menu.
LUITOOLSOPTIONS	43010	Enables (true) or disables (false) the **Options** item in the **Tools** menu.
LUITOOLSAUTOMATOR	46233	Enables (true) or disables (false) the **HotDocs Automator** item in the **Tools** menu.
LUIHIDDENDATAREMOVER	46427	Enables (true) or disables (false) the **Hidden Data Remover...** item in the **Tools** menu.

Help Menu

Name	Value	Description
LUIHELPMENU	6	Enables (true) or disables (false) the **Help** menu.
LUIHELPHOTDOCSHELP	57667	Enables (true) or disables (false) the **HotDocs Help** item in the **Help** menu.
LUIHELPONLINESUPPORT	43058	Enables (true) or disables (false) the **Online Support** item in the **Help** menu.
LUIHELPONLINEREGISTRATION	46287	Enables (true) or disables (false) the **Online Registration** item in the **Help** menu.
LUIHELPABOUTHOTDOCS	57664	Enables (true) or disables (false) the **About HotDocs** item in the **Help** menu.

HDMappingBackfill Enumeration

This enumeration is a set of modes for answer backfilling.

Backfill Modes

Name	Value	Description
Always	1	Always backfills answers.
DoNotAllow	3	Does not allow backfilling answers.
Never	0	Never backfills answers.
Prompt	2	Prompts to backfill answers.

HDOUTPUTTYPE Enumeration

This enumeration is a set of values that determine which format HotDocs uses when saving documents.

Document Type

Name	Value	Description
HD_OUTPUT_DOCUMENT	1	Document
HD_OUTPUT_ANSWERSUMMARY	2	Answer summary
HD_OUTPUT_QUESTIONSUMMARY	3	Question summary

HDPRODUCTFLAVOR Enumeration

This enumeration is a set of values that determines which HotDocs edition, or flavor, is running.

Product Flavors

Name	Value	Description
PLAYER	1	HotDocs Player
STANDARD	2	HotDocs Developer LE
PROFESSIONAL	3	HotDocs Developer
USER	4	HotDocs User

HDServerFileType Enumeration

This enumeration is a set of values that determine which file type HotDocs uses when publishing files for use with HotDocs Server.

Name	Value	Description
HDServerFilesJavaScript	2	JavaScript or .HVC file
HDServerFilesNone	0	No file
HDServerFilesSilverlight	4	Compiled Silverlight assembly
HDServerFilesTemplate	1	Template or component file

HDVARTYPE Enumeration

This enumeration is a set of values for component types in HotDocs. It is used in many places in the HotDocs API, although not all values can be used in all places. (Some values are invalid depending on where the value is used.)

Name	Value	Description
HD_ADDITIONALTEXT	12	Additional text
HD_CLAUSELIBTYPE	9	Clause library
HD_COMPUTATIONTYPE	6	Computation variable
HD_DATABASETYPE	8	Database
HD_DATEFORMAT	17	Date Format
HD_DATETYPE	3	Date variable
HD_DIALOG	13	Dialog component
HD_DOCUMENTTEXT	11	SPAN component text
HD_GROUPFORMAT	20	Group Format component
HD_MULTCHOICEFORMAT	19	Multiple Choice Format component
HD_MULTCHOICETYPE	5	Multiple Choice variable

HD_NUMBERFORMAT	16	Number Format component
HD_NUMBERTYPE	2	Number variable
HD_TEXTFORMAT	15	Text Format
HD_TEXTPATTERN	14	Text Pattern
HD_TEXTTYPE	1	Text variable
HD_TRUEFALSEFORMAT	18	True/False Format
HD_TRUEFALSETYPE	4	True/False variable
HD_UNANSWEREDTYPE	7	Unanswered (for values only)
HD_UNDEFINED	10	Undefined (indicates an error)

HotDocs.Answer Object

HotDocs.Answer Object

An *Answer* is the set of *Values* for a particular HotDocs variable. For example, the *Answer* for a non-repeated variable consists of a single piece of data (text string, number, date, or other information) stored in a single Value. The *Answer* for a repeated variable, however, consists of multiple Values—one for each time the variable is repeated.

There are several ways to iterate through the *Values* of an *Answer*. If you know the repeat indexes for the *Value* you want to retrieve, you can simply set the 0-based repeat indexes of the *Answer* to those indexes. Thus, if you want to retrieve the second *Value* of a simple repeated variable, you could set the *Answer's* repeat indexes to **1,-1,-1,-1** . (Unused indexes should be set to **-1** .) Once these indexes are set, retrieving the *Answer's Value* will return the data stored in the Value with the specified repeat indexes.

General Information

ProgID:	HotDocs.Answer.11.0 HotDocs.Answer (version-independent)
CLSID:	{4D509C2B-0223-47EB-95A5-CC8E98055F67}

The following table shows the name and IID for each interface, as well as the version of HotDocs in which it was introduced. The primary interface and the main public interface exposed by this object is *_Answer*.

Name	IID	Added in
_Answer	{DF34C5CA-1760-4262-9B56-	Added in HotDocs 6.0

	24E3EDE60994}	
_AnswerEvents	{E485B97D-1BBF-4174-A97C-4B03D5EB33C2}	Added in HotDocs 6.0

The *_AnswerEvents* interface designates an event sink interface that an application must implement in order to receive event notifications from a *HotDocs.Answer* object.

Methods

Method	Description
AddMultipleChoiceValue	This method adds an additional value to a particular Multiple Choice answer iteration.
ClearAskedFlag	This method does nothing. (It was deprecated in HotDocs 6.01.)
Create	This method creates a new HotDocs answer.
GetRepeatCount	This method returns the number of repeated values at a certain level of the repeat.
GetRepeatIndex	This method returns the repeat index information for the current value.
IsMultipleChoiceValueSet	This method indicates if a particular option in a Multiple Choice variable is selected.
IterateValues	This method starts a recursive descent of the values for an answer. It visits each value for an answer and fires the *Answer.OnValueFound* event. In many cases, this saves you (the integrator) from iterating through combinations of repeat indices to find values in an answer tree.
SetRepeatIndex	This method sets the current value of the *Answer* object to the value at the specified repeat indexes.

Properties

Property	Description
Application	[Read-only] This property returns a reference to the *Application* object.
Name	[Read-only] This property returns the name of the answer. This is a read-only property because the Name is set using the *Answer.Create* method and cannot be changed after the answer is created.
RepeatCount	[Read-only] This property returns the number of repeated values at the current level. The difference between this property and the *Answer.GetRepeatCount* method is that this property returns the repeat count for the current level only. The *GetRepeatCount* method returns the repeat count for a level specified by a set of repeat indexes given in the method call.

Type	[Read-only] This property returns the type of the *Answer* object.
Unanswered	[Read/Write] This is a Boolean property that determines whether an answer is marked as "Unanswered." If so, *Unanswered* clears the value at that repeat level.
Value	[Read/Write] This is a dynamic property that holds the value of the answer at the current repeat index.

Events

Event	Description
OnValueFoundEvent	This event is fired for every value found for the particular answer. This event is fired during the processing of the *Answer.IterateValues* method.

Example

The following **Visual C#** example opens an answer file and displays each answer it contains:

```
public class ExampleCode
{
    static string ansName;
    static string ansType;

    static void Main()
    {
        HotDocs.AnswerCollection ac = new HotDocs.AnswerCollection();
        ac.Create(@"C:\Documents\HotDocs\Answers\AnswerFile.anx");

        Console.WriteLine("Variable\tAnswer\tType");

        foreach (HotDocs.Answer ans in ac)
        {
            ansName = ans.Name;
            ansType = getType(ans.Type);
            ans.OnValueFoundEvent += new
HotDocs._AnswerEvents_OnValueFoundEventEventHandler(ans_OnValueFoundEvent);
            ans.IterateValues();
        }
        Console.ReadKey();

        ac.Close();
    }

    static void ans_OnValueFoundEvent(object Value, int repeat1, int repeat2,
int repeat3, int repeat4)
    {
        string index = "";
        string ansValue = "";
        if (repeat1 > -1) index = "[" + (repeat1 + 1).ToString() + "] ";
```

```
        //Check to see if the value is an array (multiple-select Multiple
Choice variable)
        if (Value.GetType() == typeof(System.Object[]))
            foreach (string s in (System.Object[])Value)
            {
                ansValue += s + "; ";
            }
            else
                ansValue = Value.ToString();

            Console.WriteLine(index + ansName + "\t" + ansValue + "\t" +
ansType);
    }

    static string getType(HotDocs.HDVARTYPE type)
    {
        switch (type)
        {
            case HotDocs.HDVARTYPE.HD_DATETYPE:
                return "Date";
            case HotDocs.HDVARTYPE.HD_MULTCHOICETYPE:
                return "Multiple Choice";
            case HotDocs.HDVARTYPE.HD_NUMBERTYPE:
                return "Number";
            case HotDocs.HDVARTYPE.HD_TEXTTYPE:
                return "Text";
            case HotDocs.HDVARTYPE.HD_TRUEFALSETYPE:
                return "True/False";
            default:
                return type.ToString();
        }
    }
}
```

Answer.AddMultipleChoiceValue Method

This method adds an additional value to a particular Multiple Choice answer iteration.

Only Multiple Choice variables specified as **Select All That Apply** will accept multiple values for the same repeat iteration.

Syntax

```
void AddMultipleChoiceValue ( string newValue)
```

Parameter	Description
newValue	The option text for the newly-selected option. If this text is not one of the

option text values in the component file, the value will be stored in the answer file, but it will never be used because only values in the component file are valid options.

Example

The following **Visual C#** example adds additional values to a Multiple Choice answer and displays a message box with all answers for the variable:

```
public class ExampleCode
{
    static void Main()
    {
        HotDocs.AnswerCollection asc = new HotDocs.AnswerCollection();
        HotDocs.Answer ans = new HotDocs.Answer();

        asc.Create(@"C:\Documents\HotDocs\Answers\AnswerFile.anx");
        HotDocs.HDVARTYPE vType = HotDocs.HDVARTYPE.HD_MULTCHOICETYPE;
        ans = asc.Item("Company Representative", ref vType);
        ans.AddMultipleChoiceValue("Ed Hall");
        ans.AddMultipleChoiceValue("Kim Schuster");

        object[] vals = (object[])ans.Value;
        string msg = "";
        for (int i = 0; i < vals.Length; i++)
        {
            msg = msg + vals[i].ToString() + "\r\n";
        }
        MessageBox.Show(msg);

        asc.Save(@"C:\Documents\HotDocs\Answers\AnswerFile.anx");

        asc.Close();
    }
}
```

Answer.ClearAskedFlag Method

This method does nothing. (It was deprecated in HotDocs 6.01.)

Syntax

```
void ClearAskedFlag ( )
```

Answer.Create Method

This method creates a new HotDocs answer.

Syntax

```
void Create ( string ansName, HotDocs.HDVARTYPE valType)
```

Parameter	Description
ansName	The name of the variable for this answer.
valType	The variable type for this answer. This can be one of the following values from the HDVARTYPE enumeration: • HD_TEXTTYPE • HD_NUMBERTYPE • HD_DATETYPE • HD_TRUEFALSETYPE • HD_MULTCHOICETYPE

Example

The following **Visual C#** example creates a new answer, assigns it a value, and adds it to an answer file:

```
public class ExampleCode
{
    static void Main()
    {
        HotDocs.AnswerCollection asc = new HotDocs.AnswerCollection();
        HotDocs.Answer ans = new HotDocs.Answer();

        asc.Create(@"C:\Documents\HotDocs\Answers\AnswerFile.anx");
        ans.Create("Employee Name", HotDocs.HDVARTYPE.HD_TEXTTYPE);
        ans.Value = "John Simpson";
        asc.Add(ans);
        asc.Save(@"C:\Documents\HotDocs\Answers\AnswerFile.anx");

        asc.Close();
    }
}
```

Answer.GetRepeatCount Method

This method returns the number of repeated values at a certain level of the repeat.

Syntax

```
int GetRepeatCount ( int repeat1, int repeat2, int repeat3, int repeat4 )
```

Parameter	Description
repeat1	First repeat index for the specified value.
repeat2	Second repeat index for the specified value.
repeat3	Third repeat index for the specified value.
repeat4	Fourth repeat index for the specified value.

Return Value

The repeat count for the specified value.

The *RepeatCount* property returns the repeat count at the current level; however, this method returns the repeat count at an arbitrary level without changing the repeat indexes for the *Answer* object.

Any unused repeat indexes must be set to **-1**. For instance, if you want to reference the fourth iteration of the first level of a repeated answer, you would use **3,-1,-1,-1** as the repeat index.

Example

The following **Visual C#** example displays the number of repeated values at the first repeat level:

```
public class ExampleCode
{
    static void Main()
    {
        HotDocs.AnswerCollection asc = new HotDocs.AnswerCollection();
        HotDocs.Answer ans = new HotDocs.Answer();
        HotDocs.HDVARTYPE vType = HotDocs.HDVARTYPE.HD_TEXTTYPE;

        asc.Create(@"C:\Documents\HotDocs\Answers\AnswerFile.anx");
        ans = asc.Item("Beneficiary Name", ref vType);
        MessageBox.Show(ans.GetRepeatCount(0, -1, -1, -1).ToString());

        asc.Close();
    }
}
```

Answer.GetRepeatIndex Method

This method returns the repeat index information for the current value.

Syntax

```
void GetRepeatIndex ( out int repeat1, out int repeat2, out int repeat3, out int
repeat4 )
```

Parameter	Description
repeat1	First repeat index for current value.
repeat2	Second repeat index for current value.
repeat3	Third repeat index for current value.
repeat4	Fourth repeat index for current value.

Example

The following **Visual C#** example sets the repeat indices for an answer and then displays the indices using *GetRepeatIndex* :

```
public class ExampleCode
{
    static void Main()
    {
        HotDocs.AnswerCollection asc = new HotDocs.AnswerCollection();
        HotDocs.Answer ans = new HotDocs.Answer();
        HotDocs.HDVARTYPE vType = HotDocs.HDVARTYPE.HD_TEXTTYPE;
        int rpt1, rpt2, rpt3, rpt4;

        asc.Create(@"C:\Documents\HotDocs\Answers\AnswerFile.anx");
        ans = asc.Item("Beneficiary Name", ref vType);
        ans.SetRepeatIndex(1, 0, 0, 0);

        ans.GetRepeatIndex(out rpt1,out rpt2,out rpt3,out rpt4);
        MessageBox.Show(rpt1 + ", " + rpt2 + ", " + rpt3 + ", " + rpt4);

        asc.Close();
    }
}
```

Answer.IsMultipleChoiceValueSet Method

This method indicates if a particular option in a Multiple Choice variable is selected.

For example, if you have a Multiple Choice variable with options red, blue, and yellow, this method could be used to determine if "yellow" has been selected.

Syntax

```
bool IsMultipleChoiceValueSet ( string chkValue )
```

Parameter	Description
chkValue	The option text for the option.

Return Value

A Boolean value indicating if the option is selected.

Example

The following **Visual C#** example indicates whether or not a particular Multiple Choice option is selected in an answer file:

```
public class ExampleCode
{
    static void Main()
    {
        HotDocs.AnswerCollection asc = new HotDocs.AnswerCollection();
        HotDocs.Answer ans = new HotDocs.Answer();
        HotDocs.HDVARTYPE vType = HotDocs.HDVARTYPE.HD_MULTCHOICETYPE;

        asc.Create(@"C:\Documents\HotDocs\Answers\Color.anx");
        ans = asc.Item("ColorMC", ref vType);

        MessageBox.Show(ans.IsMultipleChoiceValueSet("yellow").ToString());
    }
}
```

Answer.IterateValues Method

This method starts a recursive descent of the values for an answer. It visits each value for an answer and fires the *Answer.OnValueFound* event. In many cases, this saves you (the integrator) from iterating through combinations of repeat indices to find values in an answer tree.

Syntax

```
void IterateValues ( )
```

Example

The following **Visual C#** example opens an answer file and displays each answer it contains:

```
public class ExampleCode
{
    static string ansName;
    static string ansType;

    static void Main()
    {
        HotDocs.AnswerCollection ac = new HotDocs.AnswerCollection();
        ac.Create(@"C:\Documents\HotDocs\Answers\AnswerFile.anx");

        Console.WriteLine("Variable\tAnswer\tType");

        foreach (HotDocs.Answer ans in ac)
        {
            ansName = ans.Name;
            ansType = getType(ans.Type);
            ans.OnValueFoundEvent += new
HotDocs._AnswerEvents_OnValueFoundEventEventHandler(ans_OnValueFoundEvent);
            ans.IterateValues();
        }
        Console.ReadKey();

        ac.Close();
    }

    static void ans_OnValueFoundEvent(object Value, int repeat1, int repeat2,
int repeat3, int repeat4)
    {
        string index = "";
        string ansValue = "";
        if (repeat1 > -1) index = "[" + (repeat1 + 1).ToString() + "] ";
        //Check to see if the value is an array (multiple-select Multiple
Choice variable)
        if (Value.GetType() == typeof(System.Object[]))
            foreach (string s in (System.Object[])Value)
            {
                ansValue += s + "; ";
            }
            else
                ansValue = Value.ToString();

            Console.WriteLine(index + ansName + "\t" + ansValue + "\t" +
ansType);
    }

    static string getType(HotDocs.HDVARTYPE type)
    {
        switch (type)
        {
```

```
            case HotDocs.HDVARTYPE.HD_DATETYPE:
                return "Date";
            case HotDocs.HDVARTYPE.HD_MULTCHOICETYPE:
                return "Multiple Choice";
            case HotDocs.HDVARTYPE.HD_NUMBERTYPE:
                return "Number";
            case HotDocs.HDVARTYPE.HD_TEXTTYPE:
                return "Text";
            case HotDocs.HDVARTYPE.HD_TRUEFALSETYPE:
                return "True/False";
            default:
                return type.ToString();
        }
    }
}
```

Answer.SetRepeatIndex Method

This method sets the current value of the *Answer* object to the value at the specified repeat indexes.

Syntax

```
void SetRepeatIndex ( int repeat1, int repeat2, int repeat3, int repeat4 )
```

Parameter	Description
repeat1	First repeat index for the desired value.
repeat2	Second repeat index for the desired value.
repeat3	Third repeat index for the desired value.
repeat4	Fourth repeat index for the desired value.

Any unused repeat indexes must be set to **-1**. For instance, if you want to reference the fourth iteration of the first level of a repeated answer, you would use **3,-1,-1,-1** as the repeat index.

Example

The following **Visual C#** example uses SetRepeatIndex to add two names and e-mail addresses to an answer file:

```
public class ExampleCode
{
    static void Main()
    {
        HotDocs.AnswerCollection asc = new HotDocs.AnswerCollection();
```

```
        HotDocs.Answer ans = new HotDocs.Answer();

        asc.Create(@"C:\Documents\HotDocs\Answers\AnswerFile.anx");
        asc.FileFormat = HotDocs.HDAFFORMAT.HD2009Format;
        HotDocs.HDVARTYPE vType = HotDocs.HDVARTYPE.HD_TEXTTYPE;

        ans = asc.Item("Name", ref vType);
        ans.SetRepeatIndex(0, -1, -1, -1);
        ans.Value = "John";
        asc.Add(ans);

        ans = asc.Item("Email", ref vType);
        ans.SetRepeatIndex(0, 0, -1, -1);
        ans.Value = "John@beatles.com";
        asc.Add(ans);

        ans = asc.Item("Name", ref vType);
        ans.SetRepeatIndex(1, -1, -1, -1);
        ans.Value = "Paul";
        asc.Add(ans);

        ans = asc.Item("Email", ref vType);
        ans.SetRepeatIndex(1, 0, -1, -1);
        ans.Value = "Paul@beatles.com";
        asc.Add(ans);

        asc.Save(@"C:\Documents\HotDocs\Answers\AnswerFile.anx");

        asc.Close();

        ans = null;
        asc = null;
    }
}
```

Answer.Application Property

[Read-only] This property returns a reference to the *Application* object.

Since there is only one *Application* object on a machine at a time, this property will return a reference to the same object as the *Application* property on any other object in HotDocs, and a reference to the same object as if you created a new *HotDocs.Application* object.

Syntax

```
HotDocs._Application2 Application [ get ]
```

Answer.Name Property

[Read-only] This property returns the name of the answer. This is a read-only property because the Name is set using the *Answer.Create* method and cannot be changed after the answer is created.

Syntax

```
string Name [ get ]
```

Answer.RepeatCount Property

[Read-only] This property returns the number of repeated values at the current level. The difference between this property and the *Answer.GetRepeatCount* method is that this property returns the repeat count for the current level only. The *GetRepeatCount* method returns the repeat count for a level specified by a set of repeat indexes given in the method call.

Syntax

```
int RepeatCount [ get ]
```

Answer.Type Property

[Read-only] This property returns the type of the *Answer* object.

The *Type* may be any one of the following values from the HDVARTYPE enumeration:

- HD_TEXTTYPE
- HD_NUMBERTYPE
- HD_DATETYPE
- HD_TRUEFALSETYPE
- HD_MULTCHOICETYPE
- HD_DOCUMENTTEXT

- HD_CLAUSELIBTYPE
- HD_DATABASETYPE

Syntax

```
HotDocs.HDVARTYPE Type [ get ]
```

Answer.Unanswered Property

[Read/Write] This is a Boolean property that determines whether an answer is marked as "Unanswered." If so, *Unanswered* clears the value at that repeat level.

This property is **True** in the following situations:

- Repeated values. Placeholder values are put in the answer file in cases where not all the values at a repeat level are answered. For example if a repeated value is answered at positions 1, 2, and 4 then position 3 will have an unanswered value.
- *AnswerCollections* that have not yet been written to disk. You can create any unanswered values you want in memory. This could happen through an interview or manipulation of the *AnswerCollection*.

You can use the *Unanswered* property to delete values from the *AnswerCollection*. If you set this property to **True**, then the next time the *AnswerCollection* is saved to disk, no answer will be saved for that value.

Syntax

```
bool Unanswered [ set, get ]
```

Answer.Value Property

[Read/Write] This is a dynamic property that holds the value of the answer at the current repeat index.

Syntax

```
dynamic Value [ set, get ]
```

The data type can be interpreted like this:

Type	Data Type	Description
HD_TEXTTYPE	string	Multi-line Text answers will contain carriage return (cr) and line feed (lf) characters to delimit lines.
HD_NUMBERTYPE	number	Number value.
HD_DATETYPE	dateTime	Date value.
HD_TRUEFALSETYPE	boolean	True/False (Boolean) value.
HD_MULTCHOICETYPE	string array | string	Multiple Choice values can be interpreted several ways, depending on whether the Multiple Choice variable is specified as single-select or multiple-select: • When retrieving the single value of a Multiple Choice variable, the value will be returned as a string. • When retrieving multiple values, the value will be returned as an array of strings.

Example (Visual C#)

```
A multi-line value can be set by adding the carriage return and line feed
characters as follows:
class ExampleCode
{
    static void Main()
    {
        HotDocs.AnswerCollection asc = new HotDocs.AnswerCollection();
        HotDocs.Answer ans = new HotDocs.Answer();

        asc.Create(@"C:\temp\AnswersFile.anx");
        HotDocs.HDVARTYPE vType = HotDocs.HDVARTYPE.HD_MULTCHOICETYPE;
        ans = asc.Item("multiplechoice", ref vType);

        MessageBox.Show(ans.Value = "Line 1" + "\r\n" + "Line 2" + "\r\n" +
"Line 3");

        asc.Close();
    }
}

If you want to retrieve multiple values from a multiple-select Multiple
Choice variable,
you can retrieve an array of strings like this:

        ....
        for (int i = 0; i < ans.Value.Length; i++)
        {
            MessageBox.Show(ans.Value[i]);
        }
```

To set a single value of a Multiple Choice variable, you can use the *Value* property and a string (**VT_BSTR**),
or you can use the *AddMultipleChoiceValue* method:

```
....
ans.Value = "Single Value";

//or
ans.AddMultipleChoiceValue("Single Value");
```

To set multiple values of a multiple-select Multiple Choice variable, you can set *Value* using a
single string that contains values delimited by a vertical bar (|) character, an array of strings,
or you can use a series of calls to the *AddMultipleChoiceValue* method:

```
....
ans.Value = "FirstValue|SecondValue|ThirdValue";

//or
string[] values = new string[3];

values[0] = "FirstValue";
values[1] = "SecondValue";
values[2] = "ThridValue";
ans.Value = values;

//or
ans.AddMultipleChoiceValue("FirstValue");
ans.AddMultipleChoiceValue("SecondValue");
ans.AddMultipleChoiceValue("ThirdValue");
```

Answer.OnValueFoundEvent Event

This event is fired for every value found for the particular answer. This event is fired during the processing of the *Answer.IterateValues* method.

Syntax

```
OnValueFoundEvent (value, repeat1, repeat2, repeat3, repeat4)
```

Parameter	Description
VARIANT value	The value found. This VARIANT is interpreted using the same rules as for the Value property.
long* repeat1	First repeat index for current value.
long* repeat2	Second repeat index for current value.

long* repeat3	Third repeat index for current value.
long* repeat4	Fourth repeat index for current value.

Example

The following **Visual C#** example opens an answer file and displays each answer it contains:

```
public class ExampleCode
{
    static string ansName;
    static string ansType;

    static void Main()
    {
        HotDocs.AnswerCollection ac = new HotDocs.AnswerCollection();
        ac.Create(@"C:\Documents\HotDocs\Answers\AnswerFile.anx");

        Console.WriteLine("Variable\tAnswer\tType");

        foreach (HotDocs.Answer ans in ac)
        {
            ansName = ans.Name;
            ansType = getType(ans.Type);
            ans.OnValueFoundEvent += new
HotDocs._AnswerEvents_OnValueFoundEventEventHandler(ans_OnValueFoundEvent);
            ans.IterateValues();
        }
        Console.ReadKey();

        ac.Close();
    }

    static void ans_OnValueFoundEvent(object Value, int repeat1, int repeat2,
int repeat3, int repeat4)
    {
        string index = "";
        string ansValue = "";
        if (repeat1 > -1) index = "[" + (repeat1 + 1).ToString() + "] ";
        //Check to see if the value is an array (multiple-select Multiple
Choice variable)
        if (Value.GetType() == typeof(System.Object[]))
            foreach (string s in (System.Object[])Value)
            {
                ansValue += s + "; ";
            }
            else
                ansValue = Value.ToString();

            Console.WriteLine(index + ansName + "\t" + ansValue + "\t" +
ansType);
    }

    static string getType(HotDocs.HDVARTYPE type)
```

```
    {
        switch (type)
        {
            case HotDocs.HDVARTYPE.HD_DATETYPE:
                return "Date";
            case HotDocs.HDVARTYPE.HD_MULTCHOICETYPE:
                return "Multiple Choice";
            case HotDocs.HDVARTYPE.HD_NUMBERTYPE:
                return "Number";
            case HotDocs.HDVARTYPE.HD_TEXTTYPE:
                return "Text";
            case HotDocs.HDVARTYPE.HD_TRUEFALSETYPE:
                return "True/False";
            default:
                return type.ToString();
        }
    }
}
```

HotDocs.AnswerCollection Object

HotDocs.AnswerCollection Object

This object represents sets of answers, which most frequently come from HotDocs answer files saved on disk. However, this is not the only way answers are stored—an *AnswerCollection* object can represent answers that were input from an interview, retrieved from a database, or imported from any other source you tell your program to use.

AnswerCollection objects are also used elsewhere in the HotDocs API to specify a set of answers to use in an assembly, and when responding to events fired from the *Application* and *Assembly* objects.

General Information

ProgID:	HotDocs.AnswerCollection.11.0 HotDocs.AnswerCollection (version-independent)
CLSID:	{F6B3FF63-D730-4DCE-802D-0FAED25E7B72}

The following table shows the name and IID for each interface, as well as the version of HotDocs in which it was introduced. The primary interface and the main public interface exposed by this object is *_AnswerCollection2*.

Name	IID	Added in
_AnswerCollection	{3E419C82-EED2-4FD4-BD37-C4BC9A9FEFB1}	Added in HotDocs 6.0

_AnswerCollection2	{A9BED2DE-BAE7-4BF9-93D4-E944E81A6F8E}	Added in HotDocs 2005 SP2

Methods

Method	Description
Add	This method adds a new answer to the *AnswerCollection*. This answer must have already been initialized using the *Create* method.
Close	If an *AnswerCollection* is backed by an answer file on disk, the answer file is kept open while the *AnswerCollection* object is in use. This method closes the answer file on disk.
Create	This method initializes the *AnswerCollection* object. You must call this method before using a new *AnswerCollection* object.
Item	This method retrieves an *Answer* object from the *AnswerCollection*.
Overlay	This method overlays the filename answer file on top of the existing answer set.
Save	This method saves the *AnswerCollection* to a HotDocs answer file. The file type (.ANS or .ANX) is determined by the *FileFormat* property.
UploadAnswerCollection	This method uploads the *AnswerCollection* to the specified location (url).

Properties

Property	Description
Application	[Read-only] This property returns a reference to the *Application* object.
Count	[Read-only] This property returns the number of *Answer* objects in the *AnswerCollection*.
DefaultAnswerFile	[Read/Write] This property returns the default answer file used by the *AnswerCollection*. Using a default answer file is similar to using an overlay answer file except that the answers are underlayed into the answer set. (For details, see Create a Default Answer File.)
Description	[Read/Write] This property sets or returns the description of the *AnswerCollection*.
FileFormat	[Read/Write] This property determines the answer file format.
FileName	[Read-only] This property returns the file name of the answer file represented by the *AnswerCollection* object. To specify the file path for the answer file, pass the file path and name to either the *Create* or *Save* method.
Modified	[Read-only] This Boolean property indicates if an *AnswerCollection* has been modified or not.

Title	[Read/Write] This property sets or returns the title of the *AnswerCollection*.
XML	[Read-only] This property returns the *AnswerCollection* as an XML string.

Example

The following **Visual C#** example opens an answer file and displays each answer it contains:

```
public class ExampleCode
{
    static string ansName;
    static string ansType;

    static void Main()
    {
        HotDocs.AnswerCollection ac = new HotDocs.AnswerCollection();
        ac.Create(@"C:\Documents\HotDocs\Answers\AnswerFile.anx");

        Console.WriteLine("Variable\tAnswer\tType");

        foreach (HotDocs.Answer ans in ac)
        {
            ansName = ans.Name;
            ansType = getType(ans.Type);
            ans.OnValueFoundEvent += new
HotDocs._AnswerEvents_OnValueFoundEventEventHandler(ans_OnValueFoundEvent);
            ans.IterateValues();
        }
        Console.ReadKey();

        ac.Close();
    }

    static void ans_OnValueFoundEvent(object Value, int repeat1, int repeat2,
int repeat3, int repeat4)
    {
        string index = "";
        string ansValue = "";
        if (repeat1 > -1) index = "[" + (repeat1 + 1).ToString() + "] ";
        //Check to see if the value is an array (multiple-select Multiple
Choice variable)
        if (Value.GetType() == typeof(System.Object[]))
            foreach (string s in (System.Object[])Value)
            {
                ansValue += s + "; ";
            }
            else
                ansValue = Value.ToString();

            Console.WriteLine(index + ansName + "\t" + ansValue + "\t" +
ansType);
    }
```

```
    static string getType(HotDocs.HDVARTYPE type)
    {
        switch (type)
        {
            case HotDocs.HDVARTYPE.HD_DATETYPE:
                return "Date";
            case HotDocs.HDVARTYPE.HD_MULTCHOICETYPE:
                return "Multiple Choice";
            case HotDocs.HDVARTYPE.HD_NUMBERTYPE:
                return "Number";
            case HotDocs.HDVARTYPE.HD_TEXTTYPE:
                return "Text";
            case HotDocs.HDVARTYPE.HD_TRUEFALSETYPE:
                return "True/False";
            default:
                return type.ToString();
        }
    }
}
```

AnswerCollection.Add Method

This method adds a new answer to the *AnswerCollection* . This answer must have already been initialized using the *Create* method.

Syntax

```
void Add ( HotDocs.Answer newanswer )
```

Parameters	Description
newanswer	The *Answer* object to be added to the collection.

Example

The following **Visual C#** example adds two answers to an answer file, then saves and closes it.

```
public class ExampleCode
{
    static void Main()
    {
        HotDocs.AnswerCollection ac = new HotDocs.AnswerCollection();
        HotDocs.Answer ans;

        ac.Create(@"C:\Documents\HotDocs\Answers\AnswerFile.anx");
```

```
        ans = new HotDocs.Answer();
        ans.Create("Agreement Date", HotDocs.HDVARTYPE.HD_DATETYPE);
        ans.Value = "01/01/2009";
        ac.Add(ans);

        ans = new HotDocs.Answer();
        ans.Create("Company Representative",
HotDocs.HDVARTYPE.HD_MULTCHOICETYPE);
        ans.Value = "Stephanie Walker";
        ac.Add(ans);
        ans = null;

        ac.Save(@"C:\Documents\HotDocs\Answers\AnswerFile.anx");

        ac.Close();
    }
}
```

AnswerCollection.Close Method

If an *AnswerCollection* is backed by an answer file on disk, the answer file is kept open while the *AnswerCollection* object is in use. This method closes the answer file on disk.

Syntax

```
void Close ( )
```

Example

The following **Visual C#** example creates a new *AnswerCollection* object for a given answer file and displays the number of answers stored in the file:

```
public class ExampleCode
{
    static void Main()
    {
        HotDocs.AnswerCollection asc = new HotDocs.AnswerCollection();
        asc.Create(@"C:\Documents\HotDocs\Answers\AnswerFile.anx");
        Console.WriteLine("The answer file has " + asc.Count.ToString() + "
answers.");

        asc.Close();
    }
}
```

AnswerCollection.Create Method

This method initializes the *AnswerCollection* object. You must call this method before using a new *AnswerCollection* object.

Syntax

```
void Create ( string answerFileName)
```

Parameters	Description
answerFileName	The path for the answer file needed to start an assembly. If this file does not exist, a new file will be created. If the path does exist then the file will be opened. If this parameter is an empty string ("") then no file will be used and the *AnswerCollection* will only exist as long as it is in memory. An *AnswerCollection* object created in this manner can later be saved to a file using the *Save* method.

Example

The following **Visual C#** example creates a new *AnswerCollection*object for a given answer file and displays the number of answers stored in the file:

```
public class ExampleCode
{
    static void Main()
    {
        HotDocs.AnswerCollection asc = new HotDocs.AnswerCollection();
        asc.Create(@"C:\Documents\HotDocs\Answers\AnswerFile.anx");
        Console.WriteLine("The answer file has " + asc.Count.ToString() + "
answers.");

        asc.Close();
    }
}
```

AnswerCollection.Item Method

This method retrieves an Answer object from the *AnswerCollection* .

The index parameter can either be a number or a string. If it is a number, then the index *Answer* object in the collection will be retrieved and the vartype parameter will be set to the HDVARTYPE value for the

answer. If index is a string, it is interpreted as an Answer name and the Answer object with the name matching index and the type vartype. HotDocs answers are identified by their name and type, so vartype must be set to the correct HDVARTYPE when calling this method with a string for the index parameter.

Syntax

```
HotDocs.Answer Item ( object index, ref HotDocs.HDVARTYPE varType )
```

Parameters	Description
index	Either a number representing the position of the desired *Answer* object in the collection, or a string representing the name of the desired *Answer* object. If index is a string, then vartype must be set correctly also.
vartype	[optional] When calling *Item()* with index representing a string, vartype must be the correct HDVARTYPE for the desired *Answer* object. When the method returns, this parameter will be set to the correct HDVARTYPE for the *Answer* object in pItem.

Return Value

The *Answer* object requested. If the *Answer* object could not be located, a new *Answer* object is created with an **Unanswered** value.

AnswerCollection.Overlay Method

This method overlays the filename answer file on top of the existing answer set.

Syntax

```
void Overlay ( string overlayFileName )
```

Parameters	Description
overlayFileName	The path for the answer file to be overlayed on top of the answer set.

Return Value

An integer indicating the index of the new *Value* within the *Answer* object.

AnswerCollection.Save Method

This method saves the *AnswerCollection* to a HotDocs answer file. The file type (.ANS or .ANX) is determined by the *FileFormat* property.

Syntax

```
void Save ( string answerFileName )
```

Parameters	Description
answerFileName	[optional] The path and file name for the resulting answer file. If the answer file already exists, it will be overwritten. If the file name was passed into the *Create* method, this parameter is not necessary.

Example

The following **Visual C#** example adds two answers to an answer file, then saves and closes it.

```
public class ExampleCode
{
    static void Main()
    {
        HotDocs.AnswerCollection ac = new HotDocs.AnswerCollection();
        HotDocs.Answer ans;

        ac.Create(@"C:\Documents\HotDocs\Answers\AnswerFile.anx");

        ans = new HotDocs.Answer();
        ans.Create("Agreement Date", HotDocs.HDVARTYPE.HD_DATETYPE);
        ans.Value = "01/01/2009";
        ac.Add(ans);

        ans = new HotDocs.Answer();
        ans.Create("Company Representative",
HotDocs.HDVARTYPE.HD_MULTCHOICETYPE);
        ans.Value = "Stephanie Walker";
        ac.Add(ans);
        ans = null;

        ac.Save(@"C:\Documents\HotDocs\Answers\AnswerFile.anx");

        ac.Close();
    }
}
```

AnswerCollection.UploadAnswerCollection Method

This method uploads the *AnswerCollection* to the specified location (url).

Executing this method is similar to specifying a URL in the **Upload Answers** component file property. (For details, see the HotDocs Desktop Help File.)

Syntax

```
void UploadAnswerCollection ( string url, HotDocs.HDANSWERUPLOADFORMAT format )
```

Parameters	Description
url	The destination for the uploaded answers. It can be an HTTP or HTTPS location.
format	The format for the uploaded answers. It can be one of the following values from the HDANSWERUPLOADFORMAT enumeration: • HD_DEFAULTFORMAT: the delimited format used by HotDocs 5. • HD_XMLFORMAT: the XML format used by HotDocs ANX answer files and HotDocs Server interviews. It is the preferred format because it is much easier to work with once it gets to the server.

AnswerCollection.Application Property

[Read-only] This property returns a reference to the *Application* object.

Since there is only one *Application* object on a machine at a time, this property will return a reference to the same object as the *Application* property on any other object in HotDocs, and a reference to the same object as if you created a new *HotDocs.Application* object.

Syntax

```
HotDocs._Application2 Application [ get ]
```

AnswerCollection.Count Property

This property returns the number of *Answer* objects in the *AnswerCollection* .

Syntax

```
int Count [ get ]
```

Example

The following Visual C# example creates a new *AnswerCollection* object for a given answer file and displays the file name, format (type), title, description, and number of answers stored in the file:

```
public class ExampleCode
{
    static void Main()
    {
        HotDocs.AnswerCollection asc = new HotDocs.AnswerCollection();

        asc.Create(@"C:\Documents\HotDocs\Answers\AnswerFile.anx");

        Console.WriteLine("File: {0}\n", asc.FileName);
        Console.WriteLine("Format (Type): {0}\n", asc.FileFormat);
        Console.WriteLine("Title: {0}\n", asc.Title);
        Console.WriteLine("Description: {0}\n", asc.Description);
        Console.WriteLine("Count: {0}\n", asc.Count);

        asc.Close();
    }
}
```

AnswerCollection.DefaultAnswerFile Property

[Read/Write] This property returns the default answer file used by the *AnswerCollection*. Using a default answer file is similar to using an overlay answer file except that the answers are underlayed into the answer set. (For details, see Create a Default Answer File.)

Syntax

```
string DefaultAnswerFile [ set, get ]
```

AnswerCollection.Description Property

[Read/Write] This property sets or returns the description of the *AnswerCollection*.

Syntax

```
string Description [ set, get ]
```

Example

The following Visual C# example creates a new *AnswerCollection* object for a given answer file and displays the file name, format (type), title, description, and number of answers stored in the file:

```
public class ExampleCode
{
    static void Main()
    {
        HotDocs.AnswerCollection asc = new HotDocs.AnswerCollection();

        asc.Create(@"C:\Documents\HotDocs\Answers\AnswerFile.anx");

        Console.WriteLine("File: {0}\n", asc.FileName);
        Console.WriteLine("Format (Type): {0}\n", asc.FileFormat);
        Console.WriteLine("Title: {0}\n", asc.Title);
        Console.WriteLine("Description: {0}\n", asc.Description);
        Console.WriteLine("Count: {0}\n", asc.Count);

        asc.Close();
    }
}
```

AnswerCollection.FileFormat Property

[Read/Write] This property determines the answer file format.

The *FileFormat* may be one of the following values from the HDAFFORMAT enumeration:

- PreHD2009Format
- HD2009Format

All answer files are written in XML format by HotDocs 2009 and later. The difference between a "HotDocs 2009-10 Format" XML answer file and a "Pre-HotDocs 2009 Format" XML answer file is that HotDocs 2009-10 answer files use UTF-8 encoding and they do not contain an embedded DTD. If your answer files must be read by versions of HotDocs earlier than 2009, use the PreHD2009Format.

Syntax

```
HotDocs.HDAFFORMAT FileFormat [ set, get ]
```

Example

The following Visual C# example creates a new *AnswerCollection* object for a given answer file and displays the file name, format (type), title, description, and number of answers stored in the file:

```
public class ExampleCode
{
    static void Main()
    {
        HotDocs.AnswerCollection asc = new HotDocs.AnswerCollection();

        asc.Create(@"C:\Documents\HotDocs\Answers\AnswerFile.anx");

        Console.WriteLine("File: {0}\n", asc.FileName);
        Console.WriteLine("Format (Type): {0}\n", asc.FileFormat);
        Console.WriteLine("Title: {0}\n", asc.Title);
        Console.WriteLine("Description: {0}\n", asc.Description);
        Console.WriteLine("Count: {0}\n", asc.Count);

        asc.Close();
    }
}
```

AnswerCollection.FileName Property

[Read-only] This property returns the file name of the answer file represented by the *AnswerCollection* object. To specify the file path for the answer file, pass the file path and name to either the *Create* or *Save* method.

Syntax

```
string FileName [ get ]
```

Example

The following Visual C# example creates a new *AnswerCollection* object for a given answer file and displays the file name, format (type), title, description, and number of answers stored in the file:

```
public class ExampleCode
{
    static void Main()
```

```
    {
        HotDocs.AnswerCollection asc = new HotDocs.AnswerCollection();

        asc.Create(@"C:\Documents\HotDocs\Answers\AnswerFile.anx");

        Console.WriteLine("File: {0}\n", asc.FileName);
        Console.WriteLine("Format (Type): {0}\n", asc.FileFormat);
        Console.WriteLine("Title: {0}\n", asc.Title);
        Console.WriteLine("Description: {0}\n", asc.Description);
        Console.WriteLine("Count: {0}\n", asc.Count);

        asc.Close();
    }
}
```

AnswerCollection.Modified Property

[Read-only] This Boolean property indicates if an *AnswerCollection* has been modified or not.

Syntax

```
bool Modified [ get ]
```

Example

The following Visual C# example shows the value of the *Modified* property before and after making a change to one of the answers in the collection.

```
public class ExampleCode
{
    static void Main()
    {
        HotDocs.AnswerCollection asc = new HotDocs.AnswerCollection();
        HotDocs.HDVARTYPE iType = HotDocs.HDVARTYPE.HD_TEXTTYPE;

        asc.Create(@"C:\Documents\HotDocs\Answers\AnswerFile.anx ");
        asc.FileFormat = HotDocs.HDAFFORMAT.HD2009Format;
        Console.WriteLine("Current Modified Status (before change):
{0}", asc.Modified); asc.Item("Employee Name", ref iType).Value = "Wilma
Hernandez";
        Console.WriteLine("Current Modified Status (after change):
{0}", asc.Modified);

        asc.Close();
    }
}
```

AnswerCollection.Title Property

[Read/Write] This property sets or returns the title of the *AnswerCollection*.

Syntax

```
string Title [ set, get ]
```

Example

The following Visual C# example creates a new *AnswerCollection* object for a given answer file and displays the file name, format (type), title, description, and number of answers stored in the file:

```
public class ExampleCode
{
    static void Main()
    {
        HotDocs.AnswerCollection asc = new HotDocs.AnswerCollection();

        asc.Create(@"C:\Documents\HotDocs\Answers\AnswerFile.anx");

        Console.WriteLine("File: {0}\n", asc.FileName);
        Console.WriteLine("Format (Type): {0}\n", asc.FileFormat);
        Console.WriteLine("Title: {0}\n", asc.Title);
        Console.WriteLine("Description: {0}\n", asc.Description);
        Console.WriteLine("Count: {0}\n", asc.Count);

        asc.Close();
    }
}
```

AnswerCollection.XML Property

[Read-only] This property returns the *AnswerCollection* as an XML string.

Syntax

```
string XML [ get ]
```

Example

The following Visual C# example displays the answer file as an XML string.

```
public class ExampleCode
{
    static void Main()
    {
        HotDocs.AnswerCollection asc = new HotDocs.AnswerCollection();
        asc.Create(@"C:\Documents\HotDocs\Answers\AnswerFile.anx");

        Console.WriteLine(asc.XML);

        asc.Close();
    }
}
Here is what the XML string might look like:
<?xml version="1.0" encoding="Windows-1252" standalone="yes"?>
<AnswerSet title = "Jane Doe" version = "1.1">
    <Answer name = "Agreement Date">
        <RptValue>
            <DateValue>3/8/2008</DateValue>
            <DateValue>2/8/2008</DateValue>
            <DateValue unans = "true"/>
        </RptValue>
    </Answer>
    <Answer name = "Employee Gender">
        <MCValue>
            <SelValue>Male</SelValue>
        </MCValue>
    </Answer>
    <Answer name = "(ANSWER FILE DESCRIPTION)">
        <TextValue></TextValue>
    </Answer>
    <Answer name = "(ANSWER FILE HISTORY)">
        <TextValue>Last Will and Testament : January 31, 2008,
15:24</TextValue>
    </Answer>
    <Answer name = "Employee Name">
        <TextValue>Raylene Schofield</TextValue>
    </Answer>
</AnswerSet>
```

HotDocs.Application Object

HotDocs.Application Object

The *Application* object represents the library window in HotDocs and the main functional uses of HotDocs. It is used to customize the user interface of the library window and control the display of the library. It also serves as the root of the HotDocs object model.

The *Application* class is a singleton class, which means that one and only one instance can exist on a Windows desktop at a time. The class factory for this class enforces this rule. When an application requests an instance of this class, the class factory checks to see if an instance already exists. If so, the requesting application is given a pointer to this existent instance. This means that an integration needs to be aware that other applications or the user may be changing the state of the object. For instance, care must be taken when deleting all the *Assembly* objects in the assembly queue so that your integration does not remove assemblies started by the user or other applications.

Reference counting of the *Application* object can be a tricky concept. Because of its shared-singleton nature, an integration cannot assume that the server (hotdocs.exe) will unload when the integration releases all of the references it holds to the COM objects. There are many clients which hold references to the *Application* object. The most visible of these is the user interface. The library interface is a client of the *Application* object. When it is visible, it holds a reference to the *Application* object. When it is hidden, it releases its reference. This means that if your program makes the HotDocs library interface visible, then exits, HotDocs will continue to run because the interface still holds a COM reference. To make sure that things are in an appropriate state when your application is finished with HotDocs, you must make sure the state of the user interface is the same as when you started.

General information

ProgID:	HotDocs.Application.11.0 HotDocs.Application (version-independent)
CLSID:	{8A202ADA-F14D-4F1B-86F9-8B18EE76E0C1}

The following table shows the name and IID for each interface, as well as the version of HotDocs in which it was introduced. The primary interface and the main public interface exposed by this object is _Application4.

Name	IID	Added in
_Application	{991DE9DD-D19A-4EA0-9A07-D56F0CA44FE9}	Added in HotDocs 6.0
_Application2	{991DE9DD-D19A-4EA0-9A07-D56F0CA44FEA}	Added in HotDocs 6.1 SP1
_Application3	{991DE9DD-D19A-4EA0-9A07-D56F0CA44FEB}	Added in HotDocs 2005
_Application4	{991DE9DD-D19A-4EA0-9A07-D56F0CA44FEC}	Added in HotDocs 2005 SP2
_Application5	{5C438478-E1F3-47ac-9012-EB497A45B704}	Added in HotDocs 2008
_Application6	{D5CC34E3-13FE-46eb-88BE-B00DC4924505}	Added in HotDocs 10
_Application7	{FCE63E5C-BA73-413b-9331-19B8FC556061}	Added for HotDocs 10.2
_Application8	{9DA30382-4BA9-44B6-BD51-D8E2D28E2E9D}	Added for HotDocs 11.0
_ApplicationEvents	{287BF4B6-F8A1-4D96-B9A6-	Added in HotDocs 6.0

D1F6A56AB86C}

The _ApplicationEvents interface designates an event sink interface that an application must implement in order to receive event notifications from a *HotDocs.Application* object.

Methods

Method	Description
AddUserMenuItem	This method allows you to add custom menu items to the HotDocs library menus. For example, you can add an item to the HotDocs Help menu to display your own "About" dialog box. When a user selects the item from the menu, the *OnUserMenuItemClickedEvent* event is fired to notify your application that the menu item was chosen.
AddUserMenuItem2	This method allows you to add custom menu items to the HotDocs library menus. For example, you can add an item to the HotDocs Help menu to display your own "About" dialog box. When a user selects the item from the menu, the *OnUserMenuItemClickedEvent* event is fired to notify your application that the menu item was chosen.
ConvertModelToTemplate	This method converts a HotDocs Model to a HotDocs template.
ConvertTemplateToModel	This method converts a HotDocs template to a HotDocs Model.
CreateTemplatePackage	This method creates a template package for use with HotDocs Cloud Services or the HotDocs Open SDK.
DeleteUserMenuItem	This method allows you to remove custom items that were added to the HotDocs library window menus using *AddUserMenuItem* or *AddUserMenuItem2*. It uses the handle returned when the menu item was added to determine which menu item to remove.
getDefaultPath	This method returns the default path HotDocs uses for the specified file type. You can use this method to determine the default path for any type of file in the *HDDirectory* enumeration.
GetHotDocsSetting	This method returns a HotDocs setting from the system registry. HotDocs looks first in the HKEY_CURRENT_USER registry hive, followed by the HKEY_LOCAL_MACHINE hive. If the setting is not found in either hive, the defaultValue is returned.
OpenLibrary	This method opens a HotDocs library (.HDL) file. If another library is already open, *OpenLibrary* closes the other library and opens the library specified in the *libPath* parameter.
PrintDocument	This method prints the specified document.
PublishOnlineFiles	This method publishes a template for use with HotDocs Server. Like the **Publishing Wizard** (available in the HotDocs **Tools** menu), this method scans the template for any inserted templates and builds the JavaScript

	(.JS) and HotDocs Variable Collection (.HVC) files required for HotDocs Server. These .JS and .HVC files are then copied to an output folder along with the template files.
PublishOnlineFiles2	Use this method to publish a template for use with HotDocs Server.
ResolveReferencePath	This method converts a reference path to a full file system path. For example, if you use *SelectTemplate2* to get the path of a selected template that includes a reference path (e.g., ^PUBTest\template.rtf), this method can look up the reference path keyword and return a full path (e.g., C:\HotDocs\Templates\template.rtf).
RetrieveUrlFile	This method retrieves a file from the specified URL.
SaveDocAsPDF	This method converts a document file to a PDF file using HotDocs PDF Advantage. The conversion is done automatically, so no user intervention is necessary, although some user interface may be displayed.
SelectMultipleTemplates	This method opens the specified library in a modal dialog box, allowing users to select multiple templates by pressing **Shift** or **Ctrl** as they click templates. The selected templates are then passed back to the integrating program in a SAFEARRAY structure.
SelectMultipleTemplates2	This method opens the specified library in a modal dialog box, allowing users to select multiple templates by pressing **Shift** or **Ctrl** as they click templates. The selected templates are then passed back to the integrating program in a SAFEARRAY structure.
SelectTemplate	This method opens a modal dialog that displays the specified library, allowing the user to select a template. The path, title, and description of the selected template are then passed back to the integrating program.
SelectTemplate2	This method opens the specified library in a modal dialog box, allowing users to select a single template. The path, title, and description of the selected template are then passed back to the integrating program.
SendToWordProcessor	This method sends a document specified in the *docFileName* parameter to the word processor.
SetUserInterfaceItem	This method sets the state for various features (elements) of the HotDocs library window. For example, you can use this method to disable features your integration users should not have access to, or you can enable features users may have disabled.

Properties

Property	Description
ActiveAssembly	[Read-only] This property returns an *Assembly* object representing the template currently being assembled.

Assemblies	[Read-only] This property returns an *AssemblyCollection* object, which is the collection of *Assembly* objects in the HotDocs assembly queue. By querying the *Assemblies* property, you can get all of the *Assembly* objects that are queued for assembly.
AssemblyQueueVisible	[Read/Write] This Boolean property controls the visibility status of the HotDocs assembly queue. For example, if the *AssemblyQueueVisible* property is False, the assembly queue is not visible.
CanAssembleAll	[Read-only] This property indicates whether or not the version of HotDocs in use is capable of assembling unregistered templates. Specifically, this property returns **true** if the version of HotDocs is anything other than Player.
CanEditTemplates	[Read-only] This property indicates whether or not the version of HotDocs in use is capable of editing templates. Specifically, this property returns **true** if the version of HotDocs is Developer or Developer LE.
CommandLine	[Write-only] This property sets the HotDocs command line options as if it were started with a particular command line. Setting this property to a string is the same as if the string were passed to the executable when the program was started. For example, if the command line invokes an assembly, a new *Assembly* object is added to the queue. If the command line changes the appearance or behavior of HotDocs, the change happens immediately.
CurrentLibraryPath	[Read-only] This property returns the file system path and file name of the current (open) HotDocs library as a String value
Flavor	[Read-only] This property returns a value corresponding to which HotDocs edition (Player, User, Developer, or Developer LE) is being used.
Hwnd	[Read-only] This property returns the window handle of the HotDocs library window.
Plugins	[Read-only] This property returns a *PluginsClass* object, which represents a collection of plug-ins currently registered with HotDocs.
Version	[Read-only] This property returns the HotDocs product version number as a String value. For example, if you have HotDocs 11 installed, this property returns **11**.
Visible	[Read/Write] This Boolean property controls the visibility status of the HotDocs library window. For example, if the Visible property is False, the library window is not visible.

Events

Event	Description
AssemblyCompleteEvent	This event is fired when assembly completes.
OnAssemblyCompleteEvent	This event has been deprecated for HotDocs Desktop 11.

	AssemblyCompleteEvent is recommended for use instead. This event is fired when an assembly is completed. It returns the name and path of the template that was used to assemble the document, the path to the assembled document, a pointer to the *AnswerCollection* object used in the assembly, and the assembly handle which was given when the assembly was added to the queue.
OnAssemblyStartEvent	This event is fired when an assembly starts. It returns a reference to the *Assembly* object that represents the assembly session.
OnErrorEvent	This event is fired when an error occurs. By returning true for the override parameter, the integration can tell HotDocs not to display any user interface indicating that an error occurred, which allows the integration to either display its own error message or silently handle the error.
OnLibraryInterfaceCloseEvent	This event is fired when the user closes the HotDocs library user interface.
OnLibraryOpenEvent	This event is fired when a library is opened.
OnTemplateSelectedEvent	This event is fired when the user selects a template in the library to assemble, or selects a template at the *SelectTemplate* or *SelectMultipleTemplate* dialogs. By returning *override == true, the integration can cancel the selection of the template.
OnUserInterfaceEvent	This event is fired when the user selects items in the library user interface.
OnUserMenuItemClickedEvent	This event is fired when the user selects an integration-defined menu item.

Application.AddUserMenuItem Method

This method allows you to add custom menu items to the HotDocs library menus. For example, you can add an item to the HotDocs Help menu to display your own "About" dialog box. When a user selects the item from the menu, the *OnUserMenuItemClickedEvent* event is fired to notify your application that the menu item was chosen.

The *AddUserMenuItem2* method is very similar and performs the same basic task as this method. However, the *AddUserMenuItem2* method gives you more control over the custom menu item. For example, you can specify the position of the item in the menu and the icon that appears next to the item.

Syntax

```
int AddUserMenuItem ( string menuTxt, HotDocs.HDLIMENU menu )
```

Parameters	Description
menuTxt	The text to be inserted into the menu. To insert a separator bar in the menu, use a single hyphen (-).
menu	The name of the HotDocs menu to which the menu item will be added. This must be a member of the *HDLIMENU* enumeration.

Return Value

A menu handle used to track when a user clicks the custom menu item.

This method only manipulates the menus in the HotDocs library window. To add items to the menus in the assembly window, see the *Assembly.AddUserMenuItem* method. You can also create HotDocs plug-ins that further customize menus in the library window.

Example

The following Visual C# example adds a separator bar followed by a menu entry to the File menu in the HotDocs library window:

```
public class ExampleCode
{
    static void Main()
    {
        HotDocs.Application app = new HotDocs.Application();
        long handle;

        app.AddUserMenuItem("-", HotDocs.HDLIMENU.LI_FILE);
        handle = app.AddUserMenuItem("User Menu Entry #1",
HotDocs.HDLIMENU.LI_FILE);
    }
}
```

Application.AddUserMenuItem2 Method

This method allows you to add custom menu items to the HotDocs library menus. For example, you can add an item to the HotDocs Help menu to display your own "About" dialog box. When a user selects the item from the menu, the *OnUserMenuItemClickedEvent* event is fired to notify your application that the menu item was chosen.

The *AddUserMenuItem* method is very similar and performs the same basic task as this method. However, the *AddUserMenuItem* method does not allow you to specify the position of the item in the menu or the icon that appears next to the item.

This method was introduced with the release of HotDocs 2005.

Syntax

```
int AddUserMenuItem2 ( string menuTxt, HotDocs.HDLIMENU menu, int position, Icon Icon
)
```

Parameters	Description
menuTxt	The text to be inserted into the menu. To insert a separator bar in the menu, use a single hyphen (-).
menu	The name of the HotDocs menu to which the menu item will be added. This must be a member of the HDLIMENU enumeration.
position	This is the position in the menu where the custom menu item will be added.
Icon	This is the icon that appears next to the item in the menu.

Return Value

A menu handle used to track when a user clicks the custom menu item.

This method only manipulates the menus in the HotDocs library window. To add items to the menus in the assembly window, see the *Assembly.AddUserMenuItem* method. You can also create HotDocs plug-ins that further customize menus in the library window.

Example

The following Visual C# example adds a menu item to the File menu in the HotDocs library window with your chosen icon. This example can only be used in conjunction with a Plugin:

```
public class ExampleCode
{
    static void Main()
    {
        HotDocs.Application app = new HotDocs.Application();

        //Important: HotDocs.Icon can only be used in conjunction with a
Plugin.
        HotDocs.Icon icon = new HotDocs.Icon();
        icon.LoadIcon(@"C:\images\UserMenuIcon.ico");
```

```
        app.AddUserMenuItem2("User Menu Entry #1", HDLIMENU.LI_FILE, 5,
icon);

        Marshal.ReleaseComObject(icon);
        Marshal.ReleaseComObject(app);
    }
}
```

Application.ConvertModelToTemplate Method

This method converts a HotDocs Model to a HotDocs template.

This method was introduced with the release of HotDocs 2008.

Syntax

```
string ConvertModelToTemplate( string modelDocumentPath, ref string TemplatePath,
bool hotdocsDisplaysMessages )
```

Parameters	Description
modelDocumentPath	The file name and path of the HotDocs Model to convert.
TemplatePath	The file name and path of the template to create from the HotDocs Model.
hotdocsDisplaysMessages	Indicates if Hotdocs will display messages during the conversion.

Example

The following Visual C# example converts a HotDocs Model to a template:

```
public class ExampleCode
{
    static void Main()
    {
        HotDocs.Application app = new HotDocs.Application();
        string mdPath = @"c:\temp\modeldocument1.rtf";
        string tpPath = @"c:\temp\template1.rtf";
        app.ConvertModelToTemplate(mdPath, ref tpPath, true);
    }
}
```

Application.ConvertTemplateToModel Method

This method converts a HotDocs template to a HotDocs Model.

This method was introduced with the release of HotDocs 2008.

Syntax

```
string ConvertTemplateToModel( string TemplatePath, ref string modelDocumentPath,
bool hotdocsDisplaysMessages )
```

Parameters	Description
TemplatePath	The file name and path of the template to convert.
modelDocumentPath	The file name and path of the HotDocs Model to create from the template.
hotdocsDisplaysMessages	Indicates if HotDocs will display messages during the conversion.

Example

The following Visual C# example converts a HotDocs template to a HotDocs Model:

```
public class ExampleCode
{
    static void Main()
    {
        HotDocs.Application app = new HotDocs.Application();
        string tpPath = @"c:\temp\template1.rtf";
        string mdPath = @"c:\temp\modeldocument1.rtf";
        app.ConvertTemplateToModel(tpPath, ref mdPath, true);
    }
}
```

Application.CreateTemplatePackage Method

This method creates a template package for use with HotDocs Cloud Services or the HotDocs Open SDK. A template package is single compressed file containing a template and all its dependencies – component files, template manifests, browser interview support files (JavaScript and Silverlight), additional inserted or assembled templates, graphics, etc.. Template packages also contain a package manifest that describes the contents and structure of the package itself.

Syntax

```
void CreateTemplatePackage(string templatePath, string packagePath, HdServerFileType
fileTypes, out string manifest)
```

Parameters	Description
templatePath	The full path and file name of the template to be packaged.
packagePath	The full path and file name where the finished package should be created.
fileTypes	The types of support files to generate and embed in the package. You must specify either HDServerFilesJavaScript alone, or HDServerFilesJavaScript | HDServerFilesSilverlight.
manifest	When the method returns, this parameter will contain the package manifest (XML) that was produced. This manifest was embedded in the package itself, but its contents are returned here for convenience.

Example

The following C# example creates a template package suitable for serving both JavaScript and Silverlight browser interviews:

```
public class ExampleCode
{
    static void Main()
    {
        HotDocs.Application app = new HotDocs.Application();

        HotDocs.HDServerFileType fileTypes =
HotDocs.HDServerFileType.HDServerFilesJavaScript |
HotDocs.HDServerFileType.HDServerFilesSilverlight;
        string templatePath = @"C:\temp\Demo Editor List.docx";
        string packagePath = @"C:\temp\Demo Editor List.pkg";
        string manifest;

        app.CreateTemplatePackage(templatePath, packagePath, fileTypes, out
manifest);
    }
}
```

Application.DeleteUserMenuItem Method

This method allows you to remove custom items that were added to the HotDocs library window menus using *AddUserMenuItem* or *AddUserMenuItem2*. It uses the handle returned when the menu item was added to determine which menu item to remove.

Syntax

```
void DeleteUserMenuItem ( int uiHandle )
```

Parameters	Description
uiHandle	This is the handle of the menu item that was returned when the item was created using *AddUserMenuItem*.

Example

The following Visual C# example adds a separator bar followed by a menu entry to the File menu in the HotDocs Library interface. Later (usually somewhere else in your code), the item is removed, which leaves the separator bar as the last item in the menu and HotDocs deletes it automatically.

```
public class ExampleCode
{
    static void Main()
    {
        HotDocs.Application app = new HotDocs.Application();
        int handle;

        app.AddUserMenuItem("-", HotDocs.HDLIMENU.LI_FILE);
        handle = app.AddUserMenuItem("User Menu Entry #1",
HotDocs.HDLIMENU.LI_FILE);

        // Somewhere else in the program...
        app.DeleteUserMenuItem(handle);
    }
}
```

Separator bars added using the *AddUserMenuItem* or *AddUserMenuItem2* method do not have valid handles, which means they cannot be removed using this method. However, if other items are removed and a separator bar becomes the last entry in a menu, it is removed automatically.

Application.getDefaultPath Method

This method returns the default path HotDocs uses for the specified file type. You can use this method to determine the default path for any type of file in the *HDDirectory* enumeration.

This method was introduced with the release of HotDocs 2005 SP2.

Syntax

```
string GetDefaultPath ( HotDocs.HDDirectory directory, bool bRecent )
```

Parameters	Description
directory	This is a value corresponding to the default folder for the desired file type.
bRecent	This indicates whether you want to use the default path (false) or the most recently used path (true) for the type of file indicated in the first parameter.

Return Value

The default path HotDocs uses for the specified file type.

Example

The following Visual C# example loops through each item in the HDDirectory enumeration and displays a message box with the default and most recently used paths for that file type:

```
public class ExampleCode
{
    static void Main()
    {
        HotDocs.Application app = new HotDocs.Application();
        string msg;

        foreach (HotDocs.HDDirectory item in
Enum.GetValues(typeof(HotDocs.HDDirectory)))
        {
            msg = item.ToString() + ":\r\n";
            msg += app.GetDefaultPath(item, false) + "\r\n";
            msg += app.GetDefaultPath(item, true) + "\r\n";
            MessageBox.Show(msg);
        }
    }
}
```

Application.GetHotDocsSetting Method

This method returns a HotDocs setting from the system registry. HotDocs looks first in the HKEY_CURRENT_USER registry hive, followed by the HKEY_LOCAL_MACHINE hive. If the setting is not found in either hive, the defaultValue is returned.

This method was introduced with the release of HotDocs 2005 SP2.

Syntax

```
object GetHotDocsSetting ( string sectionName, string valueName, object defaultValue
)
```

Parameters	Description
sectionName	This is the section (or subkey) of the HotDocs registry key to search. For example, Locations would search the HKCU\HotDocs\HotDocs\Locations key.
valueName	This is the name of the value to search. For example, Program Files is a string value in the Locations key that refers to the main HotDocs program files folder.
defaultValue	This is the default value to return if HotDocs cannot find a value in the registry.

Return Value

This is the value returned by the search.

Example (Visual C#)

The following Visual C# example displays a message box with the HotDocs program files location:

```
public class ExampleCode
{
    static void Main()
    {
        HotDocs.Application app = new HotDocs.Application();

        object oValue;
        oValue = app.GetHotDocsSetting("Locations", "Program Files", "Error!
Program Files value not found.");
        MessageBox.Show(oValue.ToString());
    }
}
```

Application.OpenLibrary Method

This method opens a HotDocs library (.HDL) file. If another library is already open, *OpenLibrary* closes the other library and opens the library specified in the *libPath* parameter.

Syntax

```
void OpenLibrary ( string libPath, bool addToMRU )
```

Parameters	Description
libPath	The file system path to the library.

addToMRU	[optional] If this parameter has a value of **True**, the opened library is added to the list or most recently used libraries. (The default value is **True**.)

Example

The following Visual C# example opens a HotDocs Library file without adding it to the most recently used library list:

```
public class ExampleCode
{
    static void Main()
    {
        HotDocs.Application app = new HotDocs.Application();
        app.Visible = true;

        app.OpenLibrary(@"C:\Documents\HotDocs\Libraries\Probate.hdl",
false);
    }
}
```

Application.PrintDocument Method

This method prints the specified document.

For most file types, this method uses the *ShellExecute* command to print the specified document. Printing is only successful with documents whose host application is registered to provide print functionality; word processors are typically registered to handle printing their own document types.

Syntax

```
void PrintDocument ( string docPath )
```

Parameters	Description
docPath	The file system path to the document to print.

Example

The following Visual C# example prints a document located in the Documents folder:

```
public class ExampleCode
{
    static void Main()
    {
```

```
        HotDocs.Application app = new HotDocs.Application();

        app.PrintDocument(@"C:\Documents\HotDocs\Templates\Test.rtf");
    }
}
```

Application.PublishOnlineFiles Method

This method publishes a template for use with HotDocs Server. Like the Publishing Wizard (available in the HotDocs Tools menu), this method scans the template for any inserted templates and builds the JavaScript (.JS) and HotDocs Variable Collection (.HVC) files required for HotDocs Server. These .JS and .HVC files are then copied to an output folder along with the template files.

This method was introduced in HotDocs 6.1 SP1. Also, it can only be used with HotDocs Developer. If used with HotDocs Developer LE, User, or Player, it returns an error.

Syntax

```
void PublishOnlineFiles ( string TemplatePath, string destinationDir )
```

Parameters	Description
templatePath	The file system path of the template to publish.
destinationDir	The output folder to which resulting files are copied.

Example

The following Visual C# example creates the set of files required to assemble a template using HotDocs Server and copies them to the specified folder.

```
public class ExampleCode
{
    static void Main()
    {
        HotDocs.Application app = new HotDocs.Application();

app.PublishOnlineFiles(@"C:\Documents\HotDocs\Templates\demoempl.rtf",
@"C:\HotDocs Server");
    }
}
```

Application.PublishOnlineFiles2 Method

For HotDocs Desktop 11, this method is deprecated and throws an exception. For HotDocs Desktop 11 use *CreateTemplatePackage* instead.

Use this method to publish a template for use with HotDocs Server. Like the Publishing Wizard (available in the HotDocs Tools menu), this method scans the template for any inserted templates and builds the JavaScript (.JS), HotDocs Variable Collection (.HVC) and Silverlight (.dll) files required for HotDocs Server. This method adds these files and a copy of the template files to the output folder.

Syntax

```
void PublishOnlineFiles2(string templatePath, string destinationDir, HDServerFileType
fileTypes)
```

Parameters	Description
templatePath	The file system path of the template to publish.
destinationDir	The output folder to which resulting files are copied.
fileTypes	The types of support files to generate.

Example

The following C# example will publish suitable files for serving both JavaScript and Silverlight browser interviews:

```
public class ExampleCode
{
    static void Main()
    {
        HotDocs._Application7 app = new HotDocs.Application();

        HotDocs.HDServerFileType fileTypes =
HotDocs.HDServerFileType.HDServerFilesJavaScript |
HotDocs.HDServerFileType.HDServerFilesSilverlight;
        string templatePath = @"C:\temp\Demo Editor List.rtf";
        string destinationDir = @"C:\temp\publish";

        app.PublishOnlineFiles2(templatePath, destinationDir, fileTypes);
    }
}
```

Application.ResolveReferencePath Method

This method converts a reference path to a full file system path. For example, if you use *SelectTemplate2* to get the path of a selected template that includes a reference path (e.g., ^PUBTest\template.rtf), this method can look up the reference path keyword and return a full path (e.g., C:\HotDocs\Templates\template.rtf).

This method was introduced with the release of HotDocs 2005 SP2.

Syntax

```
string ResolveReferencePath ( string referencePath )
```

Parameters	Description
referencePath	A file path containing a reference path keyword (e.g., ^PUBTest\template.rtf).

Return Value

A full file path for the referenced template.

Example

The following Visual C# example allows the user to select a template and then it displays the unresolved and resolved reference paths.

```
public class ExampleCode
{
    static void Main()
    {
        HotDocs.Application app = new HotDocs.Application();
        string tplPath, tplTitle, tplDesc;

        app.SelectTemplate2(@"C:\Documents\HotDocs\Libraries\MyLibrary.hdl",
true, false, out tplPath, out tplTitle, out tplDesc);

        MessageBox.Show(tplPath); //Unresolved path
        tplPath = app.ResolveReferencePath(tplPath);
        MessageBox.Show(tplPath); //Resolved path
    }
}
```

Application.RetrieveUrlFile Method

This method retrieves a file from the specified URL.

This method was introduced with the release of HotDocs 2005.

Syntax

```
void RetrieveUrlFile ( string url, ref string FileName )
```

Parameters	Description
url	The URL of the file to retrieve.
FileName	The file name of the file to retrieve. If the file is not already in the Internet Explorer cache, HotDocs will save the downloaded file in the location specified in this parameter. If the file is in the cache, however, the value returned in this parameter will be the location of the file in the cache.

Example

The following Visual C# example downloads a file from a Web site and saves it on the local hard drive:

```
public class ExampleCode
{
    static void Main()
    {
        HotDocs.Application app = new HotDocs.Application();

        string url = "http://www.hotdocs.com/sites/default/files/logo_4.png";
        string FileNameDestination = @"C:\temp\image.jpg";
        string FileName = FileNameDestination;

        app.RetrieveUrlFile(url, ref FileName);

        // If the file was already in the Internet Explorer cache, FileName
will not be the same value as it was originally.
        // In that case, copy the file from the cache to the desired
location.
        if (FileName != FileNameDestination)
            System.IO.File.Copy(FileName, FileNameDestination, true);
    }
}
```

Application.SaveDocAsPDF Method

This method converts a document file to a PDF file using HotDocs PDF Advantage. The conversion is done automatically, so no user intervention is necessary, although some user interface may be displayed.

This method was introduced with the release of HotDocs 6.1 SP1.

If HotDocs PDF Advantage is not installed, this method will return an error.

Syntax

```
void SaveDocAsPDF ( string docFileName, string destinationFileName )
```

Parameters	Description
docFileName	The file system path for the document file to convert. This file must be one of the supported HotDocs document types (RTF, DOCX, WPD, HPD, HFD, PDF, EVY).
destinationFileName	The file system path for the PDF file that will be created.

Example

The following Visual C# example converts an RTF document to a PDF file:

```
public class ExampleCode
{
    static void Main()
    {
        HotDocs.Application app = new HotDocs.Application();

        app.SaveDocAsPDF(@"C:\Documents\HotDocs\Templates\Testdoc.rtf",
@"C:\Documents\HotDocs\Templates\Testdoc.pdf");
    }
}
```

Application.SelectMultipleTemplates Method

This method opens the specified library in a modal dialog box, allowing users to select multiple templates by pressing **Shift** or **Ctrl** as they click templates. The selected templates are then passed back to the integrating program in a *SAFEARRAY* structure.

Syntax

```
void SelectMultipleTemplates ( string libPath, bool bOpen, out int tplCount, out
object tplPaths, out object tplTitles, out object tplDescs )
```

Parameters	Description
libPath	The file system path to the library the integration must open. This path must point to an .HDL file. If this path is an empty string (""), then HotDocs will attempt to display the last library the user viewed.
bOpen	If bOpen == true, then the modal dialog will include an Open button, allowing the user to open a different library file. If bOpen == false, then the modal dialog will not include an Open button and the user will not be able to open a different library file.
tplCount	The number of templates the user selected.
tplPaths	It contains an array of (count) that hold the file system paths (including the file name) of the selected templates.
tplTitles	[optional in languages supported] It contains an array of (count) that hold the titles of the selected templates. If there is no title for a selected template, that portion of the array will hold an empty string ("").
tplDescs	[optional in languages supported] Contains an array of (count) that hold the descriptions of the selected templates. If there is no description for a selected template, that portion of the array will hold an empty string (""). As a note, template descriptions can be quite long, so if you use this parameter, this method could use a lot of memory.

Example (Visual C#)

The following Visual C# example displays a dialog box where users can select multiple templates from the initial library or open a different library. It then displays information about the templates selected:

```
public class ExampleCode
{
    static void Main()
    {
        HotDocs.Application app = new HotDocs.Application();

        string libPath = @"C:\Documents\HotDocs\Libraries\DOCX Tutorial
Templates.hdl";
        int tplCount;
        dynamic tplPaths;
        dynamic tplTitles;
        dynamic tplDescs;

        app.SelectMultipleTemplates(libPath, true, out tplCount, out
tplPaths, out tplTitles, out tplDescs);

        MessageBox.Show("You selected " + tplCount + " templates.");

        if (tplCount > 0)
        {
            for (int i = 0; i < tplCount; i++)
            {
```

```
                MessageBox.Show(tplPaths[i] + "\r\n" + tplTitles[i] + "\r\n"
+ tplDescs[i]);
            }
        }

        System.Runtime.InteropServices.Marshal.ReleaseComObject(app);
    }
}
```

Application.SelectMultipleTemplates2 Method

This method opens the specified library in a modal dialog box, allowing users to select multiple templates by pressing **Shift** or **Ctrl** as they click templates. The selected templates are then passed back to the integrating program in a *SAFEARRAY* structure.

The difference between this method and *SelectMultipleTemplates* is the *bResolveReferencePaths* parameter. When this parameter is FALSE, template paths containing reference keywords are returned as reference paths (e.g., ^PUBpath\templatename.rtf) instead of being resolved to full paths (e.g., C:\Templates\templatename.rtf). You can then use the *ResolveReferencePath* method to translate the reference path into a full path as needed.

This method was introduced with the release of HotDocs 2005 SP2.

Syntax

```
void SelectMultipleTemplates2 ( string libPath, bool bOpen, bool
bResolveReferencePaths, out int tplCount, out object tplPaths, out object tplTitles,
out object tplDescs )
```

Parameters	Description
libPath	The file system path to the library the integration must open. This path must point to an .HDL file. If this path is an empty string (""), then HotDocs will attempt to display the last library the user viewed.
bOpen	If bOpen == true, then the modal dialog will include an Open button, allowing the user to open a different library file. If bOpen == false, then the modal dialog will not include an Open button and the user will not be able to open a different library file.
bResolveReferencePaths	Indicates if reference paths will be resolved into full file paths (TRUE) or left as reference paths (FALSE).
tplCount	The number of templates the user selected.
tplPaths	It contains an array of (count) that hold the file system paths (including the file name) of the selected templates.

tplTitles	[optional] It contains an array of (count) that hold the titles of the selected templates. If there is no title for a selected template, that portion of the array will hold an empty string ("").
tplDescs	[optional] Contains an array of (count) that hold the descriptions of the selected templates. If there is no description for a selected template, that portion of the array will hold an empty string (""). As a note, template descriptions can be quite long, so if you use this parameter, this method could use a lot of memory.

Example (Visual C#)

The following Visual C# example displays a dialog box where users can select multiple templates from the initial library or open a different library. It then displays information about the templates selected:

```
public class ExampleCode
{
    static void Main()
    {
        HotDocs.Application app = new HotDocs.Application();

        string libPath = @"C:\Documents\HotDocs\Libraries\DOCX Tutorial
Templates.hdl";
        int tplCount;
        dynamic tplPaths;
        dynamic tplTitles;
        dynamic tplDescs;

        app.SelectMultipleTemplates2(libPath, true, false, out tplCount, out
tplPaths, out tplTitles, out tplDescs);

        MessageBox.Show("You selected " + tplCount + " templates.");

        if (tplCount > 0)
        {
            for (int i = 0; i < tplCount; i++)
            {
                MessageBox.Show(tplPaths[i] + "\r\n" + tplTitles[i] + "\r\n"
+ tplDescs[i]);
            }
        }

        System.Runtime.InteropServices.Marshal.ReleaseComObject(app);
    }
}
```

Application.SelectTemplate Method

This method opens a modal dialog that displays the specified library, allowing the user to select a template. The path, title, and description of the selected template are then passed back to the integrating program.

Syntax

```
void SelectTemplate ( string libPath, bool bOpen, out string tplPath, out string
tplTitle, out string tplDesc )
```

Parameters	Description
libPath	The file system path to the library the integration must open. This path must point to an .HDL file. If this path is an empty string (""), then HotDocs will attempt to display the last library the user viewed.
bOpen	If bOpen == true, then the modal dialog will include an 'Open' button, allowing the user to open a different library. If bOpen == false, then the modal dialog will not contain an Open button, and the user cannot open a different library.
tplPath	Contains the file system path (including file name) of the selected template. If no template was chosen, this is an empty string ("").
tplTitle	[optional] Contains the template title of the selected template.
tplDesc	[optional] Contains the template description of the selected template. As a note, template descriptions can be quite long, so if you use this parameter, this method could use a lot of memory.

Example

The following Visual C# example displays a dialog box where users can select a template. It then displays information about the template selected:

```
public class ExampleCode
{
    static void Main()
    {
        string sTitle, sDescription, sPath;
        string sLibrary = @"C:\Documents\HotDocs\Libraries\DOCX Tutorial
Templates.hdl";
        HotDocs.Application app = new HotDocs.Application();

        app.SelectTemplate(sLibrary, true, out sPath, out sTitle, out
sDescription);


Console.WriteLine("Title:\t{0}\nDescription:\t{1}\nPath:\t{2}",sTitle,sDescri
ption,sPath);
    }
}
```

Application.SelectTemplate2 Method

This method opens the specified library in a modal dialog box, allowing users to select a single template. The path, title, and description of the selected template are then passed back to the integrating program.

The difference between this method and *SelectTemplate* is the bResolveReferencePaths parameter. When this parameter is FALSE, a template path containing a reference keyword is returned as a reference path (e.g., ^PUBpath\templatename.rtf) instead of being resolved to a full path (e.g., C:\Templates\templatename.rtf). You can then use the *ResolveReferencePath* method to translate the reference path into a full path as needed.

This method was introduced with the release of HotDocs 2005 SP2.

Syntax

```
void SelectTemplate2 ( string libPath, bool bOpen, bool bResolveReferencePaths, out
string tplPath, out string tplTitle, out string tplDesc )
```

Parameters	Description
libPath	The file system path to the library the integration must open. This path must point to an .HDL file. If this path is an empty string (""), then HotDocs will attempt to display the last library the user viewed.
bOpen	If bOpen == true, then the modal dialog will include an 'Open' button, allowing the user to open a different library. If bOpen == false, then the modal dialog will not contain an Open button, and the user cannot open a different library.
bResolveReferencePaths	Indicates if a reference path will be resolved into a full file path (TRUE) or left as a reference path (FALSE).
tplPath	Contains the file system path (including file name) of the selected template. If no template was chosen, this is an empty string ("").
tplTitle	[optional] Contains the template title of the selected template.
tplDesc	[optional] Contains the template description of the selected template. As a note, template descriptions can be quite long, so if you use this parameter, this method could use a lot of memory.

Example

The following Visual C# example displays a dialog box where users can select a template. It then displays information about the template selected which resolves to a full path.

```
public class ExampleCode
{
    static void Main()
    {
        string sTitle, sDescription, sPath;
        string sLibrary = @"C:\Documents\HotDocs\Libraries\DOCX Tutorial
Templates.hdl";
        HotDocs.Application app = new HotDocs.Application();

        app.SelectTemplate2(sLibrary, true, true, out sPath, out sTitle, out
sDescription);


Console.WriteLine("Title:\t{0}\nDescription:\t{1}\nPath:\t{2}",sTitle,sDescri
ption,sPath);
    }
}
```

Application.SendToWordProcessor Method

This method sends a document specified in the *docFileName* parameter to the word processor.

Syntax

```
void SendToWordProcessor ( string docFileName )
```

Parameters	Description
docFileName	The file system path to the document to open. The file format must be one that is supported by HotDocs, for example: Word (.DOCX and .RTF), WordPerfect (.WPD), or HotDocs Filler (.HFD, .HPD).

Example

The following Visual C# example sends a document located in the Documents folder to the word processor:

```
public class ExampleCode
{
    static void Main()
    {
        string sDocument = @"C:\Documents\HotDocs\Templates\Test.docx";
        HotDocs.Application app = new HotDocs.Application();

        app.SendToWordProcessor(sDocument);
    }
```

```
}
```

Application.SetUserInterfaceItem Method

This method sets the state for various features (elements) of the HotDocs library window. For example, you can use this method to disable features your integration users should not have access to, or you can enable features users may have disabled.

Syntax

```
void SetUserInterfaceItem ( HotDocs.HDLUI element, bool enabled )
```

Parameters	Description
element	The user interface element to change. This must be an HDLUI enum member.
enabled	Turns the feature on (TRUE) or off (FALSE).

Example

The following Visual C# example disables three features in the interface to prevent the user from changing the currently open library:

```
public class ExampleCode
{
    static void Main()
    {
        HotDocs.Application app = new HotDocs.Application();

        app.SetUserInterfaceItem(HotDocs.HDLUI.LUIFILENEWLIBRARY, false);
        app.SetUserInterfaceItem(HotDocs.HDLUI.LUIFILEOPENLIBRARY, false);
        app.SetUserInterfaceItem(HotDocs.HDLUI.LUIFILEMRULIST, false);
        app.Visible = true;
    }
}
```

Application.ActiveAssembly Property

[Read-only] This property returns an *Assembly* object representing the template currently being assembled.

You should make sure there is an active assembly before using this property to find out information about the assembly. In addition, you should use caution when manipulating the *Assembly* object. The state of the object should not be changed during assembly.

Syntax

```
HotDocs.Assembly ActiveAssembly [ get ]
```

Example

The following Visual C# example displays a message box with the name of the template currently being assembled or a message indicating that there is no active assembly:

```
public class ExampleCode
{
    static void Main()
    {
        HotDocs.Application app = new HotDocs.Application();
        HotDocs.AssemblyCollectionClass asc;
        bool bActive;
        asc = app.Assemblies;

        // Find out if there is an active assembly
        bActive = false;
        if (asc.Count > 0)
            for (int i = 0; i < asc.Count; i++)
                if
(asc.Item(i).Status.Equals(HotDocs.HDASSEMBLYSTATUS.HDASMSTATUSASSEMBLING))
                    bActive = true;

        // Show message box
        if (bActive)
            MessageBox.Show (app.ActiveAssembly.TemplateTitle);
        else
            MessageBox.Show("There is no active assembly.");
    }
}
```

Application.Assemblies Property

[Read-only] This property returns an *AssemblyCollection* object, which is the collection of *Assembly* objects in the HotDocs assembly queue. By querying the *Assemblies* property, you can get all of the *Assembly* objects that are queued for assembly.

Syntax

```
HotDocs.AssemblyCollectionClass Assemblies [ get ]
```

Example

The following Visual C# example displays the number of items in the assembly queue. It also lists the title of each assembly (if the queue is not empty):

```
public class ExampleCode
{
    static void Main()
    {
        HotDocs.Application app = new HotDocs.Application();

        Console.WriteLine("The assembly queue contains {0}
template(s).",app.Assemblies.Count);
        for (int i = 0; i < app.Assemblies.Count; i++)
            Console.WriteLine("\t" + app.Assemblies.Item(i).TemplateTitle);
    }
}
```

Application.AssemblyQueueVisible Property

[Read/Write] This Boolean property controls the visibility status of the HotDocs assembly queue. For example, if the *AssemblyQueueVisible* property is False, the assembly queue is not visible.

Syntax

```
bool AssemblyQueueVisible [ set, get ]
```

Example

The following Visual C# example makes the assembly queue invisible if it is currently visible. Otherwise, it makes the assembly queue visible if it is currently invisible:

```
public class ExampleCode
{
    static void Main()
    {
        HotDocs.Application app = new HotDocs.Application();
        app.AssemblyQueueVisible = !(app.AssemblyQueueVisible);
    }
}
```

Application.CanAssembleAll Property

[Read-only] This property indicates whether or not the version of HotDocs in use is capable of assembling unregistered templates. Specifically, this property returns **true** if the version of HotDocs is anything other than Player.

This property was introduced with the release of HotDocs 10.

Syntax

```
bool CanAssembleAll [ get ]
```

Application.CanEditTemplates Property

[Read-only] This property indicates whether or not the version of HotDocs in use is capable of editing templates. Specifically, this property returns **true** if the version of HotDocs is Developer or Developer LE.

This property was introduced with the release of HotDocs 10.

Syntax

```
bool CanEditTemplates [ get ]
```

Application.CommandLine Property

[Write-only] This property sets the HotDocs command line options as if it were started with a particular command line. Setting this property to a string is the same as if the string were passed to the executable when the program was started. For example, if the command line invokes an assembly, a new *Assembly* object is added to the queue. If the command line changes the appearance or behavior of HotDocs, the change happens immediately.

Syntax

```
string CommandLine [ set ]
```

Example

The following Visual C# example sets the CommandLine property to cause a new assembly to begin using the specified template file:

```
public class ExampleCode
{
    static void Main()
    {
        HotDocs.Application app = new HotDocs.Application();
        app.CommandLine =
@"/tf=C:\Documents\HotDocs\Templates\demoempl.docx";
    }
}
```

Application.CurrentLibraryPath Property

[Read-only] This property returns the file system path and file name of the current (open) HotDocs library as a String value.

Syntax

```
string CurrentLibraryPath [ get ]
```

Example

The following Visual C# example displays a message box containing the file name and path of the current HotDocs library.

```
public class ExampleCode
{
    static void Main()
    {
        HotDocs.Application app = new HotDocs.Application();
        app.Visible = true;
        MessageBox.Show("Current Library File and Path: " +
app.CurrentLibraryPath);
    }
}
```

Application.Flavor Property

[Read-only] This property returns a value corresponding to which HotDocs edition (Player, User, Developer, or Developer LE) is being used.

This property returns one of the following values from the *HDPRODUCTFLAVOR* enumeration:

Name	Value	Description
PLAYER	1	HotDocs Player
STANDARD	2	HotDocs Developer LE
PROFESSIONAL	3	HotDocs Developer
USER	4	HotDocs User

Syntax

```
HotDocs.HDPRODUCTFLAVOR Flavor [ get ]
```

Example

The following Visual C# example displays the HotDocs flavor in a message box:

```
program class ExampleCode
{
    static void Main()
    {
        HotDocs.Application app = new HotDocs.Application();

        switch (app.Flavor)
        {
            case HotDocs.HDPRODUCTFLAVOR.PLAYER:
                MessageBox.Show("HotDocs Player");
                break;
            case HotDocs.HDPRODUCTFLAVOR.STANDARD:
                MessageBox.Show("HotDocs Developer LE");
                break;
            case HotDocs.HDPRODUCTFLAVOR.PROFESSIONAL:
                MessageBox.Show("HotDocs Developer");
                break;
            case HotDocs.HDPRODUCTFLAVOR.USER:
                MessageBox.Show("HotDocs User");
                break;
        }
    }
}
```

Application.Hwnd Property

[Read-only] This property returns the window handle of the HotDocs library window.

Syntax

```
int Hwnd [ get ]
```

Example

The following Visual C# example displays the HotDocs library window handle:

```
public class ExampleCode
{
    static void Main()
    {
        HotDocs.Application app = new HotDocs.Application();
        App.Visible = true;
        MessageBox.Show(app.Hwnd.ToString());
    }
}
```

Application.Plugins Property

[Read-only] This property returns a *PluginsClass* object, which represents a collection of plug-ins currently registered with HotDocs.

This property was introduced with the release of HotDocs 2005 SP2.

Syntax

```
HotDocs.PluginsClass Plugins [ get ]
```

Application.Version Property

[Read-only] This property returns the HotDocs product version number as a String value. For example, if you have HotDocs 11 installed, this property returns **11**.

Syntax

```
string Version [ get ]
```

Example

The following Visual C# example displays the HotDocs version number in a message box:

```
public class ExampleCode
{
    static void Main()
    {
        HotDocs.Application app = new HotDocs.Application();
        MessageBox.Show(app.Version);
    }
}
```

Application.Visible Property

[Read/Write] This Boolean property controls the visibility status of the HotDocs library window. For example, if the *Visible* property is False, the library window is not visible.

Syntax

```
bool Visible [ get, set ]
```

Example

The following Visual C# example makes the library window invisible if it is currently visible. Otherwise, it makes the library window visible if it is currently invisible.

```
public class ExampleCode
{
    static void Main()
    {
        HotDocs.Application app = new HotDocs.Application();

        if (app.Visible)
            app.Visible = false;
        else
            app.Visible = true;
    }
}
```

Application.AssemblyCompleteEvent Event

This event is fired when assembly completes.

AssemblyCompleteEvent is a replacement event method for *OnAssemblyCompleteEvent*. It is intended to eliminate the need to pass the *AnswerCollection* by pointer and instead wrap it in a object. This should help Visual Basic programmers deal with the VARIANT event. This event should be fired at the same time as *OnAssemblyCompleteEvent*.

Syntax

```
void AssemblyCompleteEvent(string tplTitle, string tplPath, string docPath, object
ansColl, int assemblyHandle)
```

Application.OnAssemblyCompleteEvent Event

This event has been deprecated for HotDocs Desktop 11. *AssemblyCompleteEvent* is recommended for use instead.

This event is fired when an assembly is completed. It returns the name and path of the template that was used to assemble the document, the path to the assembled document, a pointer to the *AnswerCollection* object used in the assembly, and the assembly handle which was given when the assembly was added to the queue.

Syntax

```
void OnAssemblyCompleteEvent ( string tplTitle, string tplPath, string docPath,
HotDocs._AnswerCollection ansColl, int AssemblyHandle )
```

Parameters	Description
tplTitle	The title of the template used during the assembly.
tplPath	The file system path (including file name) of the template.
docPath	The file system path to the assembled document. If the user chooses not to save the document, this will be an empty string ("").
ansColl	A pointer to the *AnswerCollection* object used during the assembly.
assemblyHandle	The assemblyhandle returned when the *Assembly* object was added to the queue.

Application.OnAssemblyStartEvent Event

This event is fired when an assembly starts. It returns a reference to the *Assembly* object that represents the assembly session.

Syntax

```
void OnAssemblyStartEvent ( HotDocs.Assembly assemblyObject )
```

Parameters	Description
assemblyObject	The *Assembly* object that represents the assembly session.

Application.OnErrorEvent Event

This event is fired when an error occurs. By returning true for the override parameter, the integration can tell HotDocs not to display any user interface indicating that an error occurred, which allows the integration to either display its own error message or silently handle the error.

Syntax

```
void OnErrorEvent ( int errCode, string errMesg, out bool override )
```

Parameters	Description
errcode	Numerical error code.
errmesg	Text description of error.
override	Return parameter: • *override == true means HotDocs will not display any user interface indicating that an error occurred. • *override == false means that HotDocs will display the normal error notification.

Application.OnLibraryInterfaceCloseEvent Event

This event is fired when the user closes the HotDocs library user interface.

Syntax

```
void OnLibraryInterfaceCloseEvent ( )
```

Application.OnLibraryOpenEvent Event

This event is fired when a HotDocs library is opened.

Syntax

```
void OnLibraryOpenEvent ( )
```

Application.OnTemplateSelectedEvent Event

This event is fired when the user selects a template in the library to assemble, or selects a template at the *SelectTemplate* or *SelectMultipleTemplate* dialogs. By returning *override == true, the integration can cancel the selection of the template.

Syntax

```
void OnTemplateSelectedEvent ( string tplTitle, string tplPath, out bool override )
```

Parameters	Description
tpltitle	The title of the template. If the template does not have a title, this will be an empty string ("").
tplpath	The file system path (including the file name) of the selected template.
override	Return parameter: • *override == true will cancel the selection of the template. • *override == false will allow the selection to continue normally.

Application.OnUserInterfaceEvent Event

This event is fired when the user selects items in the library user interface.

Syntax

```
void OnUserInterfaceEvent ( HotDocs.HDLUI hdEvent, out bool override )
```

Parameters	Description
hdevent	The user interface item selected. It is one of the values from the HDLUI enumeration.
override	Return parameter: • *override == true will cancel the user interface action. • *override == false will allow the action to continue normally.

Application.OnUserMenuItemClickedEvent Event

This event is fired when the user selects an integration-defined menu item.

Syntax

```
void OnUserMenuItemClickedEvent ( int menuHandle )
```

Parameters	Description
menuHandle	The handle for the user-defined menu. This is the handle returned from the *Application.AddUserMenuItem* method.

HotDocs.Assembly Object

HotDocs.Assembly Object

The *Assembly* object represents a HotDocs assembly. All HotDocs assemblies are represented by an *Assembly* object internally (with the exception of test assemblies). The *Assembly* object contains information about the inputs and outputs of the assembly, state information about how the assembly is performed, and features to customize and control the *Assembly* interface.

All assemblies are stored in a queue called the *AssemblyCollection*. The *AssemblyCollection*is exposed through the API with the *HotDocs.Application.Assemblies* property. The queue is ordered so the items at the top of the collection will be processed before the assemblies at the bottom of the queue (such is the nature of queues).

When HotDocs begins an assembly, it doesn't remove the *Assembly* object from the queue. The object is still accessible from the *HotDocs.Application.Assemblies* property or through the *HotDocs.Application.ActiveAssembly* property. Be careful about accessing the currently assembling object however. By changing the state of the currently assembling *Assembly* object, you can put HotDocs into a bad state.

General Information

ProgID:	HotDocs.Assembly.11.0 HotDocs.Assembly (version-independent)
CLSID:	{503AB2B4-5D01-4EF0-9B2B-36E1B9C738A9}

The following table shows the name and IID for each interface, as well as the version of HotDocs in which it was introduced. The primary interface and the main public interface exposed by this object is _Assembly.

Name	IID	Added in
_Assembly	{A99AB319-0378-4033-9534-DF296B6B63C6}	Added in HotDocs 6.0
_Assembly2	{A99AB320-0378-4033-9534-DF296B6B63C6}	Added in HotDocs 6.1 SP1
_Assembly3	{A99AB321-0378-4033-9534-DF296B6B63C6}	Added in HotDocs 2005 SP2
_AssemblyEvents	{58172076-C690-434D-942B-F3EA55693C98}	Added in HotDocs 6.0

The _AssemblyEvents interface designates an event sink interface that an application must implement in order to receive event notifications from a *HotDocs.Assembly* object.

Methods

Method	Description
AddUserMenuItem	This method adds a custom menu item to the HotDocs assembly window menus.
DeleteUserMenuItem	This method removes a custom menu item from the HotDocs assembly window menu that was created using the *AddUserMenuItem* method.

GetSaveAsExtDlg	This method prompts the user for a file type to use when saving a document in a document manager.
LocalBrowseDlg	This method displays a dialog box for the user to select a file.
OpenAnswerFileDlg	This method displays the **Open Answer File** dialog box.
SelectOpenAnswerFileDlg	This method displays the **Open Answer File** dialog box and returns the file name and path of the selected answer file.
SendToWordProcessor	This method sends the resulting assembled document to the word processor.
SetUserInterfaceItem	This method sets the state for the element user interface element. This method is useful to turn off features the users of your integration should not have access to and enable features users may have disabled.
UseAnswerFile	This method creates an *AnswerCollection* object, loads the answers from the *answerFilePath* answer file into it, and sets the *Assembly.AnswerCollection* property to the object. This is just a shortcut method, as everything it does you could do manually by using other methods and properties.

Properties

Property	Description
AnswerCollection	[Read/Write] This property controls the *AnswerCollection* object used during assembly. The value of this property is a reference to the *AnswerCollection* object used by the *Assembly* object.
AnswerSummaryPath	[Read/Write] This property controls the location to which the answer summary file is written. If this property is set when the assembly starts, an Answer Summary will be written in the specified location when the assembly is complete.
Application	[Read-only] This property returns a reference to the *Application* object.
AssemblyHandle	[Read-only] This property reports the handle that can identify the assembly. This value is the same as the assembly handle returned when the *Assembly* was added to the assembly queue.
CommandLine	[Write-only] This property sets command line options for the assembly as if it were started with a particular command line from the library. Setting this property to a string is the same as if the string were passed in from the library entry.
DocumentPath	[Read/Write] This property controls the location where the assembled document file is written. It is the file path for the destination file.
Hwnd	[Read-only] This property returns the window handle for the HotDocs

	Assembly interface.
KeepInQueue	[Read/Write] This Boolean property controls the behavior of the assembly queue. If it is TRUE when the assembly is complete, a reference will remain in the Assembly Queue window. If it is FALSE, all references to the assembly will be destroyed when the assembly is complete.
Map	[Read/Write] This property controls which *VarMap* object is used during assembly.
PrintWhenComplete	[Read/Write] This Boolean property controls whether the document is automatically printed when the assembly completes. It controls the same behavior as specifying a Print (/pr) option at the command line.
PromptToSaveDocument	[Read/Write] This is a Boolean property that controls whether HotDocs will prompt to save a copy of the assembled document at the end of the assembly. The initial value is the value stored in the Document Assembly folder in HotDocs Options (**Tools** > **Options** > **Document Assembly** > **Prompt to save document when closing assembly window**). If the integrator changes the *PromptToSaveDocument* property, HotDocs will disregard the setting in **HotDocs Options** in favor of the value set in the integration.
QuestionSummaryPath	[Read/Write] This property controls the location to which the question summary file is written. If this property is set when the assembly starts, a question summary will be written out as an HTML document when the assembly is complete.
ShowAnswerFileDialog	[Read/Write] This Boolean property controls whether or not the **Save Answers** dialog box appears at the end of an assembly.
Status	[Read-only] This property tells what state the *Assembly* object is in.
SuppressUnansweredWarning	[Read/Write] This is a Boolean property that sets the **Display unanswered warning before saving documents** option for the given assembly. If this property is **false**, then HotDocs will display a warning before the user saves the document or sends the document to the word processor if there are variables which are used but not answered. If this property is **true**, then all such warnings are suppressed.
TemplateDesc	[Read/Write] This property contains the description for the template that will be used to assemble a document. This property is for the integration programmer's convenience. It is set but is not used by HotDocs internally.
TemplatePath	[Read/Write] This property contains the file system path for the template that will be used to assemble a document.
TemplateTitle	[Read/Write] This property contains the title for the template that will be used to assemble a document.

Visible	[Read/Write] This Boolean property determines if the assembly interface will be visible to the user. This property must be set before assembly starts or it will have no effect. If *Visible* returns **false**, then it has the same effect as specifying the No Assembly Window (/nw) option on the command line.

Events

Event	Description
OnAssemblyCompleteEvent	This event is fired when the assembly completes.
OnAssemblyStartEvent	This event is fired when the assembly starts.
OnCanOpenFile	This event is fired when a file can be opened.
OnCloseAssemblyInterfaceEvent	This event is fired when assembly interface actually closes.
OnErrorEvent	This event is fired when an error occurs. By returning true for the override parameter, the integration can tell HotDocs not to display any error messages, allowing the integration to either display its own error message or deal with the error silently.
OnFileOpen	This event is fired when a file is opened.
OnFileSave	This event is fired when a file is saved.
OnFileSelectEvent	This event is fired when a file is selected.
OnGetAnswerFileDisplayName	This event is fired when HotDocs gets the name of an answer file to display.
OnGetMRUInfo	This event is fired when HotDocs gets information from the most recently used (MRU) list.
OnNeedAnswerEvent	This event is fired when an answer value is needed by the assembly, but not found in the *AnswerCollection* answer set. It allows the integration to provide answers as they are needed, rather than trying to provide all the answers before the assembly starts.
OnPostCloseAnswerFile	This event is fired after the answer file is closed.
OnPostSaveDocumentEvent	This event is fired after a document is saved. The document can be an assembled document, a question summary document, or an answer summary document.
OnPreCloseAnswerFile	This event is fired when HotDocs prepares to close an answer file.
OnPreSaveDocumentEvent	This event is fired prior to saving the document. By setting *showui = false, the integration can prevent the user interface relating to saving the document from showing. By setting *override = true, the integration can prevent the save from happening.

OnUserInterfaceEvent	This event is fired when the user selects certain options at the assembly interface. This can be useful if the integration wants to override a particular HotDocs feature or command.
OnUserMenuItemClickedEvent	This event is fired when the user selects an integration-defined menu item. (See *Assembly.AddUserMenuItem* method.)
PostSaveAnswersEvent	This event is fired after an answer file is saved.
PreSaveAnswersEvent	This event is fired after the user has indicated he or she wants to save the answers, but before the actual save occurs. By setting *override == true, the integration can prevent the save from happening.

Assembly.AddUserMenuItem Method

This method adds a custom menu item to the HotDocs assembly window menus.

For example, you may want to display your own "About" dialog box from the HotDocs **Help** menu. You can use the *AddUserMenuItem* method to do this. When a user selects the item, the *_AssemblyEvents.OnUserMenuItemClickedEvent* event is fired to notify your application that your menu item was chosen.

This method only manipulates the menus in the assembly interface. To add items to the library interface, see *Application.AddUserMenuItem* or What is a HotDocs plug-in?.

Syntax

```
void AddUserMenuItem ( string menuTxt, HotDocs.HDAIMENU menuItem, out int uiHandle )
```

Parameters	Description
menuTxt	The text to be inserted into the menu. (You can use a single hyphen (-) to insert a separator bar into the menu.)
menuItem	The name of the HotDocs menu to which the menu item will be added. This must be a member of the HDAIMENU enumeration.
uiHandle	This is the menu handle that the program can use to track when someone clicks the menu entry.

Example

The following Visual C# example adds a separator and item to the Help menu:

```
public class ExampleCode
{
    static void Main()
    {
        HotDocs.Application app = new HotDocs.Application();
        HotDocs.Assembly asm = new HotDocs.Assembly();
        int h_sep, h_addh;
        asm.TemplatePath = @"C:\Documents\HotDocs\Templates\demoempl.docx";

        asm.AddUserMenuItem("-", HDAIMENU.AI_HELP, out h_sep);
        asm.AddUserMenuItem("Additional Help", HDAIMENU.AI_HELP, out h_addh);

        asm.Visible = true;
        app.Assemblies.Add(asm);
    }
}
```

Assembly.DeleteUserMenuItem Method

This method removes a custom menu item from the HotDocs assembly window menu that was created using the *AddUserMenuItem* method.

You cannot delete separator bars. HotDocs manages separators internally. When you remove a menu item, HotDocs verifies that the last item in the menu is not a separator. If it is, HotDocs will automatically remove the bar from the menu.

Syntax

```
void DeleteUserMenuItem ( int uiHandle )
```

Parameters	Description
uiHandle	This is the menu handle that was returned from *AddUserMenuItem*.

Example

The following Visual C# example adds a menu item to the Help Menu, then removes it before beginning an assembly:

```
public class ExampleCode
{
    static void Main()
    {
        HotDocs.Application app = new HotDocs.Application();
        HotDocs.Assembly asm = new HotDocs.Assembly();
```

```
        int h_addh;

        asm.TemplatePath = @"C:\My
Documents\HotDocs\Templates\demoempl.docx";

        //adds a menu item to the help menu.
        asm.AddUserMenuItem("Additional Help", HDAIMENU.AI_HELP, out h_addh);

        //removes the menu item from the help menu.
        asm.DeleteUserMenuItem(h_addh);

        asm.Visible = true;
        app.Assemblies.Add(asm);

    }
}
```

Assembly.GetSaveAsExtDlg Method

This method prompts the user for a file type to use when saving a document in a document manager.

Syntax

```
string GetSaveAsExtDlg ( string docExt )
```

Parameters	Description
docExt	This is the file name extension (.hpt, .hft, .rtf, .dot, .wpt) of the template being assembled. Depending on what type of template is assembled, the options in the **Save As** dialog box will change. It could also be an answer file type (.anx or .ans).

Example

The following Visual C# example displays the **Save in Document Manager** dialog box:

```
public class ExampleCode
{
    static void Main()
    {
        HotDocs.Application app = new HotDocs.Application();
        HotDocs._Assembly3 asm = app.ActiveAssembly as HotDocs._Assembly3;

        string ext = ".rtf";
        string fileType;
        fileType = asm.GetSaveAsExtDlg(ext);
    }
```

```
}
```

Assembly.LocalBrowseDlg Method

This method displays a dialog box for the user to select a file.

Syntax

```
void LocalBrowseDlg ( int parentWnd, bool bOpenFileDialog, string defaultExt, string
filter, string initialDirectory, ref string FileName )
```

Parameters	FileName
parentWnd	A handle to the the browse dialog's parent window.
bOpenFileDialog	Indicates if the browse dialog will be an "open file" dialog (true) or "save file" dialog (false).
defaultExt	For a "save" dialog, this is the file name extension added to the end of the specified file name if the user does not specify an extension.
filter	This limits the types of files displayed in an "open file" dialog. For example, .anx would show only files with the .anx file name extension.
initialDirectory	This indicates the folder displayed when the dialog is first displayed.
FileName	This returns the file name and path of the selected file.

Assembly.OpenAnswerFileDlg Method

This method displays the **Open Answer File** dialog box.

This method works only with a visible assembly.

Syntax

```
void OpenAnswerFileDlg ( )
```

Example

The following Visual C# example displays the "Open Answer File" dialog box:

```
public class ExampleCode
{
    static void Main()
    {
        HotDocs.Application app = new HotDocs.Application();
        HotDocs._Assembly3 asm = app.ActiveAssembly as HotDocs._Assembly3;

        asm.OpenAnswerFileDlg();
    }
}
```

Assembly.SelectOpenAnswerFileDlg Method

This method displays the **Open Answer File** dialog box and returns the file name and path of the selected answer file.

This method works only with a visible assembly.

Syntax

```
void SelectOpenAnswerFileDlg ( ref string FileName )
```

Parameters	Description
FileName	The file name and path of the selected answer file.

Example

The following Visual C# example displays the "Open Answer File" dialog box and displays the file name and path of the selected answer file.

```
public class ExampleCode
{
    static void Main()
    {
        HotDocs.Application app = new HotDocs.Application();
        HotDocs._Assembly3 asm = app.ActiveAssembly as HotDocs._Assembly3;
        String filename = "";

        asm.SelectOpenAnswerFileDlg(ref fileName);
        MessageBox.Show("The selected answer file is " + filename);
    }
}
```

Assembly.SendToWordProcessor Method

This method sends the resulting assembled document to the word processor.

Syntax

```
void SendToWordProcessor ( )
```

Assembly.SetUserInterfaceItem Method

This method sets the state for the element user interface element. This method is useful to turn off features the users of your integration should not have access to and enable features users may have disabled.

Syntax

```
void SetUserInterfaceItem ( HotDocs.HDAUI element, bool enabled )
```

Parameters	Description
element	The user interface element to change. This parameter must be a member of the HDAUI enumeration.
enabled	Enables (true) or disables (false) the feature.

Example

The following Visual C# example begins an assembly after disabling three of the user interface elements:

```
public class ExampleCode
{
    static void Main()
    {
        HotDocs.Application app = new HotDocs.Application();
        HotDocs.Assembly asm = new HotDocs.Assembly();

        asm.TemplatePath = @"C:\Documents\HotDocs\Templates\demoempl.docx";
        asm.Visible = true;

        // Disable the answer file drop-down list
        asm.SetUserInterfaceItem(HotDocs.HDAUI.AUIANSWERFILEDROPDOWN, false);

        // Disable the New Answers command
```

```
        asm.SetUserInterfaceItem(HotDocs.HDAUI.AUIFILENEWANSWERS, false);

        // Disable the Open Answers command
        asm.SetUserInterfaceItem(HotDocs.HDAUI.AUIFILEOPENANSWERS, false);

        app.Assemblies.Add(asm);
    }
}
```

Assembly.UseAnswerFile Method

This method creates an *AnswerCollection* object, loads the answers from the *answerFilePath* answer file into it, and sets the *Assembly.AnswerCollection* property to the object. This is just a shortcut method, as everything it does you could do manually by using other methods and properties.

Syntax

```
void UseAnswerFile ( string answerFilePath )
```

Parameters	Description
answerFilePath	File system path to an answer file.

Example

The following Visual C# example begins an assembly using a specific answer file:

```
public class ExampleCode
{
    static void Main()
    {
        HotDocs.Application app = new HotDocs.Application();
        HotDocs.Assembly asm = new HotDocs.Assembly();

        asm.TemplatePath = @"C:\temp\Demo Employment Agreement.docx";
        asm.UseAnswerFile(@"C:\temp\EmployeeAnswers.anx");
        asm.Visible = true;
        app.Assemblies.Add(asm);
    }
}
```

Assembly.AnswerCollection Property

[Read/Write] This property controls the *AnswerCollection* object used during assembly. The value of this property is a reference to the *AnswerCollection* object used by the Assembly object.

Syntax

```
HotDocs._AnswerCollection AnswerCollection [ set, get ]
```

Assembly.AnswerSummaryPath Property

[Read/Write] This property controls the location to which the answer summary file is written. If this property is set when the assembly starts, an Answer Summary will be written in the specified location when the assembly is complete.

Answer summaries are HTML documents.

Syntax

```
string AnswerSummaryPath [ set, get ]
```

Assembly.Application Property

[Read-only] This property returns a reference to the *Application* object.

Since there is only one *Application* object on a machine at a time, this property will return a reference to the same object as the Application property on any other object in HotDocs, and a reference to the same object as if you created a new *HotDocs.Application* object.

Syntax

```
HotDocs._Application2 Application [ get ]
```

Assembly.AssemblyHandle Property

[Read-only] This property reports the handle that can identify the assembly. This value is the same as the assembly handle returned when the *Assembly* was added to the assembly queue.

Syntax

```
int AssemblyHandle [ get ]
```

Assembly.CommandLine Property

[Write-only] This property sets command line options for the assembly as if it were started with a particular command line from the library. Setting this property to a string is the same as if the string were passed in from the library entry.

Assigning a string to this property actually appends the new command line string to any existing command line options already set. Assigning an empty string ("") to this property removes any command line options already set.

Syntax

```
string CommandLine [ set ]
```

Example

The following Visual C# example sets the *CommandLine* property:

```
public class ExampleCode
{
    static void Main()
    {
        HotDocs.Application app = new HotDocs.Application();
        HotDocs.Assembly asm = new HotDocs.Assembly();

        asm.TemplatePath = @"C:\temp\Demo Employment Agreement.docx";
        asm.CommandLine = @"/af=C:\temp\EmployeeAnswers.anx";
        asm.Visible = true;
        app.Assemblies.Add(asm);
    }
}
```

Assembly.DocumentPath Property

[Read/Write] This property controls the location where the assembled document file is written. It is the file path for the destination file.

Syntax

```
string DocumentPath [ set, get ]
```

Assembly.Hwnd Property

[Read-only] This property returns the window handle for the HotDocs Assembly interface.

Syntax

```
int Hwnd [ get ]
```

Assembly.KeepInQueue Property

[Read/Write] This Boolean property controls the behavior of the assembly queue. If it is **true** when the assembly is complete, a reference will remain in the Assembly Queue window. If it is **false**, all references to the assembly will be destroyed when the assembly is complete.

This property cannot be used to change the behavior of the active assembly (e.g., *Application.ActiveAssembly*). You must set this property before the assembly begins.

Syntax

```
bool KeepInQueue [ set, get ]
```

Assembly.Map Property

[Read/Write] This property controls which *VarMap* object is used during assembly.

HotDocs does not do anything with this *VarMap* object, but it keeps this reference for the convenience of integrations. When an integration needs to provide an answer, it can look up the variable in the *VarMap* object and see what mapping was provided by the user.

Syntax

```
HotDocs._VarMap Map [ set, get ]
```

Assembly.PrintwhenComplete Property

[Read/Write] This Boolean property controls whether the document is automatically printed when the assembly completes. It controls the same behavior as specifying a Print (/pr) option at the command line.

Syntax

```
bool PrintWhenComplete [ set, get ]
```

Assembly.PromptToSaveDocument Property

[Read/Write] This is a Boolean property that controls whether HotDocs will prompt to save a copy of the assembled document at the end of the assembly. The initial value is the value stored in the Document Assembly folder in HotDocs Options (**Tools** > **Options** > **Document Assembly** > **Prompt to save document when closing assembly window**). If the integrator changes the *PromptToSaveDocument* property, HotDocs will disregard the setting in **HotDocs Options** in favor of the value set in the integration.

This property was introduced with the release of HotDocs 6.1 SP1.

Syntax

```
bool PromptToSaveDocument [ set, get ]
```

Assembly.QuestionSummaryPath Property

[Read/Write] This property controls the location to which the question summary file is written. If this property is set when the assembly starts, a question summary will be written out as an HTML document when the assembly is complete.

Syntax

```
string QuestionSummaryPath [ set, get ]
```

Assembly.ShowAnswerFileDialog Property

[Read/Write] This Boolean property controls whether or not the **Save Answers** dialog box appears at the end of an assembly.

Syntax

```
bool ShowAnswerFileDialog [ set, get ]
```

Assembly.Status Property

[Read-only] This property tells what state the *Assembly* object is in.

It can be one or more of the following values from the HDASSEMBLYSTATUS enumeration:

Name	Value	Description
HDASMSTATUSCONFIRMED	2	The *Assembly* object is awaiting assembly.
HDASMSTATUSASSEMBLING	4	The *Assembly* object is being assembled.
HDASMSTATUSCOMPLETED	8	The *Assembly* object has been assembled and the assembly completed with no errors.
HDASMSTATUSERROR	256	Assembly is complete, but an error occurred during assembly and it did not complete successfully.

Because it can be combination of the above values, an integrator needs to check the bitwise AND of the value and a mask to determine if one of the values is set.

Syntax

```
HotDocs.HDASSEMBLYSTATUS Status [ get ]
```

Example

The following Visual C# example displays the status of each assembly in the queue:

```
public class ExampleCode
{
```

```
    static void Main()
    {
        HotDocs.Application app = new HotDocs.Application();

        foreach (HotDocs.Assembly assembly in app.Assemblies)
        {
            MessageBox.Show("The status for " + assembly.TemplateTitle + "is
" + assembly.Status);
        }
    }
}
```

Assembly.SuppressUnansweredWarning Property

[Read/Write] This is a Boolean property that sets the **Display unanswered warning before saving documents** option for the given assembly. If this property is **false**, then HotDocs will display a warning before the user saves the document or sends the document to the word processor if there are variables which are used but not answered. If this property is **true**, then all such warnings are suppressed.

Syntax

```
bool SuppressUnansweredWarning [ set, get ]
```

Assembly.TemplateDesc Property

[Read/Write] This property contains the description for the template that will be used to assemble a document. This property is for the integration programmer's convenience. It is set but is not used by HotDocs internally.

Syntax

```
string TemplateDesc [ set, get ]
```

Assembly.TemplatePath Property

[Read/Write] This property contains the file system path for the template that will be used to assemble a document.

Syntax

```
string TemplatePath [ set, get ]
```

Assembly.TemplateTitle Property

[Read/Write] This property contains the title for the template that will be used to assemble a document.

Syntax

```
string TemplateTitle [ set, get ]
```

Assembly.Visible Property

[Read/Write] This Boolean property determines if the assembly interface will be visible to the user. This property must be set before assembly starts or it will have no effect. If Visible returns **false**, then it has the same effect as specifying the No Assembly Window (/nw) option on the command line.

Syntax

```
bool Visible [ set, get ]
```

Assembly.OnAssemblyCompleteEvent Event

This event is fired when the assembly completes.

Syntax

```
void OnAssemblyCompleteEvent ( )
```

Assembly.OnAssemblyStartEvent Event

This event is fired when the assembly starts.

Syntax

```
void OnAssemblyStartEvent ( )
```

Assembly.OnCanOpenFile Event

This event is fired when a file can be opened.

Syntax

```
void OnCanOpenFile ( string FileName, ref bool bCanOpen, ref bool bOverride )
```

Parameters	Description
FileName	A file name.
bCanOpen	Indicates if the file can be opened.
bOverride	Return parameter: • *override == true means that HotDocs will not open the file. • *override == false means that HotDocs will continue normally.

Return Value

An integer indicating the index of the new *Value* within the *Answer* object.

Assembly.OnCloseAssemblyInterfaceEvent Event

This event is fired when assembly interface actually closes.

Syntax

```
void OnCloseAssemblyInterfaceEvent ( )
```

Assembly.OnErrorEvent Event

This event is fired when an error occurs. By returning true for the override parameter, the integration can tell HotDocs not to display any error messages, allowing the integration to either display its own error message or deal with the error silently.

Syntax

```
void OnErrorEvent ( int errCode, string errMesg, out bool override )
```

Parameters	Description
errCode	Numerical error code.
errMesg	Text description of error.
override	Return parameter: • *override == true means that HotDocs will not display any user interface indicating that an error occurred. • *override == false means that HotDocs will display the normal error notification.

Assembly.OnFileOpen Event

This event is fired when a file is opened.

Syntax

```
void OnFileOpen ( HotDocs.HDAUI hdEvent, ref string FileName, ref string displayName,
ref bool vbDoDefaultOpen )
```

Parameters	Description
hdEvent	The user interface item selected. It is one of the values from the HAUI enumeration
FileName	The name of the file being opened.
displayName	A descriptive name of the file being opened.
bOverride	Return parameter:

- *override == true means that HotDocs will not open the file.
- *override == false means that HotDocs will continue normally.

Assembly.OnFileSave Event

This event is fired when a file is saved.

Syntax

```
void OnFileSave ( HotDocs.HDAUI hdEvent, ref string FileName, ref string displayName,
ref bool bOverride, ref bool vbDoDefaultSave )
```

Parameters	Description
hdEvent	The user interface item selected. It is one of the values from the HAUI enumeration
FileName	The name of the file being saved.
displayName	A descriptive name of the file being opened.
bOverride	Return parameter: • *override == true means that HotDocs will not perform the file save. • *override == false means that HotDocs will continue normally.
vbDoDefaultSave	true/false (Boolean) value

Assembly.OnFileSelectEvent Event

This event is fired when a file is selected.

Syntax

```
void OnFileSelectEvent ( HotDocs.HDAUI hdEvent, ref string FileName, bool
bSelectOnly, ref bool override )
```

Parameters	Description

hdEvent	The event that has occurred.
FileName	The name of the selected file.
bSelectOnly	true/false (Boolean) value
override	Return parameter: • *override == true means that HotDocs will not perform the default file selection operation. • *override == false means that HotDocs will continue normally.

Assembly.OnGetAnswerFileDisplayName Event

This event is fired when HotDocs gets the name of an answer file to display.

Syntax

```
void OnGetAnswerFileDisplayName ( ref string FileName, ref string displayName )
```

Parameters	Description
FileName	The name of the answer file being displayed.
displayName	A descriptive name for the answer file.

Assembly.OnGetMRUInfo Event

This event is fired when HotDocs gets information from the most recently used (MRU) list.

Syntax

```
void OnGetMRUInfo ( ref string FileName, ref string displayName, ref bool bOverride )
```

Parameters	Description
FileName	The name of a file being retrieved.
displayName	The descriptive name of a file being retrieved.

bOverride	Return parameter: • *override == true means that HotDocs will not get the information from the MRU list. • *override == false means that HotDocs will continue normally.

Assembly.OnNeedAnswerEvent Event

This event is fired when an answer value is needed by the assembly, but not found in the *AnswerCollection* answer set. It allows the integration to provide answers as they are needed, rather than trying to provide all the answers before the assembly starts.

This event fires once for each answer, not for each value in an answer. This means that all the values for repeated answers must be set during the first firing of this event for each answer.

Because of the complexity of answer repeat indexing, you should avoid using this event when working with nested repeats.

Syntax

```
void OnNeedAnswerEvent ( int AssemblyHandle, HotDocs.Answer pAnswer )
```

Parameters	Description
assemblyHandle	The assemblyhandle returned when the *Assembly* object was added to the queue.
pAnswer	The *Answer* object that represents the answer that needs a value. If the integration wants to provide a value for this answer, the regular *Answer* object methods are used to set the value. It can be queried to determine which answer is needed.

Assembly.OnPostCloseAnswerFile Event

This event is fired after the answer file is closed.

Syntax

```
void OnPostCloseAnswerFile ( )
```

Assembly.OnPostSaveDocumentEvent Event

This event is fired after a document is saved. The document can be an assembled document, a question summary document, or an answer summary document.

Syntax

```
void OnPostSaveDocumentEvent ( HotDocs.HDOUTPUTTYPE outputType, string docPath )
```

Parameters	Description
outputType	The type of document that was saved. It can be one of the following values from the HDOUTPUTTYPE enumeration: • HD_OUTPUT_DOCUMENT • HD_OUTPUT_ANSWERSUMMARY • HD_OUTPUT_QUESTIONSUMMARY
docPath	The file system path where the file was saved.

Assembly.OnPreCloseAnswerFile Event

This event is fired when HotDocs prepares to close an answer file.

Syntax

```
void OnPreCloseAnswerFile ( string FileName )
```

Parameters	Description
FileName	The name of the answer file being closed.

Assembly.OnPreSaveDocumentEvent Event

This event is fired prior to saving the document. By setting *showui = false, the integration can prevent the user interface relating to saving the document from showing. By setting *override = true, the integration can prevent the save from happening.

Syntax

```
void OnPreSaveDocumentEvent ( ref string pathName, HotDocs.HDOUTPUTTYPE outputType,
out bool showui, out bool override )
```

Parameters	Description
pathName	The path and file name that HotDocs believes the file should be saved to. This might be an empty string ("") if there are no indicators. The integration can change this value. If the value is changed, HotDocs will either save the file to the new value (if no user interface is shown), or suggest the new value as the location to save (if the user interface is shown).
outputType	The type of document to be saved. It can be one of the following values from the HDOUTPUTTYPE enumeration: • HD_OUTPUT_DOCUMENT • HD_OUTPUT_ANSWERSUMMARY • HD_OUTPUT_QUESTIONSUMMARY
showui	Return parameter. Specifies if HotDocs should display any user interface associated with the save: • *showui == true means the user interface is shown. • *showui == false means the user interface is not shown.
override	Return parameter: • *override == true means that HotDocs will not perform the file save. • *override == false means that HotDocs will continue normally.

Assembly.OnUserInterfaceEvent Event

This event is fired when the user selects certain options at the assembly interface. This can be useful if the integration wants to override a particular HotDocs feature or command.

Syntax

```
void OnUserInterfaceEvent ( HotDocs.HDAUI hdEvent, out bool override )
```

Parameters	Description
hdEvent	The user interface item selected. It will be one of the members of the HDAUI enumeration.
override	Return parameter: • *override == true will cancel the user interface action. • *override == false will allow the action to continue normally.

Assembly.OnUserMenuItemClickedEvent Event

This event is fired when the user selects an integration-defined menu item. (See *Assembly.AddUserMenuItem* Method.)

Syntax

```
void OnUserMenuItemClickedEvent ( int menuHandle )
```

Parameters	Description
menuHandle	The handle for the user-defined menu. This is the handle returned from *_Assembly.AddUserMenuItem()*.

Assembly.PostSaveAnswersEvent Event

In HotDocs 10 and earlier this event was listed as *OnPostSaveAnswersEvent*. It is intended to eliminate the need to pass the *AnswerCollection* by pointer and instead wrap it in a object. This should help Visual Basic programmers deal with the VARIANT event. This event should be fired at the same time as *OnPostSaveAnswersEvent*.

This event is fired after an answer file is saved.

Syntax

```
void PostSaveAnswersEvent ( HotDocs._AnswerCollection answers )
```

Parameters	Description
answers	The answer collection that was saved to disk.

Assembly.PreSaveAnswersEvent Event

In HotDocs 10 and earlier this event was listed as *OnPreSaveAnswersEvent*. It is intended to eliminate the need to pass the *AnswerCollection* by pointer and instead wrap it in a object. This should help Visual Basic programmers deal with the VARIANT event. This event should be fired at the same time as *OnPreSaveAnswersEvent*.

This event is fired after the user has indicated he or she wants to save the answers, but before the actual save occurs. By setting *override == true, the integration can prevent the save from happening.

Syntax

```
void PreSaveAnswersEvent ( HotDocs._AnswerCollection answers, out bool override )
```

Parameters	Description
answers	The *AnswerCollection* object that represents the answer file about to be saved.
override	Return parameter: • *override == true means that HotDocs will not perform the answer file save. • *override == false means that HotDocs will continue normally.

HotDocs.AssemblyCollectionClass Object

HotDocs.AssemblyCollectionClass Object

This object represents the collection of *Assembly* objects awaiting assembly in the HotDocs assembly queue. The integration cannot create this object. To use it, you must retrieve the *Assemblies* property from the *Application* object.

General Information

ProgID:	HotDocs.AssemblyCollection.11.0 HotDocs.AssemblyCollection (version-independent)
CLSID:	{C5038B17-D521-4f4b-91C1-62F55A622D25}

The following table shows the name and IID for each interface, as well as the version of HotDocs in which it was introduced. The primary interface and the main public interface exposed by this object is *AssemblyCollection*.

Name	IID	Added in
AssemblyCollection	{BF92F712-50A9-4b2c-9D14-0956C1883C5A}	Added in HotDocs 6.0

Methods

Method	Description
Add	This method adds a *HotDocs.Assembly* object to the HotDocs assembly queue.
AddToQueue	This method creates a new *HotDocs.Assembly* object and adds it to the HotDocs assembly queue with a set of commonly used default values.
Clear	This method removes all the *Assembly* objects (except the currently executing assembly) from the assembly collection.
FindByHandle	When given an *assemblyHandle*, this method returns the index for the corresponding *Assembly* object.
Insert	This method inserts a *HotDocs.Assembly* object into the assembly queue at the specified position. This method is similar to the *AssemblyCollection.Add()* method, except it allows the integration to specify the location of the object in the queue.
Item	This method retrieves the Indexth item from the collection and returns it to the caller.
Move	This method moves the oldindex item in the collection to the newindex position.
Remove	This method removes the *Assembly* object at the Indexth position in the assembly queue. However, this method cannot remove the *Assembly* object for the active assembly.

Properties

Property	Description
Application	[Read-only] This property returns a reference to the *Application* object.
Count	[Read-only] This property returns the number of items in the *AssemblyCollectionClass* object.

AssemblyCollectionClass.Add Method

This method adds a *HotDocs.Assembly* object to the HotDocs assembly queue.

Syntax

```
int Add ( HotDocs.Assembly newVal)
```

Parameters	Description
newVal	The *Assembly* object to add to the collection.

Return Value

A handle to identify the *Assembly* object.

AssemblyCollectionClass.AddToQueue Method

This method creates a new *HotDocs.Assembly* object and adds it to the HotDocs assembly queue with a set of commonly used default values.

This method is simply a shortcut for adding an assembly to the queue.

Syntax

```
int AddToQueue ( string templateFilePath, bool Visible, string docPath, string
answerPath )
```

Parameters	Description
templateFilePath	The file path and name for the template to be used for assembly. This must point to a HotDocs template or the method will fail.

Visible	Sets the Visible property for the *Assembly* object.
docPath	The location for the assembled document. This is used for the *Assembly.DocumentPath* property.
answerPath	The file path and name of the answer file to use in the assembly.

Return Value

A handle to identify the *Assembly* object.

AssemblyCollectionClass.Clear Method

This method removes all the *Assembly* objects (except the currently executing assembly) from the assembly collection.

Syntax

```
void Clear ( )
```

AssemblyCollectionClass.FindByHandle Method

When given an assemblyHandle, this method returns the index for the corresponding *Assembly* object.

Syntax

```
int FindByHandle ( int AssemblyHandle )
```

Parameters	Description
assemblyHandle	The handle returned when the *Assembly* object was added to the queue.

Return Value

The index for the *Assembly* object corresponding to the assemblyHandle.

AssemblyCollectionClass.Insert Method

This method inserts a *HotDocs.Assembly* object into the assembly queue at the specified position. This method is similar to the *AssemblyCollection.Add()* method, except it allows the integration to specify the location of the object in the queue.

Syntax

```
int Insert ( int index, HotDocs.Assembly newVal )
```

Parameters	Description
Index	The position in the assembly queue where the object should be inserted.
newval	The new *Assembly* object.

Return Value

A handle to identify the *Assembly* object.

AssemblyCollectionClass.Item Method

This method retrieves the Indexth item from the collection and returns it to the caller.

Syntax

```
HotDocs.AssemblyItem ( int index )
```

Parameters	Description
Index	The position of the desired item.

Return Value

The specified *HotDocs.Assembly* object.

AssemblyCollectionClass.Move Method

This method moves the oldindex item in the collection to the newindex position.

Syntax

```
void Move ( int oldindex, int newindex )
```

Parameters	Description
oldindex	The index of the *Assembly* object to move.
newindex	The new position for the *Assembly* object.

AssemblyCollectionClass.Remove Method

This method removes the *Assembly* object at the Indexth position in the assembly queue. However, this method cannot remove the *Assembly* object for the active assembly.

Syntax

```
void Remove ( int index )
```

Parameters	Description
index	The position of the *Assembly* object to remove.

AssemblyCollectionClass.Application Property

[Read-only] This property returns a reference to the *Application* object.

Since there is only one *Application* object on a machine at a time, this property will return a reference to the same object as the *Application* property on any other object in HotDocs, and a reference to the same object as if you created a new *HotDocs.Application* object.

Syntax

```
HotDocs._Application2 Application [ get ]
```

AssemblyCollectionClass.Count Property

[Read-only] This property returns the number of items in the *AssemblyCollectionClass* object.

Syntax

```
int Count [ get ]
```

HotDocs.Component Object

HotDocs.Component Object

This object represents a HotDocs variable or other component stored in a HotDocs component file.

General Information

ProgID:	HotDocs.Component.11.0 HotDocs.Component (version-independent)
CLSID:	{50DED91B-A330-48BA-B7F4-F48803D63D3F}

The following table shows the name and IID for each interface, as well as the version of HotDocs in which it was introduced. The primary interface and the main public interface exposed by this object is *_Component2*.

Name	IID	Added in
_Component	{98A99D7D-FF5D-4A4F-A154-BC8B9FFD597E}	Added in HotDocs 6.0
_Component2	{98A99D7E-FF5D-4A4F-A154-BC8B9FFD597E}	Added in HotDocs 2005 SP1
_Component3	{98A99D7F-FF5D-4A4F-A154-BC8B9FFD597E}	Added in HotDocs 2006

Methods

Method	Description
DisplayEditor	This method displays a user interface for modifying a component's properties. For example, if the component is a Text variable, this method displays the **Text Variable Editor**.

Properties

Property	Description
Application	[Read-only] This property returns a reference to the *Application* object.
DBName	[Read-only] This is a string property that tells the name of the database component (if any) that links this variable component to a field in a database table.
DialogName	[Read-only] This is a string property that specifies the name of the dialog in which this variable component appears.
HelpText	[Read-only] This is a string property that tells what resource text has been included for a variable.
Name	[Read-only] This is a string property that tells the name of the variable.
Prompt	[Read-only] This is a string property that tells the prompt of the variable.
Properties	[Read-only] This property returns a *ComponentProperties* object, which is a collection of all properties associated with the Component object. Each property in the collection is in turn represented by a *ComponentProperty* object.
Title	[Read-only] This property returns the title of the component.
Type	[Read-only] This HDVARTYPE property tells which type the variable is. This can be one of the values from the HDVARTYPE enumeration.

Component.DisplayEditor Method

This method displays a user interface for modifying a component's properties. For example, if the component is a Text variable, this method displays the **Text Variable Editor**.

Syntax

```
void DisplayEditor ( int parentWnd )
```

Parameters	Description
parentWnd	The parent window of the editor window to be displayed.

Component.Application Property

[Read-only] This property returns a reference to the *Application* object.

Since there is only one *Application* object on a machine at a time, this property will return a reference to the same object as the *Application* property on any other object in HotDocs, and a reference to the same object as if you created a new *HotDocs.Application* object.

Syntax

```
HotDocs._Application2 Application [ get ]
```

Component.DBName Property

[Read-only] This is a string property that tells the name of the database component (if any) that links this variable component to a field in a database table.

Syntax

```
string DBName [ get ]
```

Example

The following Visual C# example displays information about each of the variable components in a given component file:

```
public class ExampleCode
{
    static void Main()
    {
        HotDocs.ComponentCollection ccl = new HotDocs.ComponentCollection();

        ccl.Open(@"C:\Documents\HotDocs\Templates\demoempl.cmp");

        HotDocs.HDVARTYPE varType = HotDocs.HDVARTYPE.HD_TEXTTYPE;
        HotDocs.Component comp;

        for (int i=0; i < ccl.Count; i++)
        {
            comp = ccl.Item(i, ref varType) as HotDocs.Component;

            Console.WriteLine("Name: {0}",comp.Name);
            Console.WriteLine("Type: {0}", comp.Type);
            Console.WriteLine("Title:{0}", comp.Title);
            Console.WriteLine("Prompt: {0}", comp.Prompt);
```

```
            Console.WriteLine("Help Text: {0}", comp.HelpText);
            Console.WriteLine("Dialog Name: {0}", comp.DialogName);
            Console.WriteLine("Database Name: {0}", comp.DBName);
            Console.WriteLine();
        }
        Console.ReadKey();
    }
}
```

Component.DialogName Property

[Read-only] This is a string property that specifies the name of the dialog in which this variable component appears.

Syntax

```
string DialogName [ get ]
```

Example

The following Visual C# example displays information about each of the variable components in a given component file:

```
public class ExampleCode
{
    static void Main()
    {
        HotDocs.ComponentCollection ccl = new HotDocs.ComponentCollection();

        ccl.Open(@"C:\Documents\HotDocs\Templates\demoempl.cmp");

        HotDocs.HDVARTYPE varType = HotDocs.HDVARTYPE.HD_TEXTTYPE;
        HotDocs.Component comp;

        for (int i=0; i < ccl.Count; i++)
        {
            comp = ccl.Item(i, ref varType) as HotDocs.Component;

            Console.WriteLine("Name: {0}",comp.Name);
            Console.WriteLine("Type: {0}", comp.Type);
            Console.WriteLine("Title:{0}", comp.Title);
            Console.WriteLine("Prompt: {0}", comp.Prompt);
            Console.WriteLine("Help Text: {0}", comp.HelpText);
            Console.WriteLine("Dialog Name: {0}", comp.DialogName);
            Console.WriteLine("Database Name: {0}", comp.DBName);
            Console.WriteLine();
        }
        Console.ReadKey();
```

```
    }
}
```

Component.HelpText Property

[Read-only] This is a string property that tells what resource text has been included for a variable.

Syntax

```
string HelpText [ get ]
```

Example

The following Visual C# example displays information about each of the variable components in a given component file:

```
public class ExampleCode
{
    static void Main()
    {
        HotDocs.ComponentCollection ccl = new HotDocs.ComponentCollection();

        ccl.Open(@"C:\Documents\HotDocs\Templates\demoempl.cmp");

        HotDocs.HDVARTYPE varType = HotDocs.HDVARTYPE.HD_TEXTTYPE;
        HotDocs.Component comp;

        for (int i=0; i < ccl.Count; i++)
        {
            comp = ccl.Item(i, ref varType) as HotDocs.Component;

            Console.WriteLine("Name: {0}",comp.Name);
            Console.WriteLine("Type: {0}", comp.Type);
            Console.WriteLine("Title:{0}", comp.Title);
            Console.WriteLine("Prompt: {0}", comp.Prompt);
            Console.WriteLine("Help Text: {0}", comp.HelpText);
            Console.WriteLine("Dialog Name: {0}", comp.DialogName);
            Console.WriteLine("Database Name: {0}", comp.DBName);
            Console.WriteLine();
        }
        Console.ReadKey();
    }
}
```

Component.Name Property

[Read-only] This is a string property that tells the name of the variable.

Syntax

```
string Name [ get ]
```

Example

The following Visual C# example displays information about each of the variable components in a given component file:

```
public class ExampleCode
{
    static void Main()
    {
        HotDocs.ComponentCollection ccl = new HotDocs.ComponentCollection();

        ccl.Open(@"C:\Documents\HotDocs\Templates\demoempl.cmp");

        HotDocs.HDVARTYPE varType = HotDocs.HDVARTYPE.HD_TEXTTYPE;
        HotDocs.Component comp;

        for (int i=0; i < ccl.Count; i++)
        {
            comp = ccl.Item(i, ref varType) as HotDocs.Component;

            Console.WriteLine("Name: {0}",comp.Name);
            Console.WriteLine("Type: {0}", comp.Type);
            Console.WriteLine("Title:{0}", comp.Title);
            Console.WriteLine("Prompt: {0}", comp.Prompt);
            Console.WriteLine("Help Text: {0}", comp.HelpText);
            Console.WriteLine("Dialog Name: {0}", comp.DialogName);
            Console.WriteLine("Database Name: {0}", comp.DBName);
            Console.WriteLine();
        }
        Console.ReadKey();
    }
}
```

Component.Prompt Property

[Read-only] This is a string property that tells the prompt of the variable.

Syntax

```
string Prompt [ get ]
```

Example

The following Visual C# example displays information about each of the variable components in a given component file:

```
public class ExampleCode
{
    static void Main()
    {
        HotDocs.ComponentCollection ccl = new HotDocs.ComponentCollection();

        ccl.Open(@"C:\Documents\HotDocs\Templates\demoempl.cmp");

        HotDocs.HDVARTYPE varType = HotDocs.HDVARTYPE.HD_TEXTTYPE;
        HotDocs.Component comp;

        for (int i=0; i < ccl.Count; i++)
        {
            comp = ccl.Item(i, ref varType) as HotDocs.Component;

            Console.WriteLine("Name: {0}",comp.Name);
            Console.WriteLine("Type: {0}", comp.Type);
            Console.WriteLine("Title:{0}", comp.Title);
            Console.WriteLine("Prompt: {0}", comp.Prompt);
            Console.WriteLine("Help Text: {0}", comp.HelpText);
            Console.WriteLine("Dialog Name: {0}", comp.DialogName);
            Console.WriteLine("Database Name: {0}", comp.DBName);
            Console.WriteLine();
        }
        Console.ReadKey();
    }
}
```

Component.Properties Property

[Read-only] This property returns a *ComponentProperties* object, which is a collection of all properties associated with the *Component* object. Each property in the collection is in turn represented by a *ComponentProperty* object.

Syntax

```
HotDocs.ComponentProperties Properties [ get ]
```

Component.Type Property

[Read-only] This HDVARTYPE property tells which type the variable is. This can be one of the values from the HDVARTYPE enumeration.

Syntax

```
HotDocs.HDVARTYPE Type [ get ]
```

Example

The following Visual C# example displays information about each of the variable components in a given component file:

```
public class ExampleCode
{
    static void Main()
    {
        HotDocs.ComponentCollection ccl = new HotDocs.ComponentCollection();

        ccl.Open(@"C:\Documents\HotDocs\Templates\demoempl.cmp");

        HotDocs.HDVARTYPE varType = HotDocs.HDVARTYPE.HD_TEXTTYPE;
        HotDocs.Component comp;

        for (int i=0; i < ccl.Count; i++)
        {
            comp = ccl.Item(i, ref varType) as HotDocs.Component;

            Console.WriteLine("Name: {0}",comp.Name);
            Console.WriteLine("Type: {0}", comp.Type);
            Console.WriteLine("Title:{0}", comp.Title);
            Console.WriteLine("Prompt: {0}", comp.Prompt);
            Console.WriteLine("Help Text: {0}", comp.HelpText);
            Console.WriteLine("Dialog Name: {0}", comp.DialogName);
            Console.WriteLine("Database Name: {0}", comp.DBName);
            Console.WriteLine();
        }
        Console.ReadKey();
    }
}
```

Component.Title Property

[Read-only] This property returns the title of the component.

This property was introduced with the release of HotDocs 2005 SP1.

Syntax

```
string Title [ get ]
```

Example

The following Visual C# example displays information about each of the variable components in a given component file:

```
public class ExampleCode
{
    static void Main()
    {
        HotDocs.ComponentCollection ccl = new HotDocs.ComponentCollection();

        ccl.Open(@"C:\Documents\HotDocs\Templates\demoempl.cmp");

        HotDocs.HDVARTYPE varType = HotDocs.HDVARTYPE.HD_TEXTTYPE;
        HotDocs.Component comp;

        for (int i=0; i < ccl.Count; i++)
        {
            comp = ccl.Item(i, ref varType) as HotDocs.Component;

            Console.WriteLine("Name: {0}",comp.Name);
            Console.WriteLine("Type: {0}", comp.Type);
            Console.WriteLine("Title:{0}", comp.Title);
            Console.WriteLine("Prompt: {0}", comp.Prompt);
            Console.WriteLine("Help Text: {0}", comp.HelpText);
            Console.WriteLine("Dialog Name: {0}", comp.DialogName);
            Console.WriteLine("Database Name: {0}", comp.DBName);
            Console.WriteLine();
        }
        Console.ReadKey();
    }
}
```

HotDocs.ComponentCollection Object

HotDocs.ComponentCollection Object

This object represents a collection of *HotDocs.Component* objects. The backing data source for a *ComponentCollection* is a HotDocs component file.

General Information

ProgID:	HotDocs.ComponentCollection.11.0 HotDocs.ComponentCollection (version-independent)
CLSID:	{2C2098F9-D471-4806-9759-504F65C4171B}

The following table shows the name and IID for each interface, as well as the version of HotDocs in which it was introduced. The primary interface and the main public interface exposed by this object is _ComponentCollection4.

Name	IID	Added in
_ComponentCollection	{2B539DA7-2727-4AF0-A166-009CA0F48A5A}	Added in HotDocs 6.0
_ComponentCollection2	{2B539DA8-2727-4AF0-A166-009CA0F48A5A}	Added in HotDocs 6.1
_ComponentCollection3	{2B539DA9-2727-4AF0-A166-009CA0F48A5A}	Added in HotDocs 2005
_ComponentCollection4	{2B539DAA-2727-4AF0-A166-009CA0F48A5A}	Added in HotDocs 2006

Methods

Method	Description
Create	This method creates a new, empty component file. For example, you may want to create a new component file so you can seed it with data (variables) that correspond to your integrating application. If filePath refers to a file that already exists, an error is returned.
CreateComponent	This method allows you to create virtually any type of HotDocs variable or component in a component file. Once the component is created, its properties can be set and modified using the *ComponentProperties* object. If the component already exists in the component collection, this method does nothing.
CreateVariable	This method creates a new variable in the open component file. You can only use it to create Text, Number, Date, and True/False variables. (Other variable types cannot be created using this method.) Also, if a variable with the same name already exists in the component file, nothing will happen. You can also use the *CreateComponent* method, which allows you to create many other types of variables and components.
Item	This method retrieves a *Component* object from a *ComponentCollection*.
Open	This method opens a HotDocs component file and populates the *ComponentCollection* with the variables from it. If the component file opened

	by this method points to separate shared component file, the shared component file will be opened and used to populate the *ComponentCollection*.
OpenBase	This method opens a HotDocs component file and populates the *ComponentCollection* with the variables from it. Unlike the *Open* method, however, this method does not open shared component files. For example, if you open component file A that points to component file B, the *ComponentCollection* would contain the variables from component file A.
OpenForEdit	This method opens a component file for editing.

Properties

Property	Description
Application	[Read-only] This property returns a reference to the *Application* object.
Count	[Read-only] This property returns the number of *Component* objects in the *ComponentCollection*.
FileName	[Read-only] Returns the file name of the HotDocs component file. To specify the path for the backing component file, pass the file path and name to *_ComponentCollection.Open()*.
OnlyVariables	[Read/Write] This property acts as a filter for the component collection to limit the items to variables or all components. If it is true, the *Count* will include only variables, and if it is false, it will include all variables and components. Likewise, if you use an integer as the index with the Item method, the components you can retrieve will be either all components or only variables. (Even with this property set to true, you can refer to any component using the Item method if you specify the component name and type instead of just the index number.) The default value for this property is true.
ReadOnly	[Read-only] This property indicates whether the component collection (component file) is read-only or writable.

ComponentCollection.Create Method

This method creates a new, empty component file. For example, you may want to create a new component file so you can seed it with data (variables) that correspond to your integrating application. If filePath refers to a file that already exists, an error is returned.

This method was introduced with the release of HotDocs 6.1.

Syntax

```
void Create ( string filePath )
```

Parameters	Description
filePath	The file system path of the new component file.

Example

The following Visual C# example creates a new component file and creates nine Text variables in it.

```
public class ExampleCode
{
    static void Main()
    {
        HotDocs.ComponentCollection ccl = new HotDocs.ComponentCollection();

        ccl.Create(@"C:\temp\Createdfile.cmp");
        for (int i = 1; i < 10; i++)
        ccl.CreateVariable("Variable " +
i.ToString(),HotDocs.HDVARTYPE.HD_TEXTTYPE, "Prompt for Variable " +
i.ToString());
    }
}
```

ComponentCollection.CreateComponent Method

This method allows you to create virtually any type of HotDocs variable or component in a component file. Once the component is created, its properties can be set and modified using the *ComponentProperties* object. If the component already exists in the component collection, this method does nothing.

This method was introduced with the release of HotDocs 2006.

Syntax

```
void CreateComponent ( string componentName, HotDocs.HDVARTYPE componentType )
```

Parameters	Description
componentName	The name of the component to create.

componentType	The type of the new component.

Example

The following Visual C# example creates a new component and then displays the names of each property in its *ComponentProperties* collection.

```
public class ExampleCode
{
    static void Main()
    {
        HotDocs.ComponentCollection ccl = new HotDocs.ComponentCollection();

        ccl.OpenForEdit(@"C:\Documents\HotDocs\Templates\demoempl.cmp");

        HotDocs.HDVARTYPE varType = HotDocs.HDVARTYPE.HD_TEXTTYPE;
        HotDocs.Component comp;

        ccl.CreateComponent("Attorney Name", varType);

        comp = ccl.Item("Attorney Name", ref varType) as HotDocs.Component;

        for (int i = 0; i < comp.Properties.Count; i++)
            Console.WriteLine(comp.Properties.Item(i).Name);

        Console.ReadKey();
    }
}
```

ComponentCollection.CreateVariable Method

This method creates a new variable in the open component file. You can only use it to create Text, Number, Date, and True/False variables. (Other variable types cannot be created using this method.) Also, if a variable with the same name already exists in the component file, nothing will happen. You can also use the *CreateComponent* method, which allows you to create many other types of variables and components.

This method was introduced with the release of HotDocs 6.1.

Syntax

```
void CreateVariable ( string variableName, HotDocs.HDVARTYPE varType, string Prompt )
```

Parameters	Description

variableName	The name of the variable to create.
varType	The type of variable to create. This value can be: • HD_TEXTTYPE • HD_NUMBERTYPE • HD_DATETYPE • HD_TRUEFALSETYPE
prompt	[optional] The prompt for the newly created variable.

Example

The following Visual C# example creates a new component file and creates nine Text variables in it.

```
public class ExampleCode
{
    static void Main()
    {
        HotDocs.ComponentCollection ccl = new HotDocs.ComponentCollection();

        ccl.Create(@"C:\temp\Createdfile.cmp");

        for (int i = 1; i < 10; i++)
            ccl.CreateVariable("Variable " + i.ToString(),
HotDocs.HDVARTYPE.HD_TEXTTYPE,Prompt for Variable " + i.ToString());
    }
}
```

ComponentCollection.Item Method

This method retrieves a *Component* object from a *ComponentCollection*.

Syntax

```
HotDocs._Component Item ( object index, ref HotDocs.HDVARTYPE varType )
```

Parameters	Description
index	This parameter can be either a number or a string. If it is a number, it represents the position of the desired *Component* object in the collection. If it is a string, it represents the variable name of the desired *Component* object. Since HotDocs variables are identified by name and type, the second parameter, vartype, must be set to the correct HDVARTYPE value when calling

	this method with a string for the index parameter. If the *ComponentCollection.OnlyVariables* property is **true**, a numerical index can only access components that are Text, Number, Date, True/False, and Multiple Choice variables. If you want to access other component types with a numerical index, set OnlyVariables to **false**.
vartype	[optional] When the index parameter is a number, this parameter returns the correct HDVARTYPE for the retrieved Component object. When the index parameter is a string, this parameter must be the HDVARTYPE for the component with that variable name.

Return Value

The *Component* object requested.

ComponentCollection.Open Method

This method opens a HotDocs component file and populates the *ComponentCollection* with the variables from it. If the component file opened by this method points to separate shared component file, the shared component file will be opened and used to populate the *ComponentCollection*.

Syntax

```
void Open ( string componentFileName )
```

Parameters	Description
componentFileName	The file system path of a HotDocs template, clause library, or component file. In the case of a template or clause library, this method will open the associated component file.

You can close an open component file by disposing the *ComponentCollection* object.

ComponentCollection.OpenBase Method

This method opens a HotDocs component file and populates the *ComponentCollection* with the variables from it. Unlike the *Open* method, however, this method does not open shared component files. For

example, if you open component file A that points to component file B, the *ComponentCollection* would contain the variables from component file A.

Syntax

```
void OpenBase ( string FileName )
```

Parameters	Description
FileName	The file system path of a HotDocs template, clause library, or component file. In the case of a template or clause library, this method will open the associated component file.

ComponentCollection.OpenForEdit Method

This method opens a component file for editing.

This method was introduced with the release of HotDocs 2006.

Syntax

```
void OpenForEdit ( string FileName )
```

Parameters	Description
FileName	The file name and path of the component file to open.

ComponentCollection.Application Property

[Read-only] This property returns a reference to the *Application* object.

Since there is only one *Application* object on a machine at a time, this property will return a reference to the same object as the *Application* property on any other object in HotDocs, and a reference to the same object as if you created a new *HotDocs.Application* object.

Syntax

```
HotDocs._Application2 Application [ get ]
```

ComponentCollection.Count Property

[Read-only] This property returns the number of *Component* objects in the *ComponentCollection*.

Syntax

```
int Count [ get ]
```

Example

The following Visual C# example displays a list of all variable components in a given component file.

```
public class ExampleCode
{
    static void Main()
    {
        HotDocs.ComponentCollection ccl = new HotDocs.ComponentCollection();
        string msg;

        ccl.Open(@"C:\Documents\HotDocs\Templates\demoempl.cmp");

        for (int i = 1; i < ccl.Count; i++)
        {
            msg = ccl.Item(i).Name + " - ";
            switch (ccl.Item(i).Type)
            {
                case HDVARTYPE.HD_TEXTTYPE:
                    Console.WriteLine(msg + " Text Variable");
                    break;
                case HDVARTYPE.HD_NUMBERTYPE:
                    Console.WriteLine(msg + " Number Variable");
                    break;
                case HDVARTYPE.HD_DATETYPE:
                    Console.WriteLine(msg + " Date Variable");
                    break;
                case HDVARTYPE.HD_TRUEFALSETYPE:
                    Console.WriteLine(msg + " True/False Variable");
                    break;
                case HDVARTYPE.HD_MULTCHOICETYPE:
                    Console.WriteLine(msg + " Multiple Choice Variable");
                    break;
                case HDVARTYPE.HD_COMPUTATIONTYPE:
                    Console.WriteLine(msg + " Computation Variable");
                    break;
            }
        }
        Console.ReadKey();
    }
```

```
}
```

ComponentCollection.FileName Property

[Read-only] Returns the file name of the HotDocs component file. To specify the path for the backing component file, pass the file path and name to *_ComponentCollection.Open()*.

Syntax

```
string FileName [ get ]
```

ComponentCollection.OnlyVariables Property

[Read/Write] This property acts as a filter for the component collection to limit the items to variables or all components. If it is true, the *Count* will include only variables, and if it is false, it will include all variables and components. Likewise, if you use an integer as the index with the *Item* method, the components you can retrieve will be either all components or only variables. (Even with this property set to true, you can refer to any component using the Item method if you specify the component name and type instead of just the index number.) The default value for this property is true.

This property was introduced with the release of HotDocs 2006.

Syntax

```
bool OnlyVariables [ set, get ]
```

ComponentCollection.ReadOnly Property

[Read-only] This property indicates whether the component collection (component file) is read-only or writable.

This property was introduced with the release of HotDocs 2006.

Syntax

```
bool ReadOnly [ get ]
```

HotDocs.ComponentProperties Object

HotDocs.ComponentProperties Object

This object represents a collection of *HotDocs.ComponentProperty* objects.

General Information

ProgID:	HotDocs.ComponentProperties.11.0 HotDocs.ComponentProperties (version-independent)
CLSID:	{8378295C-C9F5-4364-AA28-AB2A4CEDCD87}

The following table shows the name and IID for each interface, as well as the version of HotDocs in which it was introduced. The primary interface and the main public interface exposed by this object is *_ComponentProperties*.

Name	IID	Added in
_ComponentProperties	{FC5369B0-97B8-4822-9D6E-992D124222CF}	Added in HotDocs 2006

Methods

Method	Description
Add	This method adds a new *ComponentProperty* to the *ComponentProperties* collection.
Item	This method returns a specific *ComponentProperty* object from the *ComponentProperties* collection.

Properties

Property	Description
Count	[Read-only] This property returns the number of properties in the *ComponentProperties* collection.

ComponentProperties.Add Method

This method adds a new *ComponentProperty* to the *ComponentProperties* collection.

This method was introduced with the release of HotDocs 2006.

Syntax

```
HotDocs.ComponentProperty Add ( string propertyName, object newVal )
```

Parameters	Description
propertyName	The name of the property to add to the collection. New, custom property names should always be in ALL CAPS. If you create a property name that is not in all capital letters and later try to refer to it that way, you will receive an error because HotDocs converts the name you specify to all capitals when it creates the property.) Existing property names, such as the default properties HotDocs assigns to all components, may be mixed case.
newVal	The value to assign to the new property.

ComponentProperties.Item Method

This method returns a specific *ComponentProperty* object from the *ComponentProperties* collection.

This method was introduced with the release of HotDocs 2006.

Syntax

```
HotDocs.ComponentProperty Item ( object index )
```

Parameters	Description
index	An index to specify which property to retrieve. This can be either a number or a property name.

Return Value

A specific *ComponentProperty* object from the *ComponentProperties* collection.

ComponentProperties.Count Property

[Read-only] This property returns the number of properties in the *ComponentProperties* collection.

This property was introduced with the release of HotDocs 2006.

Syntax

```
int Count [ get ]
```

HotDocs.ComponentProperty Object

HotDocs.ComponentProperty Object

This object represents a specific property of a HotDocs component. For example, it may be the title or prompt of a variable.

General Information

ProgID:	HotDocs.ComponentProperty.11.0 HotDocs.ComponentProperty (version-independent)
CLSID:	{8578D0FF-563A-4A7C-A3B7-08AC6448C3C8}

The following table shows the name and IID for each interface, as well as the version of HotDocs in which it was introduced. The primary interface and the main public interface exposed by this object is *_ComponentProperty*.

Name	IID	Added in
_ComponentProperty	{24048358-5809-46AC-A294-3C1ED9DE10E8}	Added in HotDocs 2006

Properties

Property	Description
Name	[Read-only] This property returns the name of the component property. (A list of the names of HotDocs-defined properties can be found here.)

ReadOnly	[Read-only] This Boolean property indicates whether or not the component property is read-only. For example, checking the value of this property in your code before you change the value of a property can avoid unnecessary problems.
Value	[Read/Write] This property returns the value stored in the component property. You can use this property to read the existing value of a property or change its value as needed. (Before changing the value, however, make sure the property is not read-only.)
VariantType	[Read-only] This property returns an integer corresponding to the component property's variant type.

ComponentProperty.Name Property

[Read-only] This property returns the name of the component property. (A list of the names of HotDocs-defined properties can be found here.)

This property was introduced with the release of HotDocs 2006.

Syntax

```
string Name [ get ]
```

ComponentProperty.ReadOnly Property

[Read-only] This Boolean property indicates whether or not the component property is read-only. For example, checking the value of this property in your code before you change the value of a property can avoid unnecessary problems.

This property was introduced with the release of HotDocs 2006.

Syntax

```
bool ReadOnly [ get ]
```

ComponentProperty.Value Property

[Read/Write] This property returns the value stored in the component property. You can use this property to read the existing value of a property or change its value as needed. (Before changing the value, however, make sure the property is not read-only.)

This property was introduced with the release of HotDocs 2006.

Syntax

```
object Value [ set, get ]
```

ComponentProperty.VariantType Property

[Read-only] This property returns an integer corresponding to the component property's variant type.

This property was introduced with the release of HotDocs 2006.

Syntax

```
int VariantType [ get ]
```

Return Value

VariantType may return one of the following values:

Type	Description
3	Integer
5	R8
8	String
11	Boolean
8204	Array

HotDocs.Dependency Object

HotDocs.Dependency Object

This object represents a single dependency (file) that is required by a template. A dependency may be a template, component, image, or any other kind of file required by the template.

General Information

ProgID:	HotDocs.Dependency.11.0 HotDocs.Dependency (version-independent)
CLSID:	{8DB78A29-8734-4d6a-8BF2-B32596905EE9}

The following table shows the name and IID for each interface, as well as the version of HotDocs in which it was introduced. The primary interface and the main public interface exposed by this object is _Dependency.

Name	IID	Added in
_Dependency	{E7C21BE3-BA72-4d17-A3DD-18AA82AB255E}	Added in HotDocs 10.1

Properties

Property	Description
Dependencies	[Read-only] This property returns a collection of depenencies on which the current dependency depends.
DependencyType	[Read-only] This property returns one of the values from the *DependencyType* Enumeration that indicates what kind of dependency it is.
Target	[Read-only] This property returns the target name of the dependency. For example, if your template contains an INSERT instruction, the target of the dependency as a result of that instruction is the name of the template file being inserted.

Dependency.Dependencies Property

[Read-only] This property returns a collection of depenencies on which the current dependency depends.

Syntax

```
HotDocs.DependencyCollection Dependencies [ get ]
```

Dependency.DependencyType Property

[Read-only] This property returns one of the values from the *DependencyType* Enumeration that indicates what kind of dependency it is.

Syntax

```
HotDocs.DependencyType DependencyType [ get ]
```

Dependency.Target Property

[Read-only] This property returns the target name of the dependency. For example, if your template contains an INSERT instruction, the target of the dependency as a result of that instruction is the name of the template file being inserted.

Syntax

```
string Target [ get ]
```

HotDocs.DependencyCollection Object

HotDocs.DependencyCollection Object

This object represents a collection of *Dependency* objects, which are required by a given template. This collection may be accessed through the *TemplateInfo* object.

General Information

ProgID:	HotDocs.DependencyCollection.11.0 HotDocs.DependencyCollection (version-independent)
CLSID:	{4ADA1473-8871-4c9f-B630-9D3D461991BC}

The following table shows the name and IID for each interface, as well as the version of HotDocs in which it was introduced. The primary interface and the main public interface exposed by this object is *_DependencyCollection*.

Name	IID	Added in
_DependencyCollection	{F846631F-E8E8-4dba-B139-8547D5A3BBEA}	Added in HotDocs 10.1

Methods

Method	Description
GetEnumerator	This method returns an IEnumerator, which you can use to iterate through all dependencies in the collection.
Item	This method returns a specific *Dependency* object from the collection.

Properties

Property	Description
Count	[Read-only] This property returns the number of dependencies in the collection.

DependencyCollection.GetEnumerator Method

This method returns an *IEnumerator*, which you can use to iterate through all dependencies in the collection.

Syntax

```
IEnumerator GetEnumerator ()
```

DependencyCollection.Item Method

This method returns a specific *Dependency* object from the collection.

Syntax

```
HotDocs.Dependency Item ( int index )
```

Parameters	Description
index	The index number of the dependency to retrieve.

DependencyCollection.Count Property

[Read-only] This property returns the number of dependencies in the collection.

Syntax

```
int Count [ get ]
```

HotDocs.Icon Object

HotDocs.Icon Object

This object represents an icon, which you can display next to a custom menu command. You can also overlay an icon on top of the icons displayed next to items in the HotDocs library.

General Information

ProgID:	HotDocs.Icon.11.0 HotDocs.Icon (version-independent)
CLSID:	{D1B6F1CC-C9DA-47a1-B58A-7BF32EB62CE2}

The following table shows the name and IID for each interface, as well as the version of HotDocs in which it was introduced. The primary interface and the main public interface exposed by this object is *_Icon*.

Name	IID	Added in
_Icon	{D1B6F1CB-C9DA-47a1-B58A-7BF32EB62CE2}	HotDocs 6.2 SP1

Methods

Method	Description
LoadBitmap	This method loads a bitmap (.bmp) file.

LoadIcon	This method loads an icon (.ico) file.

Properties

Property	Description
HBITMAP	[Write-only] This property sets the HBITMAP for the icon.
HICON	[Write-only] This property sets the HICON for the icon.
index	[Read/Write] This property sets or returns the index of the icon.
maskColor	[Read/Write] This property sets or returns the mask color for an icon loaded from a bitmap (.bmp) file. It has no effect on icons loaded from an icon (.ico) file because those files have an embedded mask.

This method is only used in conjunction with a HotDocs Plugin.

Icon.LoadBitmap Method

This method loads a bitmap (.bmp) file.

Syntax

```
void LoadBitmap( string FileName )
```

Parameters	Description
FileName	The name of the bitmap file to load.

Icon.LoadIcon Method

This method loads an icon (.ico) file.

Syntax

```
void LoadIcon( string FileName )
```

Parameters	Description

FileName	The name of the icon file to load.

Example

The following Visual C# example adds a menu item to the File menu in the HotDocs library window with your chosen icon. This example can only be used in conjunction with a Plugin:

```
public class ExampleCode
{
    static void Main()
    {
        HotDocs.Application app = new HotDocs.Application();

        //Important: HotDocs.Icon can only be used in conjunction with a
Plugin.
        HotDocs.Icon icon = new HotDocs.Icon();
        icon.LoadIcon(@"C:\images\UserMenuIcon.ico");
        app.AddUserMenuItem2("User Menu Entry #1", HDLIMENU.LI_FILE, 5,
icon);

        Marshal.ReleaseComObject(icon);
        Marshal.ReleaseComObject(app);
    }
}
```

Icon.HBITMAP Property

[Write-only] This property sets the HBITMAP for the icon.

Syntax

```
uint HBITMAP [ set ]
```

Icon.HICON Property

[Write-only] This property sets the HICON for the icon.

Syntax

```
uint HICON [ set ]
```

Icon.index Property

[Read/Write] This property sets or returns the index of the icon.

Syntax

```
int index [ set, get ]
```

Icon.maskColor Property

[Read/Write] This property sets or returns the mask color for an icon loaded from a bitmap (.bmp) file. It has no effect on icons loaded from an icon (.ico) file because those files have an embedded mask.

*maskColor*is treated directly as a Win32 COLORREF value, which has a hexadecimal form of **0x00bbggrr**. (The low-order byte contains the red value, the second byte contains the green value, and the third byte contains the blue value.)

Syntax

```
uint maskColor [ set, get ]
```

HotDocs.Library Object

HotDocs.Library Object

This object represents a HotDocs library (.HDL) file. It allows an integration to iterate through all the items in the library to inspect, add, or delete items.

General Information

ProgID:	HotDocs.Library.11.0 HotDocs.Library (version-independent)
CLSID:	{06B3E595-39E0-4b08-B719-8BCE64A04D70}

The following table shows the name and IID for each interface, as well as the version of HotDocs in which it was introduced. The primary interface and the main public interface exposed by this object is *_Library2*.

Name	IID	Added in
_Library	{FC0AEADD-27D4-460b-8D64-A8CFBC2EFAEC}	Added in HotDocs 6.0
_Library2	{FC0AEADE-27D4-460b-8D64-A8CFBC2EFAEC}	Added in HotDocs 2005

Methods

Method	Description
Close	This method closes an open library file.
New	This method creates a new library file and opens it. If you pass in a *libFileName* that already exists, then that file is opened using the *Open* method. Once you are finished using the library, close it using the *Close* method.
Open	This method opens an existing library file. If the file specified by *fileName* doesn't exist, this method will return an error. (To create and open a new library file, use the *New* method.) When you are finished using the library, you should close it using the *Close* method.
Save	This method saves the library to disk. HotDocs saves the library to the location specified in the *New* or *Open* method call.

Properties

Property	Description
Application	[Read-only] This property returns a reference to the *Application* object.
Description	[Read/Write] This property sets or returns the description for the library. (The description appears in the Properties tab when the top folder of the library is selected.)
MainFolder	[Read-only] This property returns a *LibraryEntity* object representing the root folder of the library.
Redraw	[Write-only] This Boolean property causes the library window to be redrawn (refreshed).
Title	[Read/Write] This property sets or returns the title for the library. (The title appears as the top folder in the library window.)

Library.Close Method

This method closes an open library file.

Syntax

```
void Close ( )
```

Library.New Method

This method creates a new library file and opens it. If you pass in a *libFileName* that already exists, then that file is opened using the *Open* method. Once you are finished using the library, close it using the *Close* method.

HotDocs automatically maintains the state of the library file currently opened in the library interface. HotDocs always has a library open, even if the interface is not visible. This will cause problems for integrators that attempt to make changes to the library file that HotDocs has open using the *Library* object. (Changes that are made through the *_Library* and *_LibraryEntity* interfaces may not be saved correctly.) For this reason, it is recommended that the integration change the currently open library file to some known file using the *_Application.OpenLibrary()* method before attempting to edit a library using these interfaces.

Syntax

```
void New ( string libFileName )
```

Parameters	Description
libFileName	The library file to create.

Library.Open Method

This method opens an existing library file. If the file specified by *fileName* doesn't exist, this method will return an error. (To create and open a new library file, use the *New* method.) When you are finished using the library, you should close it using the *Close* method.

HotDocs automatically maintains the state of the library file currently opened in the library interface. HotDocs always has a library open, even if the interface is not visible. This will cause problems for integrators that attempt to make changes to the library file that HotDocs has open using the *Library* object. (Changes that are made through the *_Library* and *_LibraryEntity* interfaces may not be saved correctly.) For this reason, it is recommended that the integration change the currently open library file to some known file using the *_Application.OpenLibrary()* method before attempting to edit a library using these interfaces.

Syntax

```
void Open ( string libFileName )
```

Parameters	Description
libFileName	The library file to open.

Library.Save Method

This method saves the library to disk. HotDocs saves the library to the location specified in the *New* or *Open* method call.

HotDocs automatically maintains the state of the library file currently opened in the library interface. HotDocs always has a library open, even if the interface is not visible. This will cause problems for integrators that attempt to make changes to the library file that HotDocs has open using the *Library* object. (Changes that are made through the *_Library* and *_LibraryEntity* interfaces may not be saved correctly.) For this reason, it is recommended that the integration change the currently open library file to some known file using the *_Application.OpenLibrary()* method before attempting to edit a library using these interfaces.

Syntax

```
void Save ( )
```

Library.Application Property

[Read-only] This property returns a reference to the *Application* object.

Since there is only one *Application* object on a machine at a time, this property will return a reference to the same object as the *Application* property on any other object in HotDocs, and a reference to the same object as if you created a new *HotDocs.Application* object.

Syntax

```
HotDocs._Application2 Application [ get ]
```

Library.Description Property

[Read/Write] This property sets or returns the description for the library. (The description appears in the Properties tab when the top folder of the library is selected.)

HotDocs automatically maintains the state of the library file currently opened in the library interface. HotDocs always has a library open, even if the interface is not visible. This will cause problems for integrators that attempt to make changes to the library file that HotDocs has open using the *Library* object. (Changes that are made through the *_Library* and *_LibraryEntity* interfaces may not be saved correctly.) For this reason, it is recommended that the integration change the currently open library file to some known file using the *_Application.OpenLibrary()* method before attempting to edit a library using these interfaces.

Syntax

```
string Description [ set, get ]
```

Library.MainFolder Property

[Read-only] This property returns a *LibraryEntity* object representing the root folder of the library.

Syntax

```
HotDocs.LibraryEntity MainFolder [ get ]
```

Library.Redraw Property

[Write-only] This Boolean property causes the library window to be redrawn (refreshed).

This property was introduced with the release of HotDocs 2005.

Syntax

```
bool Redraw [ set ]
```

Library.Title Property

[Read/Write] This property sets or returns the title for the library. (The title appears as the top folder in the library window.)

HotDocs automatically maintains the state of the library file currently opened in the library interface. HotDocs always has a library open, even if the interface is not visible. This will cause problems for integrators that attempt to make changes to the library file that HotDocs has open using the *Library* object. (Changes that are made through the *_Library* and *_LibraryEntity* interfaces may not be saved correctly.) For this reason, it is recommended that the integration change the currently open library file to some known file using the *_Application.OpenLibrary()* method before attempting to edit a library using these interfaces.

Syntax

```
string Title [ set, get ]
```

HotDocs.LibraryEntity Object

HotDocs.LibraryEntity Object

This object represents any valid item in a HotDocs library, including folders, templates, clause libraries, URLs, documents, and so forth.

General Information

ProgID:	HotDocs.LibraryEntity.11.0 HotDocs.LibraryEntity (version-independent)
CLSID:	{4D54CA35-5FB1-4e93-905C-84EE9B1FE69B}

The following table shows the name and IID for each interface, as well as the version of HotDocs in which it was introduced. The primary interface and the main public interface exposed by this object is *_LibraryEntity*.

Name	IID	Added in
_LibraryEntity	{84CEB33D-E30D-4d7e-9ABA-C6D6D9EDCBF3}	Added in HotDocs 2005 SP2
_LibraryEntity2	{84CEB33E-E30D-4d7e-9ABA-C6D6D9EDCBF3}	Added in HotDocs 2005 SP2

Methods

Method	Description
AddFolder	This method adds a new folder to the library. The *LibraryEntity* object on which this method is called must represent a folder in which the new subfolder will be added.
AddTemplate	This method adds a new item to a library folder. The *LibraryEntity* object on which this method is called must represent a folder to which the new item will be added.
Item	This method returns the specified item from the subordinate *_LibraryEntity* collection. The *_LibraryEntity* object on which this method is called must represent a folder, since templates have no subordinate objects in a library.
Remove	This method removes the *LibraryEntity* object from the library.

Properties

Property	Description
Application	[Read-only] This property returns a reference to the *Application* object.
Count	[Read-only] This property returns the number of subordinate objects in the collection. The *LibraryEntity* object on which this property is called should represent a folder, since templates have no subordinate objects in a library.
Description	[Read/Write] This property sets/returns the description for selected library item.
ID	[Read-only] This property returns a numerical identifier for the *LibraryEntity*.
isFolder	[Read-only] If this Boolean property is true, then the *_LibraryEntity* object represents a folder. However, if it is false, the object represents some other library item, such as a template, clause library, URL, and so forth.
OverlayIndex	[Read/Write] This property is a long index to the *Icon* object. (To clear an overlay, set the index to -1.)

Parent	[Read-only] This property returns a *LibraryEntity* object for the parent object. If the current object is the root of the Library, then the return value is NULL.
TemplateFullPath	[Read-only] This property returns the full file system path of the template represented by the *LibraryEntity*. This is different than the *TemplatePath* property, which may contain only a file name or a reference path and file name.
TemplatePath	[Read/Write] This property sets/returns the file system path for the template. The *LibraryEntity* object on which this property is called must represent a non-folder entity because folders have no file system path.
Title	[Read/Write] This property sets/returns the title for the object. The title is the text that appears next to the icon in the list of library items. It also appears in the Properties tab of the library window when the item is selected and viewed.

LibraryEntity.AddFolder Method

This method adds a new folder to the library. The *LibraryEntity* object on which this method is called must represent a folder in which the new subfolder will be added.

Syntax

```
HotDocs.LibraryEntity AddFolder ( string folderTitle, string Description, int atIndex
)
```

Parameters	Description
folderTitle	The title of the new folder.
Description	The description of the new folder
atIndex	[optional] The index (in the collection of subordinate objects) of where to insert the new folder. If this parameter is omitted, or if its value is **-1**, the new folder is added at the end of collection.

Return Value

A *LibraryEntity* object that represents the newly created folder.

LibraryEntity.AddTemplate Method

This method adds a new item to a library folder. The *LibraryEntity* object on which this method is called must represent a folder to which the new item will be added.

Syntax

```
void AddTemplate ( string tplTitle, string filePath, string Description, int atIndex
)
```

Parameters	Description
title	The title of the new library item (e.g., template, Web address, URL, or clause library).
filePath	The file path for the new library item. (For a Web address, this is the URL.)
Description	The description of the new library item.
atIndex	[optional] The position in the folder where the new library item will be inserted. If this parameter is omitted, or if its value is **-1**, the new item is added at the bottom of the folder.

LibraryEntity.Item Method

This method returns the specified item from the subordinate *_LibraryEntity* collection. The *_LibraryEntity* object on which this method is called must represent a folder, since templates have no subordinate objects in a library.

Syntax

HotDocs.LibraryEntity Item (int index)

Parameters	Description
index	The index in the collection of subordinate objects for the object desired. The index must be in the range 0..Count -1.

Return Value

A *LibraryEntity* object that represents the desired object.

LibraryEntity.Remove Method

This method removes the *LibraryEntity* object from the library.

Syntax

```
void Remove ( )
```

LibraryEntity.Application Property

[Read-only] This property returns a reference to the *Application* object.

Since there is only one *Application* object on a machine at a time, this property will return a reference to the same object as the *Application* property on any other object in HotDocs, and a reference to the same object as if you created a new *HotDocs.Application* object.

Syntax

```
HotDocs._Application2 Application [ get ]
```

LibraryEntity.Count Property

[Read-only] This property returns the number of subordinate objects in the collection. The *LibraryEntity* object on which this property is called should represent a folder, since templates have no subordinate objects in a library.

Syntax

```
int Count [ get ]
```

LibraryEntity.Description Property

[Read/Write] This property sets/returns the description for selected library item.

Syntax

```
string Description [ set, get ]
```

LibraryEntity.ID Property

[Read-only] This property returns a numerical identifier for the *LibraryEntity*.

Syntax

```
int ID [ get ]
```

LibraryEntity.IsFolder Property

[Read-only] If this Boolean property is true, then the *_LibraryEntity* object represents a folder. However, if it is false, the object represents some other library item, such as a template, clause library, URL, and so forth.

Syntax

```
bool isFolder [ get ]
```

LibraryEntity.OverlayIndex Property

[Read/Write] This property is a long index to the *Icon* object. (To clear an overlay, set the index to -1.)

Syntax

```
int OverlayIndex [ set, get ]
```

LibraryEntity.Parent Property

[Read-only] This property returns a *LibraryEntity* object for the parent object. If the current object is the root of the Library, then the return value is NULL.

Syntax

```
HotDocs.LibraryEntity Parent [ get ]
```

LibraryEntity.TemplateFullPath Property

[Read-only] This property returns the full file system path of the template represented by the *LibraryEntity*. This is different than the *TemplatePath* property, which may contain only a file name or a reference path and file name.

Syntax

```
string TemplateFullPath [ get ]
```

LibraryEntity.TemplatePath Property

[Read/Write] This property sets/returns the file system path for the template. The *LibraryEntity* object on which this property is called must represent a non-folder entity because folders have no file system path.

Syntax

```
string TemplatePath [ set, get ]
```

LibraryEntity.Title Property

[Read/Write] This property sets/returns the title for the object. The title is the text that appears next to the icon in the list of library items. It also appears in the Properties tab of the library window when the item is selected and viewed.

Syntax

```
string Title [ set, get ]
```

HotDocs.Plugin Object

HotDocs.Plugin Object

This object represents a HotDocs plug-in. It allows an integration to get information about a plug-in, such as its CLSID or Description.

This object was introduced with the release of HotDocs 2005 SP2.

General Information

ProgID:	HotDocs.Plugin.11.0 HotDocs.Plugin (version-independent)
CLSID:	{4D54CA36-5FB1-4e93-905C-84EE9B1FE69B}

The following table shows the name and IID for each interface, as well as the version of HotDocs in which it was introduced. The primary interface and the main public interface exposed by this object is *_Plugin*.

Name	IID	Added in
_Plugin	{84CEB33F-E30D-4d7e-9ABA-C6D6D9EDCBF3}	Added in HotDocs 2005 SP2

Properties

Property	Description
CLSID	[Read/Write] This property sets/returns the class ID (CLSID) of the plug-in. The CLSID should be surrounded by braces ({}). For example, {12345678-1111-2222-AAAA-DDDDDDDDDDDD}.
Description	[Read/Write] This property sets/returns the description of the plug-in. HotDocs does not currently display this description in the user interface, but it may be displayed in future releases.
priorityClass	[Read/Write] This property sets/returns the priority class number (long) of the plug-in. The priority class may be either 100 or 200. In general, plug-ins that change file paths of selected templates should be members of the 100 priority class so they are called first; plug-ins that do not need to change file paths should be members of the 200 priority class.
priorityIndex	[Read/Write] This property sets/returns the priority index of the plug-in within its assigned *priorityClass*. Plug-ins with lower indexes (and lower priority

classes) are called before plug-ins with higher indexes. The *priorityIndex* may be any number 1-99.

Plugin.CLSID Property

[Read/Write] This property sets/returns the class ID (CLSID) of the plug-in. The CLSID should be surrounded by braces ({}). For example, {12345678-1111-2222-AAAA-DDDDDDDDDDDD}.

Syntax

```
string CLSID [ set, get ]
```

Example (Visual C#)

The following Visual C# example displays various properties of each registered plugin:

```
public class ExampleCode
{
    static void Main()
    {
        HotDocs.Application app = new HotDocs.Application();
        HotDocs.PluginsClass plugins = app.Plugins;
        string msg;

        for (int i = 0; i < plugins.Count; i++)
        {
            msg = "CLSID: " + plugins.Item(i).CLSID + "\n\r";
            msg += "Description: " + plugins.Item(i).Description + "\n\r";
            msg += "priorityClass: " + plugins.Item(i).priorityClass +
"\n\r";
            msg += "priorityIndex: " + plugins.Item(i).priorityIndex;
            MessageBox.Show(msg);
        }
    }
}
```

Plugin.Description Property

[Read/Write] This property sets/returns the description of the plug-in. HotDocs does not currently display this description in the user interface, but it may be displayed in future releases.

Syntax

```
string Description [ set, get ]
```

Example (Visual C#)

The following Visual C# example displays various properties of each registered plugin:

```
public class ExampleCode
{
    static void Main()
    {
        HotDocs.Application app = new HotDocs.Application();
        HotDocs.PluginsClass plugins = app.Plugins;
        string msg;

        for (int i = 0; i < plugins.Count; i++)
        {
            msg = "CLSID: " + plugins.Item(i).CLSID + "\n\r";
            msg += "Description: " + plugins.Item(i).Description + "\n\r";
            msg += "priorityClass: " + plugins.Item(i).priorityClass +
"\n\r";
            msg += "priorityIndex: " + plugins.Item(i).priorityIndex;
            MessageBox.Show(msg);
        }
    }
}
```

Plugin.priorityClass Property

[Read/Write] This property sets/returns the priority class number (long) of the plug-in. The priority class may be either 100 or 200. In general, plug-ins that change file paths of selected templates should be members of the 100 priority class so they are called first; plug-ins that do not need to change file paths should be members of the 200 priority class.

Syntax

```
int priorityClass [ set, get ]
```

Example (Visual C#)

The following Visual C# example displays various properties of each registered plugin:

```
public class ExampleCode
{
```

```
    static void Main()
    {
        HotDocs.Application app = new HotDocs.Application();
        HotDocs.PluginsClass plugins = app.Plugins;
        string msg;

        for (int i = 0; i < plugins.Count; i++)
        {
            msg = "CLSID: " + plugins.Item(i).CLSID + "\n\r";
            msg += "Description: " + plugins.Item(i).Description + "\n\r";
            msg += "priorityClass: " + plugins.Item(i).priorityClass +
"\n\r";
            msg += "priorityIndex: " + plugins.Item(i).priorityIndex;
            MessageBox.Show(msg);
        }
    }
}
```

Plugin.priorityIndex Property

[Read/Write] This property sets/returns the priority index of the plug-in within its assigned *priorityClass*. Plug-ins with lower indexes (and lower priority classes) are called before plug-ins with higher indexes. The *priorityIndex* may be any number 1-99.

Syntax

```
int priorityIndex [ set, get ]
```

Example (Visual C#)

The following Visual C# example displays various properties of each registered plugin:

```
public class ExampleCode
{
    static void Main()
    {
        HotDocs.Application app = new HotDocs.Application();
        HotDocs.PluginsClass plugins = app.Plugins;
        string msg;

        for (int i = 0; i < plugins.Count; i++)
        {
            msg = "CLSID: " + plugins.Item(i).CLSID + "\n\r";
            msg += "Description: " + plugins.Item(i).Description + "\n\r";
            msg += "priorityClass: " + plugins.Item(i).priorityClass +
"\n\r";
            msg += "priorityIndex: " + plugins.Item(i).priorityIndex;
            MessageBox.Show(msg);
```

```
        }
    }
}
```

HotDocs.PluginsClass Object

HotDocs.PluginsClass Object

This object represents a collection of registered HotDocs plug-ins. It allows an integration to determine which plug-ins are registered, register new plug-ins, or unregister existing plug-ins.

General Information

ProgID:	HotDocs.PluginsClass.11.0 HotDocs.PluginsClass (version-independent)
CLSID:	{F90E67E0-4489-4546-AD04-26B9AF7DCA87}

The following table shows the name and IID for each interface, as well as the version of HotDocs in which it was introduced. The primary interface and the main public interface exposed by this object is *Plugins*.

Name	IID	Added in
Plugins	{6CEA447A-6CA5-446B-9E78-3EE86B6D44EE}	Added in HotDocs 2005 SP2

Methods

Method	Description
Item	This method returns a *Plugin* object for the registered plug-in specified by the index parameter.
Register	This method registers the plug-in.
Unregister	This method unregisters a plug-in, removing its CLSID from the HotDocs registry. Unregistering a plug-in prevents HotDocs from loading the plug-in during startup.

Properties

Property	Description
Count	[Read-only] This property returns the number of registered plug-ins.

PluginsClass.Item Method

This method returns a *Plugin* object for the registered plug-in specified by the index parameter.

Syntax

```
HotDocs.Plugin Item ( int index )
```

Parameters	Description
index	The index of a specific registered plug-in to be returned.

Return Value

A Plugin object for the registered plug-in specified by the index parameter.

Example (Visual C#)

The following Visual C# example displays various properties of each registered plugin:

```
public class ExampleCode
{
    static void Main()
    {
        HotDocs.Application app = new HotDocs.Application();
        HotDocs.PluginsClass plugins = app.Plugins;
        string msg;

        for (int i = 0; i < plugins.Count; i++)
        {
            msg = "CLSID: " + plugins.Item(i).CLSID + "\n\r";
            msg += "Description: " + plugins.Item(i).Description + "\n\r";
            msg += "priorityClass: " + plugins.Item(i).priorityClass +
"\n\r";
            msg += "priorityIndex: " + plugins.Item(i).priorityIndex;
            MessageBox.Show(msg);
        }
    }
}
```

PluginsClass.Register Method

This method registers the plug-in.

Syntax

```
void Register ( string CLSID, string descriptionStr, int priorityClass, int
priorityIndex )
```

Parameters	Description
CLSID	The class ID for the plug-in (e.g., "{12345678-1111-2222-AAAA-DDDDDDDDDDDD}").
descriptionStr	A description for the plug-in.
priorityClass	Either 100 or 200. Use 100 if it is going to change the file path; otherwise, use 200. This priority is followed when HotDocs is editing or assembling a template; if more than one plug-in is registered, this determines the order.
priorityIndex	Any number 1-99 to indicate the priority.

Example

The following Visual C# example registers a plug-in:

```
[ComRegisterFunction]
public static void RegisterPlugin(Type t)
{
    HotDocs.Application app = new HotDocs.Application();
    app.Plugins.Register("{12345678-1111-2222-AAAA-DDDDDDDDDDDD}", "File
Handler", 100, 1);
    System.Runtime.InteropServices.Marshal.ReleaseComObject(app);
}
```

PluginsClass.Unregister Method

This method unregisters a plug-in, removing its CLSID from the HotDocs registry. Unregistering a plug-in prevents HotDocs from loading the plug-in during startup.

Syntax

```
void Unregister ( string CLSID )
```

Parameters	Description
CLSID	This is the CLSID of the plug-in class to unregister.

Example

The following Visual C# example unregisters a plug-in:

```
[ComUnregisterFunction]
public static void UnRegisterPlugin(Type t)
{
    HotDocs.Application app = new HotDocs.Application();
    app.Plugins.Unregister("{12345678-1111-2222-AAAA-DDDDDDDDDDD}");
    System.Runtime.InteropServices.Marshal.ReleaseComObject(app);
}
```

PluginsClass.Count Property

[Read-only] This property returns the number of registered plug-ins.

Syntax

```
int Count [ get ]
```

Example (Visual C#)

The following Visual C# example displays various properties of each registered plugin:

```
public class ExampleCode
{
    static void Main()
    {
        HotDocs.Application app = new HotDocs.Application();
        HotDocs.PluginsClass plugins = app.Plugins;
        string msg;

        for (int i = 0; i < plugins.Count; i++)
        {
            msg = "CLSID: " + plugins.Item(i).CLSID + "\n\r";
            msg += "Description: " + plugins.Item(i).Description + "\n\r";
            msg += "priorityClass: " + plugins.Item(i).priorityClass +
"\n\r";
            msg += "priorityIndex: " + plugins.Item(i).priorityIndex;
            MessageBox.Show(msg);
        }
    }
}
```

HotDocs.TemplateInfo Object

HotDocs.TemplateInfo Object

This object lets you gather information about the files on which a given HotDocs template depends. For example, most templates consist of a template (e.g., .RTF) file, along with a corresponding component file. If you use shared component files, a template would also depend on that shared component file. Likewise, INSERT or ASSEMBLE instructions in a template also introduce additional dependencies.

General Information

ProgID:	HotDocs.TemplateInfo.11.0 HotDocs.TemplateInfo (version-independent)
CLSID:	{DCDABAA3-E17D-4474-954D-E43B1D91F7EB}

The following table shows the name and IID for each interface, as well as the version of HotDocs in which it was introduced. The primary interface and the main public interface exposed by this object is *_TemplateInfo*.

Name	IID	Added in
_TemplateInfo	{6579F052-B492-426c-B690-04FDF3639B83}	Added in HotDocs 10.1
__TemplateInfoEvents	{932C7EA1-1BA4-4f51-8B2F-822866355714}	Added in HotDocs 10.1

Methods

Method	Description
Close	This method closes the template.
Open	This method opens a template to inspect its file dependencies. When you are finished with the template, call *Close*.

Properties

Property	Description
ComponentCollection	[Read-only] This property returns the *ComponentCollection* for the specified template.
Dependencies	[Read-only] This property returns a *DependencyCollection*, which is a list of all dependencies for the template. Unlike the *RecursiveDepencencies* property, this property shows only the top-level dependencies for the

	template. If you want to get all of the dependencies for each dependency, you must recurse through each dependency in the collection.
EffectiveComponentFile	[Read-only] This property returns the file path of the "effective" component file used by the template. For example, if the template uses a shared component file, this will return the path of the shared file instead of the template's own component file.
PointedToComponentFile	[Read-only] This property returns the file path of the shared component file used by the template. If there is no shared component file, it returns an empty string.
PrimaryComponentFile	[Read-only] This property returns the file name of the template's primary component file. (In the case of a template that uses a shared component file, this property is used to get the path of the template's own component file and not the shared component file path.)
RecursiveDependencies	[Read-only] This property returns a *DependencyCollection*, which is a complete, recursive list of all dependencies for the template and its dependencies. If you just want the top-level list of dependencies without recursion, use *Dependencies* instead.

TemplateInfo.Close Method

This method closes the template.

Syntax

```
void Close ( )
```

TemplateInfo.Open Method

This method opens a template to inspect its file dependencies. When you are finished with the template, call *Close*.

Syntax

```
void Open ( string filePath )
```

Parameters	Description

filePath	The file path of the template file to open.

TemplateInfo.ComponentCollection Property

[Read-only] This property returns the *ComponentCollection* for the specified template.

Syntax

```
HotDocs.ComponentCollection ComponentCollection [ get ]
```

TemplateInfo.Dependencies Property

[Read-only] This property returns a *DependencyCollection*, which is a list of all dependencies for the template. Unlike the *RecursiveDepencencies* property, this property shows only the top-level dependencies for the template. If you want to get all of the dependencies for each dependency, you must recurse through each dependency in the collection.

Take care when recursing through a template's dependencies to keep track of which dependencies have already been inspected. Otherwise, if there is a circular dependency (e.g., Template A depends on Template B, which depends on Template A), your recursive function could enter an endless loop.

Syntax

```
HotDocs.DependencyCollection Dependencies [ get ]
```

TemplateInfo.EffectiveComponentFile Property

[Read-only] This property returns the file path of the "effective" component file used by the template. For example, if the template uses a shared component file, this will return the path of the shared file instead of the template's own component file.

Syntax

```
string EffectiveComponentFile [ get ]
```

TemplateInfo.PointedToComponentFile Property

[Read-only] This property returns the file path of the shared component file used by the template. If there is no shared component file, it returns an empty string.

Syntax

```
string PointedToComponentFile [ get ]
```

TemplateInfo.PrimaryComponentFile Property

[Read-only] This property returns the file name of the template's primary component file. (In the case of a template that uses a shared component file, this property is used to get the path of the template's own component file and not the shared component file path.)

Syntax

```
string PrimaryComponentFile [ get ]
```

TemplateInfo.RecursiveDependencies Property

[Read-only] This property returns a *DependencyCollection*, which is a complete, recursive list of all dependencies for the template and its dependencies. If you just want the top-level list of dependencies without recursion, use *Dependencies* instead.

Syntax

```
HotDocs.DependencyCollection RecursiveDependencies [ get ]
```

HotDocs.VarMap Object

HotDocs.VarMap Object

This object represents a variable mapping object. VarMaps are used to map HotDocs variables in a template to source names in a data source. The *VarMap* object represents three collections:

Collection	Description
HDVariables	A list of HotDocs variables that can be mapped to fields in your data store. This collection is usually populated by copying the list of variables (and their types) from a HotDocs component file.
SourceNames	A list of fields in your data store.
Mapping	A list of mappings between variables and source names (fields) in your data store. Each item in this collection maps one variable name to one source name.

The *VarMap* object also includes some user interface to allow users to assign Mappings between Variables and SourceNames.

The purpose for mapping is to easily create associations between HotDocs variables and data fields in your application. HotDocs does not use these associations internally, but provides this mechanism for the integration to access them. For example, if your application contains a data store with information about customers, and the user wants to assemble a document using the HotDocs integration with data from your data store, you need to know how fields in your data store map to the HotDocs variables.

For example, you could hard code variable names into your application. This would work, as long as the HotDocs templates never changed and every reference field in the data store used that variable name.

An alternate solution would be to use the HotDocs mapping API. Using the mapping API, you can present a dialog with a list of all the fields in the data store and all the HotDocs variables in the template. Then the fields in the data store can be mapped to HotDocs variables. VarMap objects can be stored in HotDocs map (.HMF) files so the mapping only needs to be done once.

Finally, when your application is loading data into an answer collection for assembly, or when HotDocs queries the application for a value, you can look at the map collection to determine which field in your data store to use to provide the data element for the variable.

General Information

ProgID:	HotDocs.VarMap.11.0 HotDocs.VarMap (version-independent)
CLSID:	{4E2481F7-DFB8-4E85-B51F-B12B1D82A377}

The following table shows the name and IID for each interface, as well as the version of HotDocs in which it was introduced. The primary interface and the main public interface exposed by this object is *_VarMap2*.

Name	IID	Added in
_VarMap	{28E330B9-BF78-4849-990E-1B403383E4D4}	Added in HotDocs 6.0
_VarMap2	{28E330BA-BF78-4849-990E-1B403383E4D4}	Added in HotDocs 2005 SP2

Methods

Method	Description
HDVariablesAdd	This method adds a new item to the *HDVariables* collection. However, it does not add a new HotDocs variable to the component file—it simply creates a new item representing a variable.
HDVariablesItem	This method retrieves a variable item from the *HDVariables* collection. A HotDocs variable is identified by a name (varname) and type (vartype). The index parameter is a number which will return the indexth variable from the collection.
MappingAdd	This method adds a new mapping and associates the variable name (varName) with the source name (srcName).
MappingAdd2	This method is similar to *MappingAdd*; it adds a new mapping and associates the variable name (varname) with the source name (srcname). The difference is that it also allows you to specify the write-back mode for the mapping using a value from the HDMappingBackfill enumeration. (See also *MappingAddEx2*.)
MappingAddEx2	This method is similar to *MappingAdd*; it adds a new mapping and associates the variable name (varName) with the source name (srcName). The difference is that it also allows you to specify the write-back mode for the mapping using a value from the HDMappingBackfill enumeration, and variable types for the variable and source from the HDVARTYPE enumeration. (See also *MappingAdd2*.)
MappingItem	This method retrieves the variable name (varname) and source name (srcname) for a mapping in the collection.
MappingItem2	This method, like *MappingItem*, retrieves the variable name (varName) and source name (srcName) for a mapping in the collection. The Index parameter can either be an index number or the name of a HotDocs variable. It also allows you to retrieve variable types for the variable and source, and the current write-back mode for the mapping.
MappingRemove	This method removes a mapping from the Mapping collection.
OpenComponentFile	This method opens a component file and populates the *HDVariables* collection with the components from the component file. (Any existing variables in the *HDVariables* collection are not erased.)

OpenMapFile	This method opens a HotDocs map file, reads in the mappings, and populates the collections. Any existing members of the collections will not be erased.
SaveMapFile	This method saves the mappings to the filename file. If a file already exists, it is overwritten.
ShowUserInterface	This method shows the mapping user interface where users can assign mappings between HotDocs variables and source names.
SourceNamesAdd	This method adds a new source to the *SourceNames* collection. (See also *SourceNamesAdd2*.)
SourceNamesAdd2	This method adds a new source to the *SourceNames* collection. It is similar to *SourceNamesAdd*, but it accepts a third parameter, backfill, which allows you to specify the write-back mode for the source.
SourceNamesItem	This method retrieves the source name and source type from the *SourceNames* collection. The Index parameter is the index number for a position in the *SourceNames* collection. (See also *SourceNamesItem2*.)
SourceNamesItem2	This method retrieves the source name and source type from the *SourceNames* collection. The Index parameter is the index number for a position in the *SourceNames* collection. It is similar to *SourceNamesItem*, but it accepts a fourth parameter, backfill, which returns the write-back mode for the source.
SourceNamesRemove	This method removes the Indexth mapping from the *SourceNames* collection.

Properties

Property	Description
Application	[Read-only] This property returns a reference to the *Application* object.
DefaultBackfill	[Read/Write] This property indicates the default write-back mode for mapped variables.
HDVariablesCount	[Read-only] This property returns the number of items in the Variables collection.
MappingCount	[Read-only] This property returns the number of items in the Mappings collection.
MapTextAndMultipleChoice	[Read/Write] This Boolean property allows variable mapping between Multiple Choice and Text values. (If it is true, Text values may be mapped to Multiple Choice values and vice versa.)
ShowBackfillColumn	[Read/Write] This Boolean property is used to hide (false) or show (true) the "Write Back" column in the mapping interface. This column allows users to select the write-back mode for each variable mapping.
SourceNamesCount	[Read-only] This property returns the number of items in the *SourceNames* collection.

VarMap.HDVariablesAdd Method

This method adds a new item to the *HDVariables* collection. However, it does not add a new HotDocs variable to the component file—it simply creates a new item representing a variable.

Syntax

```
void HDVariablesAdd ( string varName, HotDocs.HDVARTYPE varType )
```

Parameters	Description
varName	The name of the item.
varType	The type of the item. The vartype can be one of the following values, from the HDVARTYPE enumeration: • HD_TEXTTYPE • HD_NUMBERTYPE • HD_DATETYPE • HD_TRUEFALSETYPE • HD_MULTCHOICETYPE

VarMap.HDVariablesItem Method

This method retrieves a variable item from the *HDVariables* collection. A HotDocs variable is identified by a name (varname) and type (vartype). The index parameter is a number which will return the indexth variable from the collection.

Syntax

```
void HDVariablesItem ( int index, out string varName, ref HotDocs.HDVARTYPE varType )
```

Parameters	Description
index	A number representing the position of the desired variable in the collection.
varName	If the variable is found in the collection, then this will be set to the name of the variable. If the variable cannot be found, then this will be an empty ("")

	string.
varType	When the method returns, this parameter will be set to the correct HDVARTYPE for the variable requested. It may be one of the following values: • HD_TEXTTYPE • HD_NUMBERTYPE • HD_DATETYPE • HD_TRUEFALSETYPE • HD_MULTCHOICETYPE

VarMap.MappingAdd Method

This method adds a new mapping and associates the variable name (varName) with the source name (srcName).

Syntax

```
void MappingAdd ( string varName, string srcName )
```

Parameters	Description
varName	The name of the HotDocs variable to map.
srcName	The name of the source for the mapping.

VarMap.MappingAdd2 Method

This method is similar to *MappingAdd*; it adds a new mapping and associates the variable name (varname) with the source name (srcname). The difference is that it also allows you to specify the write-back mode for the mapping using a value from the *HDMappingBackfill* enumeration. (See also *MappingAddEx2*.)

This method was introduced with the release of HotDocs 2005 SP2.

Syntax

```
void MappingAdd2 ( string varName, string srcName, HotDocs.HDMappingBackfill backfill
)
```

Parameters	Description
varName	The name of the HotDocs variable to map.
srcName	The name of the source for the mapping.
backfill	The write-back mode for the mapping.

VarMap.MappingAddEx2 Method

This method is similar to *MappingAdd*; it adds a new mapping and associates the variable name (varName) with the source name (srcName). The difference is that it also allows you to specify the write-back mode for the mapping using a value from the *HDMappingBackfill* enumeration, and variable types for the variable and source from the *HDVARTYPE* enumeration. (See also *MappingAdd2*.)

This method was introduced with the release of HotDocs 2005 SP2.

Syntax

```
void MappingAddEx2 ( string varName, HotDocs.HDVARTYPE varType, string srcName,
HotDocs.HDVARTYPE srcType, HotDocs.HDMappingBackfill backfill )
```

Parameters	Description
varName	The name of the HotDocs variable to map.
varType	The type of the HotDocs variable to map.
srcName	The name of the source for the mapping.
srcType	The type of the source for the mapping.
backfill	The write-back mode for the mapping.

Return Value

An integer indicating the index of the new *Value* within the *Answer* object.

VarMap.MappingItem Method

This method retrieves the variable name (varname) and source name (srcname) for a mapping in the collection.

Syntax

```
void MappingItem ( object index, out string varName, out string srcName )
```

Parameters	Description
Index	The index of the item in the Mapping collection to retrieve. This can either be a number, which will return the mapping at the Index position in the collection, or a string, which is interpreted as the name of a HotDocs variable.
varName	The name of the HotDocs variable in the mapping.
srcName	The name of the source in the mapping.

VarMap.MappingItem2 Method

This method, like *MappingItem*, retrieves the variable name (varName) and source name (srcName) for a mapping in the collection. The Index parameter can either be an index number or the name of a HotDocs variable. It also allows you to retrieve variable types for the variable and source, and the current write-back mode for the mapping.

This method was introduced with the release of HotDocs 2005 SP2.

Syntax

```
void MappingItem2 ( object index, out string varName, out HotDocs.HDVARTYPE varType,
out string srcName, out HotDocs.HDVARTYPE srcType, out HotDocs.HDMappingBackfill
backfill )
```

Parameters	Description
index	The index of the item in the Mapping collection to retrieve. This can either be a number, which will return the mapping at the Index position in the collection, or a string, which is interpreted as the name of a HotDocs variable.
varname	The name of the HotDocs variable in the mapping.
varType	The variable type for the mapping item.
srcname	The name of the source in the mapping.
srcType	The source variable type for the mapping item.
backfill	The write-back mode for the mapping item.

VarMap.MappingRemove Method

This method removes a mapping from the *Mapping* collection.

Syntax

```
void MappingRemove ( int index )
```

Parameters	Description
index	The position of the desired mapping in the collection.

VarMap.OpenComponentFile Method

This method opens a component file and populates the *HDVariables* collection with the components from the component file. (Any existing variables in the *HDVariables* collection are not erased.)

Syntax

```
void OpenComponentFile ( string componentFileName )
```

Parameters	Description
componentFileName	The file path and name of the HotDocs component file (.CMP) to load.

VarMap.OpenMapFile Method

This method opens a HotDocs map file, reads in the mappings, and populates the collections. Any existing members of the collections will not be erased.

Syntax

```
void OpenMapFile(string mapFileName)
```

Parameters	Description
mapFileName	The path and file name of the HotDocs map file (.HMF) to load.

VarMap.SaveMapFile Method

This method saves the mappings to the filename file. If a file already exists, it is overwritten.

Beginning with HotDocs 2009, map files are saved in an XML file format, which cannot be read by earlier versions of HotDocs.

Syntax

```
void SaveMapFile(string mapFileName)
```

Parameters	Description
mapFileName	The file name and path of the HotDocs map file (.HMF).

VarMap.ShowUserInterface Method

This method shows the mapping user interface where users can assign mappings between HotDocs variables and source names.

Syntax

```
bool ShowUserInterface ( bool showImport, bool showLoad, int windowHandle, string
fromString, bool comboBox )
```

Parameters	Description
showImport	[optional,defaultvalue(TRUE)] Show the **Import** button, which allows the user to an load existing HotDocs map (.HMF) file into the variable map.
showLoad	[optional,defaultvalue(TRUE)] Show the **Load** button, which allows the user to load components from a component file into the variable map.
windowHandle	[optional,defaultvalue(0)] The parent window for the variable map dialog. If this is non-null, then the variable map dialog is displayed modal to this parent window.
fromString	[optional] String to be displayed in the Map variables in box in the user interface. If this parameter is not specified, HotDocs will show the file path for the component file that was loaded.

comboBox	[optional,defaultvalue(FALSE)] Indicates whether the controls in the Map to column of the user interface are combo boxes or list boxes. If they are combo boxes, the interface will allow the user to add items to the HDVariables collection from the user interface. If they are list boxes, the user must select an existing item.

Return Value

Indicates if the user clicked the **OK** button or the **Cancel** button. If ok == true, then the user clicked the **OK** button.

Example (Visual C#)

The following Visual C# example displays the HotDocs Variable Mapping dialog box:

```
public class ExampleCode
{
    static void Main()
    {
        HotDocs.VarMap vMap = new HotDocs.VarMap();

        vMap.ShowUserInterface(true, false, vMap.Application.Hwnd, "",
false);
    }
}
```

VarMap.SourceNamesAdd Method

This method adds a new source to the *SourceNames* collection. (See also *SourceNamesAdd2*.)

Syntax

```
void SourceNamesAdd ( string sourceName, HotDocs.HDVARTYPE varType )
```

Parameters	Description
sourceName	The name of the source.
varType	The variable type for the source. In the user interface, the source will be selectable only for variables of this type. This parameter must be one of the following values from the HDVARTYPE enumeration: • HD_TEXTTYPE

- HD_NUMBERTYPE
- HD_DATETYPE
- HD_TRUEFALSETYPE
- HD_MULTCHOICETYPE

VarMap.SourceNamesAdd2 Method

This method adds a new source to the *SourceNames* collection. It is similar to *SourceNamesAdd*, but it accepts a third parameter, backfill, which allows you to specify the write-back mode for the source.

This method was introduced with the release of HotDocs 2005 SP2.

Syntax

```
void SourceNamesAdd2 ( string sourceName, HotDocs.HDVARTYPE varType,
HotDocs.HDMappingBackfill backfill )
```

Parameters	Description
sourcename	The name of the source.
vartype	The variable type for the source. In the user interface, the source will be selectable only for variables of this type. This parameter must be one of the following values from the HDVARTYPE enumeration: • HD_TEXTTYPE • HD_NUMBERTYPE • HD_DATETYPE • HD_TRUEFALSETYPE • HD_MULTCHOICETYPE
backfill	The write-back mode for the source.

VarMap.SourceNamesItem Method

This method retrieves the source name and source type from the *SourceNames* collection. The Index parameter is the index number for a position in the *SourceNames* collection. (See also *SourceNamesItem2*.)

Syntax

```
void SourceNamesItem ( int index, out string sourceName, out HotDocs.HDVARTYPE
varType )
```

Parameters	Description
Index	The index of the item in the SourceNames collection to retrieve.
sourcename	The name of the source.
vartype	The variable type of the source. In the user interface, the source will be selectable only for variables of this type. This parameter must be one of the following values from the HDVARTYPE enumeration: • HD_TEXTTYPE • HD_NUMBERTYPE • HD_DATETYPE • HD_TRUEFALSETYPE • HD_MULTCHOICETYPE

VarMap.SourceNamesItem2 Method

This method retrieves the source name and source type from the *SourceNames* collection. The Index parameter is the index number for a position in the *SourceNames* collection. It is similar to *SourceNamesItem*, but it accepts a fourth parameter, backfill, which returns the write-back mode for the source.

This method was introduced with the release of HotDocs 2005 SP2.

Syntax

```
void SourceNamesItem2 ( int index, out string sourceName, out HotDocs.HDVARTYPE
varType, out HotDocs.HDMappingBackfill backfill )
```

Parameters	Description
Index	The index of the item in the *SourceNames* collection to retrieve.
sourceName	The name of the source.
varType	The variable type of the source. In the user interface, the source will be selectable only for variables of this type. This parameter must be one of the

	following values from the HDVARTYPE enumeration: • HD_TEXTTYPE • HD_NUMBERTYPE • HD_DATETYPE • HD_TRUEFALSETYPE • HD_MULTCHOICETYPE
backfill	The write-back mode for the source.

VarMap.SourceNamesRemove Method

This method removes the Indexth mapping from the *SourceNames* collection.

Syntax

```
void SourceNamesRemove ( int index )
```

Parameters	Description
Index	The position of the desired sourcename in the collection.

VarMap.Application Property

[Read-only] This property returns a reference to the *Application* object.

Since there is only one Application object on a machine at a time, this property will return a reference to the same object as the *Application* property on any other object in HotDocs, and a reference to the same object as if you created a new *HotDocs.Application* object.

Syntax

```
HotDocs._Application2 Application [ get ]
```

VarMap.DefaultBackfill Property

[Read/Write] This property indicates the default write-back mode for mapped variables.

This property was introduced with the release of HotDocs 2005 SP2.

Syntax

```
HotDocs.HDMappingBackfill DefaultBackfill [ set, get ]
```

It can be any value from the *HDMappingBackfill* enumeration:

- Always
- DoNotAllow
- Never
- Prompt

If the value of this property is DoNotAllow, the write-back mode cannot be changed in the mapping interface.

VarMap.HDVariablesCount Property

[Read-only] This property returns the number of items in the Variables collection.

Syntax

```
int HDVariablesCount [ get ]
```

VarMap.MappingCount Property

[Read-only] This property returns the number of items in the Mappings collection.

Syntax

```
int MappingCount [ get ]
```

VarMap.MapTextAndMultipleChoice Property

[Read/Write] This Boolean property allows variable mapping between Multiple Choice and Text values. (If it is true, Text values may be mapped to Multiple Choice values and vice versa.)

This property was introduced with the release of HotDocs 2005 SP2.

Syntax

```
bool MapTextAndMultipleChoice [ set, get ]
```

VarMap.ShowBackfillColumn Property

[Read/Write] This Boolean property is used to hide (false) or show (true) the "Write Back" column in the mapping interface. This column allows users to select the write-back mode for each variable mapping.

This property was introduced with the release of HotDocs 2005 SP2.

Syntax

```
bool ShowBackfillColumn [ set, get ]
```

VarMap.SourceNamesCount Property

[Read-only] This property returns the number of items in the *SourceNames* collection.

Syntax

```
int SourceNamesCount [ get ]
```

Answer Source API

About the HotDocs Answer Source API

What is an answer source integration?

When completing a HotDocs interview to produce a custom document, HotDocs users typically enter their answers by typing them directly at the answer-gathering dialog. Those answers are then merged into the assembled document.

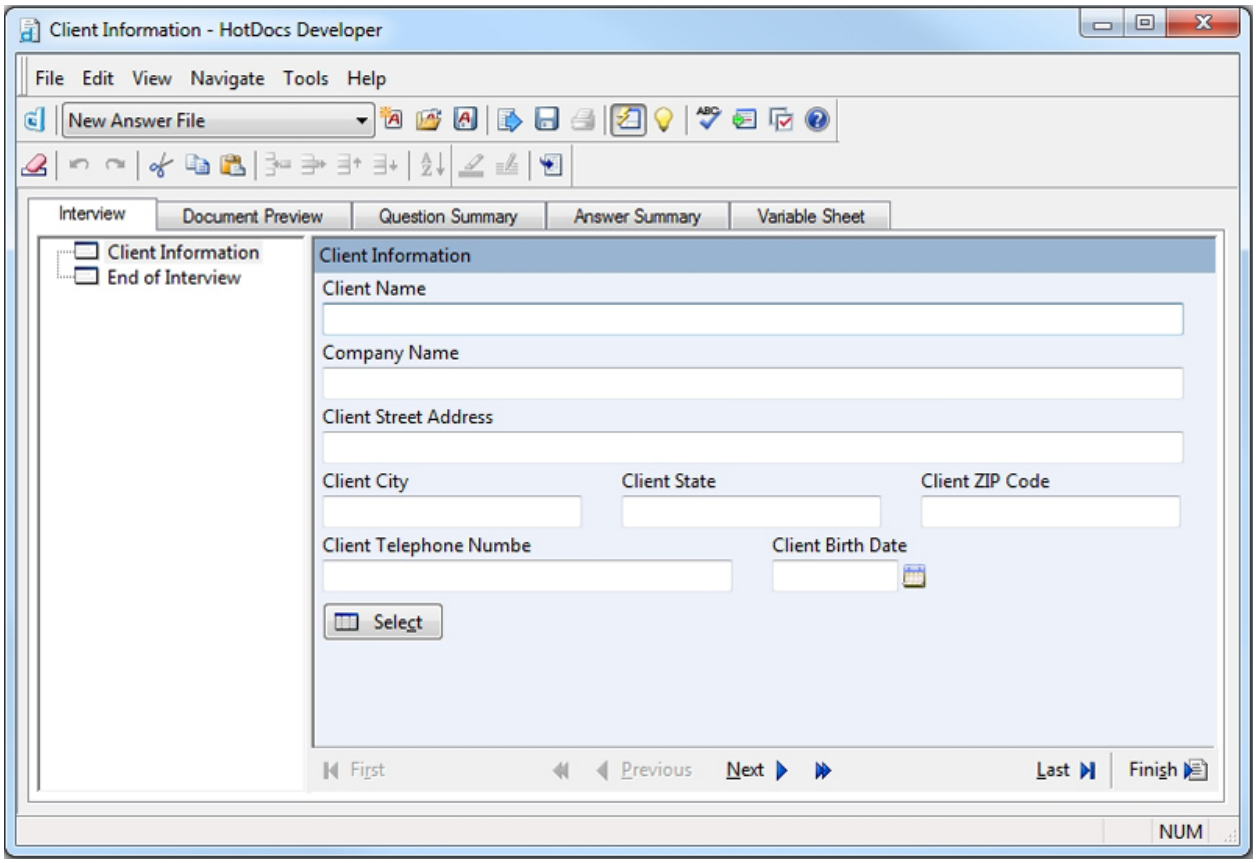
In some situations, however, the data the user needs to enter is stored in a separate application. Rather than requiring the user to manually look up the answers in the third-party application and then re-enter the data in HotDocs, an integration between the two products can be created that allows the user to click a button on a HotDocs dialog and have immediate access to data stored by the third-party.

For example, an integration between HotDocs and Microsoft Outlook allows users to click a button in a HotDocs dialog and have the Contacts list in Outlook open. The user then double-clicks on a given contact and the data about that person is automatically merged into the answer fields of the dialog.

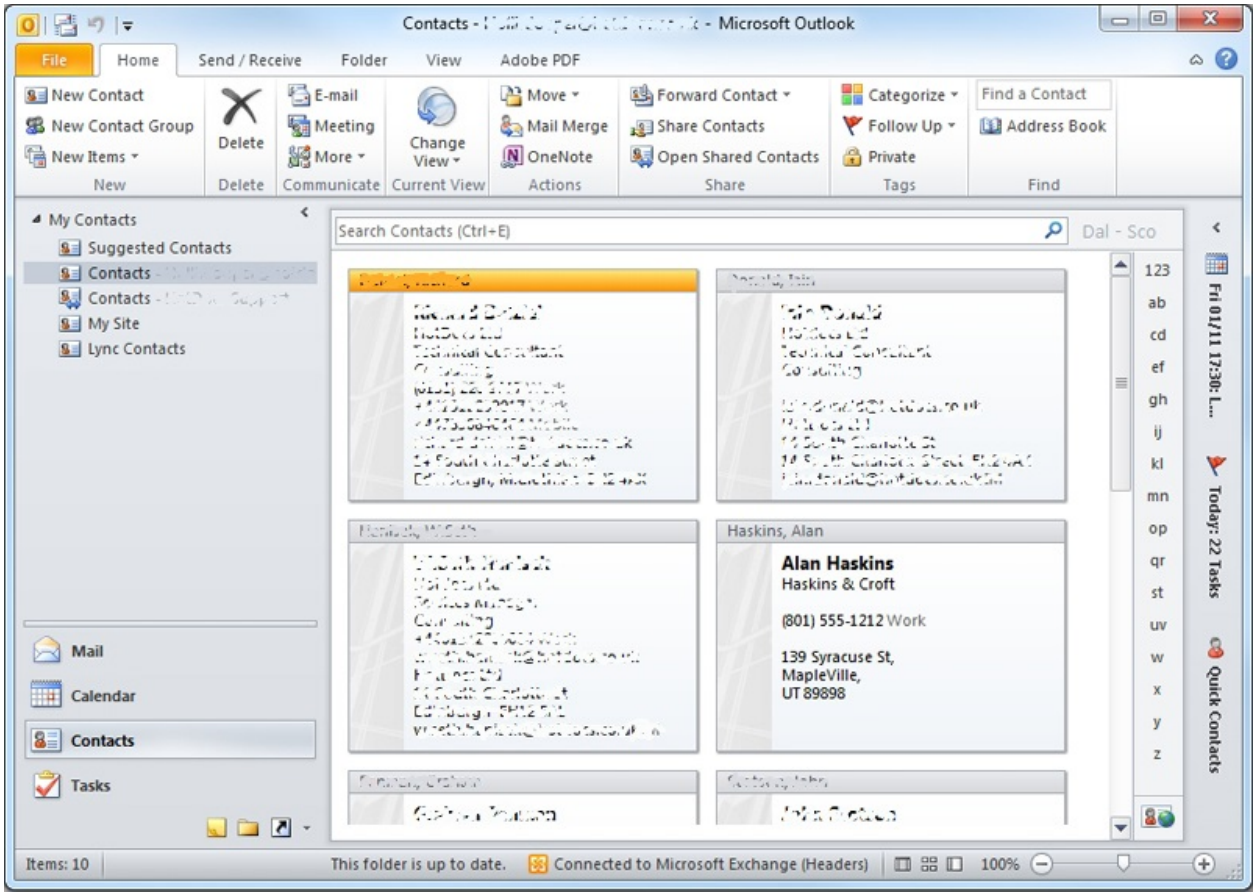
Answer Source Integration Example

To start, it may help to understand how an answer source integration looks and works to both an end user and a developer. In the following example, a DLL has been created that links a dialog to the Microsoft Outlook Contacts list.

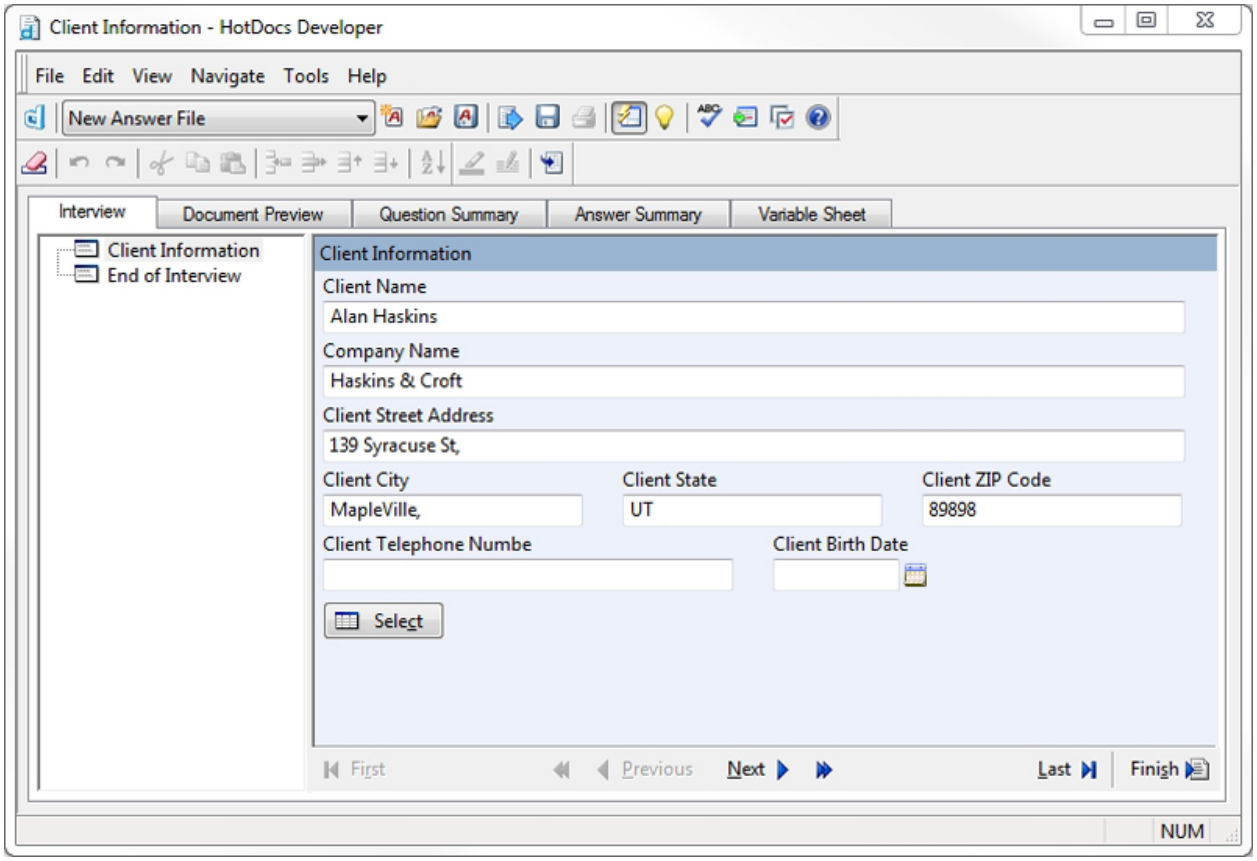
As a user completes a HotDocs interview, dialogs which have answer source integrations appear with a Select button on them.



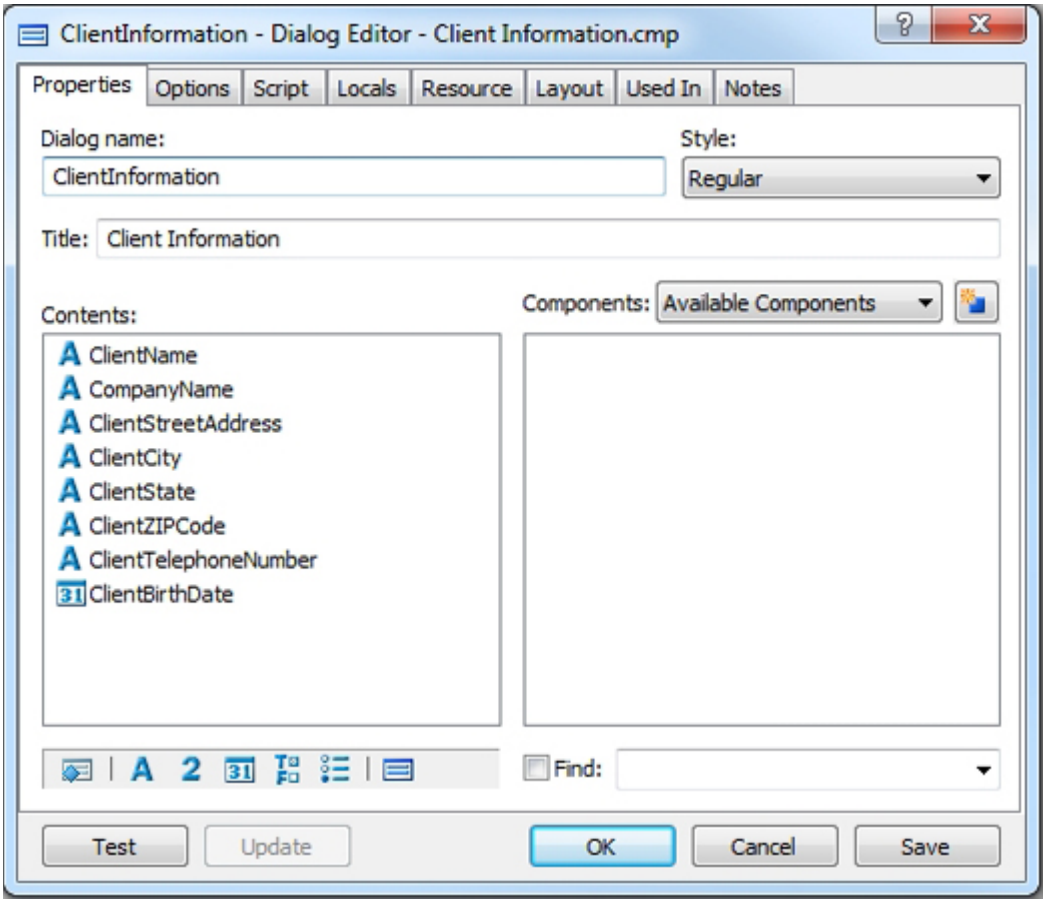
When users click the Select button, the answer source DLL opens Outlook and displays the Contacts list in a modal dialog box.



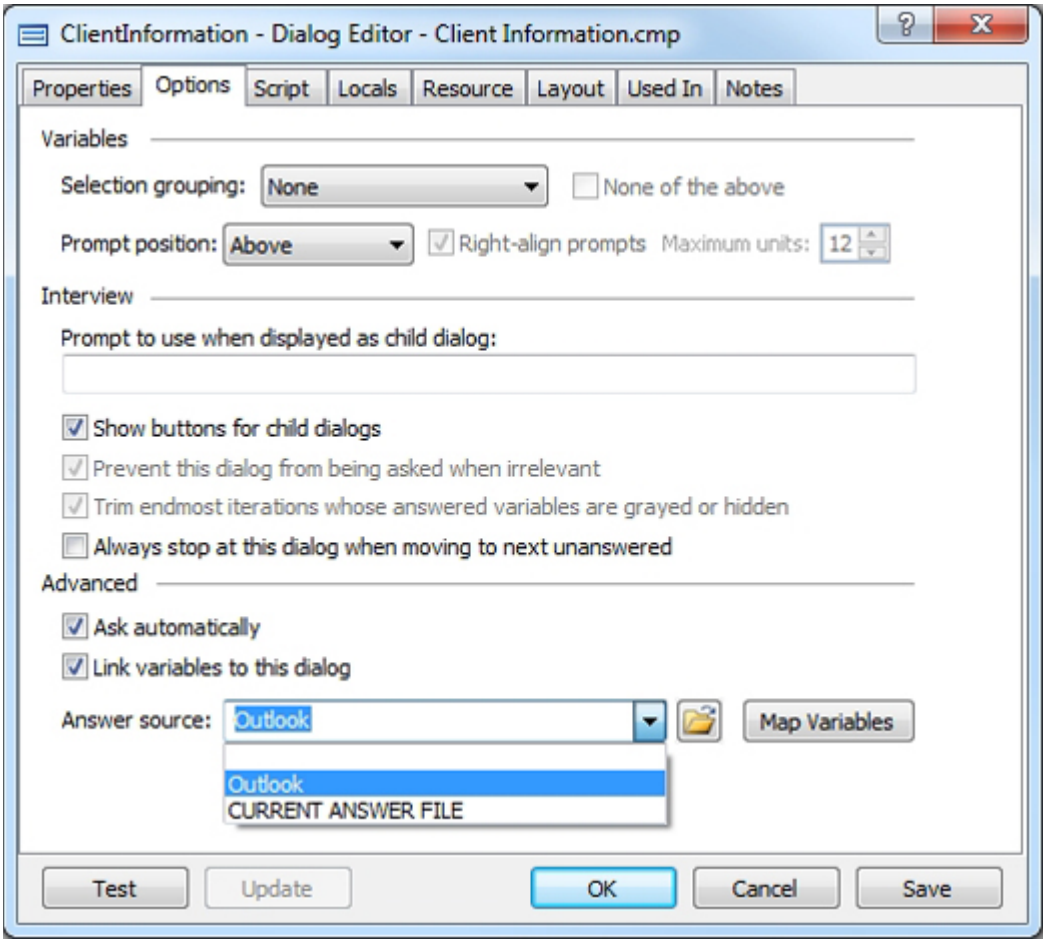
The user then double-clicks a contact in the list and the data associated with the record is merged into the HotDocs dialog.



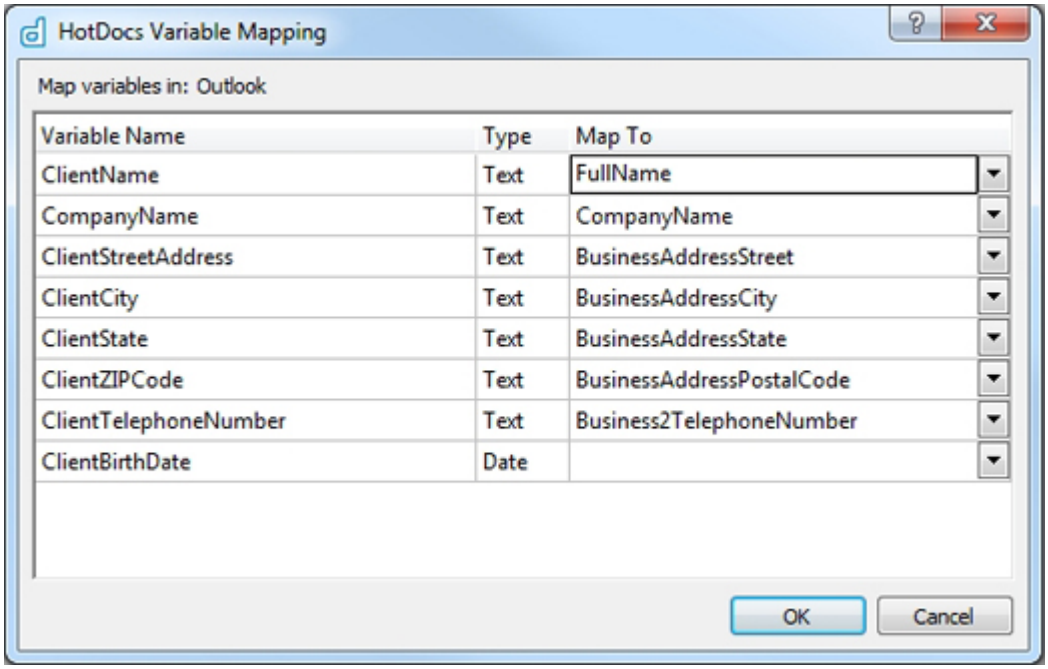
From the perspective of a HotDocs template developer, the steps required to "hook" the answer source integration to a given dialog is simple. In HotDocs Developer, a template developer creates a dialog and adds the variables to the dialog whose values will come from the third-party application.



Once the dialog is created and the variables have been added, the template developer then chooses the answer source DLL at the Options tab of the Dialog Editor.



Once the answer source is selected, the template developer then maps (or associates) variables in the dialog to fields in the answer source (in this case, to fields in the Contacts list).



Once the variables are mapped to the fields, the answer source integration is complete.

How do I create an answer source integration?

To create an answer source integration

1. Create an answer source DLL.

   An answer source is a standard Microsoft Windows 32-bit Dynamic Link Library (DLL) that provides a link from HotDocs to the source application through a defined API that the answer source DLL implements. This API consists of a number of functions that HotDocs uses to communicate with the source application. The answer source DLL can be written using any language which supports the creation of standard Windows DLL files.

   If you use Visual Basic, you must use a third-party tool to create the DLL because Visual Basic can only create an ActiveX DLL.

   The answer source DLL allows HotDocs to query the application for the data it needs to populate the fields of a HotDocs dialog when asked. For this to work, the following must be true:

- HotDocs must be able to determine if the answer source is installed correctly and if it is ready to be called. (See *IsAvailable*.)

- HotDocs must be able to retrieve a list of names and types of all available fields in the answer source. This information will be used when mapping variables to the fields in the answer source. (See *GetFieldNameW* and *GetFieldName*.)
- HotDocs assumes the data source contains records which can be uniquely identified by a 32-bit integer.
- The answer source must display (or cause to be displayed) some user interface whereby users can browse or search the data in the application and select a record. The answer source must then return to HotDocs the 32-bit ID of the selected record. (See *ChooseRecord* and *ChooseMultipleRecords*.)
- After the user chooses a specific record, HotDocs must be able to use the 32-bit record ID to request data for specific fields of that record. (See *GetFieldW* and *GetField*.)

To facilitate the interaction described above, and to allow the DLL to be successfully loaded and used by HotDocs, every answer source DLL must expose and implement the following four functions, or entry points: *IsAvailable*, *GetFieldNameW* (or *GetFieldName*), *ChooseRecord*, and *GetFieldW* (or *GetField*).

Your answer source DLL must export these functions by name, without any C++ name mangling; that is, they must be exported as standard C language functions. Also, they should be declared to use the WINAPI calling convention (**__stdcall** in Visual C++).

2. Register the answer source DLL.

   Once the answer source DLL has been created, it must be saved in the HotDocs program files folder (for example, C:\Program Files\HotDocs). In addition, you must create a registry entry that allows HotDocs to recognize the DLL and display it at the Dialog Editor. This registry entry is also used during an interview when the user attempts to access the answer source.

   To create the registry key, navigate to **HKEY_LOCAL_MACHINE > SOFTWARE > HotDocs > HotDocs > Answer Sources**. (You may need to create the Answer Sources key.) Then create a new string value in this key using an easily identifiable name for the answer source. (This name is what appears in the **Answer source** drop-down list at the **Dialog Editor**.) The value should be the file name and extension of the answer source DLL.

   Once this answer source DLL is created, registered, and saved to the HotDocs program folder (for example, C:\Program Files\HotDocs), HotDocs will call this DLL to obtain the information it needs from the application.

3. Map HotDocs variables to answer source fields.

   In this step, you must associate the answer source with dialogs in your HotDocs templates. This is done by specifying an answer source at the **Options** tab of the **Dialog Editor**. Once the answer source is selected, the template developer must then map (associate) variables in the dialog to fields in the answer source.

   When matching fields in the answer source to variables in HotDocs, it helps to understand that HotDocs recognizes five basic data types: text, number, date, Boolean, and multiple choice. Any

fields in the application that you wish to allow users to map to HotDocs variables should be associated with one of these types. Each type has an integer equivalent that is used in calls to the answer source DLL. These integers, as well as other specific information about each HotDocs data type, is listed here:

HotDocs Data Type	Integer Equivalent	Notes
TEXT	1	Text values in HotDocs are 8-bit strings (Windows-1252 or Latin-1 encoding) up to a maximum length of 15,000 bytes.
NUMBER	2	Number values are passed to HotDocs as strings. Numbers are always interpreted in base 10. Both decimal and integer values are accepted.
DATE	3	Date values are passed to HotDocs as strings. Dates should be formatted in an unambiguous manner when passed to HotDocs, for example, as **YYYY-MM-DD**. HotDocs does not currently support time values.
TRUE/FALSE	5	True/False (Boolean) values are passed into HotDocs as strings. To pass a true value in, the string should contain the word **TRUE**. To pass a false value in, the string should contain the word **FALSE**.
MULTIPLE CHOICE	6	Multiple choice values can be passed into HotDocs two ways—as single-select values or as multiple-select values. Single-select values are passed as simple strings. Multiple-select values are passed as delimited strings, with multiple answers delimited by the vertical bar (|). For example, "Value1|Value2|Value3" would pass three values to HotDocs. In earlier versions, you could use the string <^> to delimit options. While this delimiter still works, you are encouraged to use the vertical bar instead.
UNANSWERED		If the answer source returns an empty string when HotDocs requests a value, HotDocs will not change the current answer. Answer source DLLs can cause HotDocs to clear an existing answer by passing a special string token to HotDocs: <^UNANSWERED^>.

Once the answer source is specified and the variables are mapped, HotDocs displays a **Select** button in the answer-gathering dialog for these questions. When the user clicks **Select**, HotDocs displays a modal dialog box with the available records from your application. When the user selects a record, the information in the record is automatically merged into the corresponding answer files in the dialog.

HotDocs Answer Source API

HotDocs Answer Source API

Every answer source DLL must expose and implement the following four entry points: *IsAvailable*, *GetFieldName*, *ChooseRecord*, and *GetField*. In addition to the required entry points, HotDocs 2005 SP2 introduced eight additional entry points to facilitate multiple record selection and two-way communication between HotDocs and the answer source. (For example, these entry points allow HotDocs to write answers back to the answer source if they are changed during the interview.)

Your answer source DLL must export these functions by name, without any C++ name mangling; that is, they must be exported as standard C language functions. Also, they should be declared to use the WINAPI calling convention (**__stdcall** in Visual C++).

Answer Source DLL Entry Points (Functions)

Function	Description
BeginUpdateBatch	This function is called to signify the beginning of a batch of updates.
ChooseMultipleRecords	This function is called during assembly when the end user clicks the **Select** button on a Spreadsheet dialog. The implementation of this entry point should display (or cause to be displayed) a modal dialog box where the user can browse, search, filter, or otherwise review information made available by the answer source application. When the user chooses one or more records containing the data, the implementation should close the modal dialog box and return the number of records selected.
ChooseRecord	This function is called during assembly when the end user clicks the **Select** button on a Regular or Repeated Series dialog. The implementation of this entry point should display (or cause to be displayed) a modal dialog box where the user can browse, search, filter, or otherwise review information made available by the answer source application. When the user chooses a single record containing the data, the implementation should close the modal dialog box and return a 32-bit integer identifier that HotDocs will use later (in calls to *GetField* or *GetFieldW*).
CloseRecord	This function is called by HotDocs to indicate that access to the specified record is no longer needed.
CommitUpdates	This function is called by HotDocs to commit (save) any changes that have been made to fields in the specified record. For example, if a user selects a record and then changes the answers that came from that record, this function is called to update fields in the original record with changed answers.
EndUpdateBatch	This function is called to signify the ending of a batch of updates. At this point, HotDocs writes answers back to the database as necessary.
GetChosenRecords	This function is called if *ChooseMultipleRecords* returns a value greater than

zero. It fills the recordIDs array with the record identifiers of the selected records in the order in which they are to appear within HotDocs. The records associated with these identifiers are assumed to be in an open state ready for read-only access.

GetField — This function is called shortly after execution returns from the *ChooseRecord* entry point if the Unicode version of this function (*GetFieldW*) is not defined. It is called a number of times to retrieve the data from the individual fields of the selected record in the answer source application.

GetFieldW — This function is called shortly after execution returns from the *ChooseRecord* entry point. It is called a number of times to retrieve the data from the individual fields of the selected record in the answer source application.

GetFieldAccess — This function is called by HotDocs to determine what type of access is allowed for a field in the answer source, and only if the Unicode version of this function (*GetFieldAccessW*) is not defined. Access may be read-only, or read/write, meaning answers can be written back to the original record if they are changed during the interview.

GetFieldAccessW — This function is called by HotDocs to determine what type of access is allowed for a field in the answer source. Access may be read-only, or read/write, meaning answers can be written back to the original record if they are changed during the interview.

GetFieldName — This function is used by HotDocs to enumerate the names and data types of all the available fields in the answer source application if the Unicode version of this function (*GetFieldNameW*) is not defined. *GetFieldName* is called repeatedly when a template developer attempts to map variables in a dialog to fields in the answer source, in order to retrieve the list of possible fields for mapping. HotDocs passes 0 in fieldNum the first time it calls *GetFieldName* (i.e., when it requests the first field name), and increments the number with each successive call. The implementation should set the field name and return the data type of the field.

GetFieldNameW — This function is used by HotDocs to enumerate the names and data types of all the available fields in the answer source application. *GetFieldNameW* is called repeatedly when a template developer attempts to map variables in a dialog to fields in the answer source, in order to retrieve the list of possible fields for mapping. HotDocs passes 0 in fieldNum the first time it calls *GetFieldNameW* (i.e., when it requests the first field name), and increments the number with each successive call. The implementation should set the field name and return the data type of the field.

IsAvailable — This function is called by HotDocs to determine whether the answer source DLL has initialized itself properly, is in communication with the answer source application, and is ready to be used. It is called each time HotDocs loads the answer source DLL.

OpenRecord — This function is called by HotDocs to open the specified record in preparation for accessing it. The answer source should prepare the record

	for the type of access requested by the mode.
SetField	This function is called by HotDocs (if the Unicode version of this function, *SetFieldW*, is not defined) to set a new value for a particular field of a specified record in the answer source.
SetFieldW	This function is called by HotDocs to set a new value for a particular field of a specified record in the answer source.
SupportsBackfill	This function is called by HotDocs to determine if the answer source supports back-filling of modified field values. If **HD_FAILURE (0)** is returned then HotDocs will not call *GetFieldAccess* or *GetFieldAccessW* for each field and will not allow any values modified in HotDocs to be written back to the answer source (back-filled).

BeginUpdateBatch Function

This function is called to signify the beginning of a batch of updates.

This function was introduced with the release of HotDocs 2006. Earlier versions of HotDocs will not call this function.

Syntax

```
void BeginUpdateBatch ( )
```

Example (Visual C++)

The following Visual C++ example displays a MessageBox when the *BeginUpdateBatch* function is called:

```
void WINAPI BeginUpdateBatch()

{

  MessageBox(NULL, "Click OK to continue.", "BeginUpdateBatch", MB_OK);

}
```

ChooseMultipleRecords Function

This function is called during assembly when the end user clicks the **Select** button on a Spreadsheet dialog. The implementation of this entry point should display (or cause to be displayed) a modal dialog box where the user can browse, search, filter, or otherwise review information made available by the answer source application. When the user chooses one or more records containing the data, the implementation should close the modal dialog box and return the number of records selected.

This function was introduced with the release of HotDocs 2005 SP2. Earlier versions of HotDocs will not call this function. On Regular and Repeated Series dialogs, the *ChooseRecord* function is called instead of *ChooseMultipleRecords*.

Syntax

```
long ChooseMultipleRecords ( )
```

Return Value

Return the number of records selected. (If the user cancels the dialog without selecting records, the function should return zero (0).) This value is used to calculate the size of the buffer passed to *GetChosenRecords*.

Example (Visual C++)

The following Visual C++ example displays a MessageBox where the user can choose to either select two (2) or zero (0) records:

```
long WINAPI ChooseMultipleRecords()

{

  long nResult = MessageBox(NULL, "Do you want to choose two records? If you
click no, you will be choosing zero records.", "ChooseMultipleRecords",
MB_YESNO);

  if (nResult == IDYES)

    return 2;

  else

    return 0;

}
```

ChooseRecord Function

This function is called during assembly when the end user clicks the **Select** button on a Regular or Repeated Series dialog. The implementation of this entry point should display (or cause to be displayed) a modal dialog box where the user can browse, search, filter, or otherwise review information made available by the answer source application. When the user chooses a single record containing the data, the implementation should close the modal dialog box and return a 32-bit integer identifier that HotDocs will use later (in calls to *GetField* or *GetFieldW*) to fetch the associated data.

On Spreadsheet dialogs, the *ChooseMultipleRecords* function is called instead of *ChooseRecord*.

Syntax

```
long ChooseRecord ( )
```

Return Value

This function should return the 32-bit record number if a record was selected by the user. If a record was not selected, or if the user cancels the record selection, it should return zero (0).

HotDocs assumes that all record IDs returned by an answer source are greater than or equal to zero. That is, negative record IDs are not allowed.

Example (Visual C++)

The following Visual C++ example displays a MessageBox where the user can choose to either select record #129 or not.

```
long WINAPI ChooseRecord()

{

  long nResult = MessageBox(NULL, "Do you want to select record #129?",
"ChooseRecord", MB_YESNO);

  if (nResult == IDYES)

    return 129;

  else

    return 0;

}
```

CloseRecord Function

This function is called by HotDocs to indicate that access to the specified record is no longer needed.

This function was introduced with the release of HotDocs 2005 SP2. Earlier versions of HotDocs will not call this function.

Syntax

```
long CloseRecord ( long lRecordID )
```

Parameters	Description
lRecordID	The record identifier (key) of the record to close. HotDocs assumes that all record IDs returned by an answer source are greater than or equal to zero. That is, negative record IDs are not allowed.

Return Value

Return **HD_SUCCESS (1)** if successful, and **HD_FAILURE (0)** if otherwise.

CommitUpdates Function

This function is called by HotDocs to commit (save) any changes that have been made to fields in the specified record. For example, if a user selects a record and then changes the answers that came from that record, this function is called to update fields in the original record with changed answers.

This function was introduced with the release of HotDocs 2005 SP2. Earlier versions of HotDocs will not call this function.

Syntax

```
long CommitUpdates ( long lRecordID )
```

Parameters	Description
lRecordID	The record identifier (key) for the record to be committed to storage. The record identified by this ID will be in an opened state ready for read/write access. HotDocs assumes that all record IDs returned by an answer source are greater than or equal to zero. That is, negative record IDs are not allowed.

Return Value

Return **HD_SUCCESS (1)** if successful, and **HD_FAILURE (0)** if otherwise.

EndUpdateBatch Function

This function is called to signify the ending of a batch of updates. At this point, HotDocs writes answers back to the database as necessary.

This function was introduced with the release of HotDocs 2006. Earlier versions of HotDocs will not call this function.

Syntax

```
void EndUpdateBatch ( )
```

GetChosenRecords Function

This function is called if *ChooseMultipleRecords* returns a value greater than zero. It fills the recordIDs array with the record identifiers of the selected records in the order in which they are to appear within HotDocs. The records associated with these identifiers are assumed to be in an open state ready for read-only access.

This function was introduced with the release of HotDocs 2005 SP2. Earlier versions of HotDocs will not call this function.

Syntax

```
long GetChosenRecords ( long* recordIDs )
```

Parameters	Description
recordIDs	An array of long values allocated and passed in by the caller (HotDocs) guaranteed to be **sizeof(long)*NumSelRecs** where *NumSelRecs* is the value returned by the previous call to *ChooseMultipleRecords*. HotDocs assumes that all record IDs returned by an answer source are greater than or equal to zero. That is, negative record IDs are not allowed.

Return Value

Return **HD_SUCCESS (1)** if successful, and **HD_FAILURE (0)** if otherwise.

GetField Function

This function is called shortly after execution returns from the *ChooseRecord* entry point if the Unicode version of this function (*GetFieldW*) is not defined. It is called a number of times to retrieve the data from the individual fields of the selected record in the answer source application.

If you need HotDocs to pass a Unicode string to this function, use *GetFieldW*.

Syntax

```
long GetField ( long key, LPCSTR fieldName, long typeID, LPSTR data, long maxLen )
```

Parameters	Description
key	The record key returned by *ChooseRecord*.
fieldName	The name of the field whose value is requested.
typeID	The variable type of the field whose value is requested. (See HotDocs Data Types.)
data	A buffer to contain the field value, as an 8-bit string with Windows-1252 encoding. Pass the string **<^UNANSWERED^>** to indicate that the field should be set to unanswered in HotDocs. If you pass an empty string, HotDocs will not change the current answer for that variable.
maxLen	The maximum size (in bytes) of the data buffer.

Return Value

Return **HD_SUCCESS (1)** if successful, and **HD_FAILURE (0)** if otherwise.

Example (Visual C++)

The following Visual C++ example returns a field value.

```
long WINAPI GetField( long key, LPCSTR fieldName, long typeID, LPSTR data, long
maxLen )

{

  if (key == 129)

    {
```

```
            if (_stricmp(fieldName, "First Name") == 0)
              {
                strncpy_s(data, maxLen, "Greg", maxLen);
                data[maxLen-1] = '\0';
                return HD_SUCCESS;
              }
            else if (_stricmp(fieldName, "Last Name") == 0)
              {
                strncpy_s(data, maxLen, "Jones", maxLen);
                data[maxLen-1] = '\0';
                return HD_SUCCESS;
              }
          }
        return HD_FAILURE;
      }
```

GetFieldW Function

This function is called shortly after execution returns from the *ChooseRecord* entry point. It is called a number of times to retrieve the data from the individual fields of the selected record in the answer source application.

This function was introduced with the release of HotDocs 2009 to support Unicode strings. Earlier versions of HotDocs will not call this function.

Syntax

```
long GetField ( long key, LPCWSTR fieldName, long typeID, LPCWSTR data, long maxLen )
```

Parameters	Description

key	The record key returned by *ChooseRecord*.
fieldName	The name of the field whose value is requested.
typeID	The variable type of the field whose value is requested. (See HotDocs Data Types.)
data	A buffer to contain the field value, as an 8-bit Unicode string. Pass the string **<^UNANSWERED^>** to indicate that the field should be set to unanswered in HotDocs. If you pass an empty string, HotDocs will not change the current answer for that variable.
maxLen	The maximum size (in bytes) of the data buffer.

Return Value

Return **HD_SUCCESS (1)** if successful, and **HD_FAILURE (0)** if otherwise.

Example (Visual C++)

The following Visual C++ example returns a field value.

```
long WINAPI GetFieldW( long key, LPCWSTR fieldName, long typeID, LPWSTR data,
long maxLen )

{

  if (key == 129)

    {

      if (_wcsicmp(fieldName, L"First Name") == 0)

        {

          wcsncpy_s(data, maxLen, L"Greg", maxLen);

          data[maxLen-1] = '\0';

          return HD_SUCCESS;

        }

      else if (_wcsicmp(fieldName, L"Last Name") == 0)

        {

          wcsncpy_s(data, maxLen, L"Jones", maxLen);

          data[maxLen-1] = '\0';
```

```
            return HD_SUCCESS;
          }
        }
      return HD_FAILURE;
    }
```

GetFieldAccess Function

This function is called by HotDocs to determine what type of access is allowed for a field in the answer source, and only if the Unicode version of this function (*GetFieldAccessW*) is not defined. Access may be read-only, or read/write, meaning answers can be written back to the original record if they are changed during the interview.

This function was introduced with the release of HotDocs 2005 SP2. Earlier versions of HotDocs will not call this function. If you need HotDocs to pass a Unicode string to this function, use *GetFieldAccessW*.

Syntax

```
long GetFieldAccess ( LPCSTR szFieldName )
```

Parameters	Description
szFieldName	The name of the field that is to have its value retrieved.

Return Value

Return **HD_READONLY (0)** if only read access is allowed, or **HD_READWRITE (1)** if both read and write (back-fill) access is allowed.

Example (Visual C++)

The following Visual C++ example returns asks if read/write access is allowed:

```
long WINAPI GetFieldAccess(LPCWSTR szFieldName)
{
  if (MessageBox(NULL, "Is read/write supported?", "GetFieldAccess", MB_YESNO)
== IDYES)
```

```
    return HD_READWRITE;

  else

    return HD_READONLY;

}
```

GetFieldAccessW Function

This function is called by HotDocs to determine what type of access is allowed for a field in the answer source. Access may be read-only, or read/write, meaning answers can be written back to the original record if they are changed during the interview.

This function was introduced with the release of HotDocs 2009 to support Unicode strings. Earlier versions of HotDocs will not call this function.

Syntax

```
long GetFieldAccessW ( LPCWSTR szFieldName )
```

Parameters	Description
szFieldName	The name of the field that is to have its value retrieved.

Return Value

Return **HD_READONLY (0)** if only read access is allowed, or **HD_READWRITE (1)** if both read and write (back-fill) access is allowed.

Example (Visual C++)

The following Visual C++ example returns asks if read/write access is allowed:

```
long WINAPI GetFieldAccessW(LPCWSTR szFieldName)

{

  if (MessageBox(NULL, "Is read/write supported?", "GetFieldAccessW", MB_YESNO)
== IDYES)

    return HD_READWRITE;

  else
```

```
        return HD_READONLY;

    }
```

GetFieldName Function

This function is used by HotDocs to enumerate the names and data types of all the available fields in the answer source application if the Unicode version of this function (*GetFieldNameW*) is not defined. *GetFieldName* is called repeatedly when a template developer attempts to map variables in a dialog to fields in the answer source, in order to retrieve the list of possible fields for mapping. HotDocs passes **0** in fieldNum the first time it calls *GetFieldName* (i.e., when it requests the first field name), and increments the number with each successive call. The implementation should set the field name and return the data type of the field.

If you need HotDocs to pass a Unicode string to this function, use *GetFieldNameW*.

Syntax

```
long GetFieldName ( LPSTR fieldName, long fieldNum )
```

Parameters	Description
fieldName	The buffer for the field name. It can hold up to a maximum of 50 characters plus a NULL terminator.
fieldNum	The number of the requested field name.

Return Value

Return the data type of the field (see HotDocs Data Types for a list of data types), or zero (0) to indicate there are no more fields to enumerate.

Example (Visual C++)

The following Visual C++ example uses *GetFieldName*:

```
    long WINAPI GetFieldName( LPSTR fieldName, long fieldNum )

    {

      long maxLen = 50;

        if (fieldNum == 0)

          {
```

```
        strcpy_s(fieldName, maxLen, "First Name");
        return 1;
      }
    else if (fieldNum == 1)
      {
        strcpy_s(fieldName, maxLen, "Last Name");
        return 1;
      }
    else
      return 0;
}
```

GetFieldNameW Function

This function is used by HotDocs to enumerate the names and data types of all the available fields in the answer source application. *GetFieldNameW* is called repeatedly when a template developer attempts to map variables in a dialog to fields in the answer source, in order to retrieve the list of possible fields for mapping. HotDocs passes 0 in fieldNum the first time it calls *GetFieldNameW* (i.e., when it requests the first field name), and increments the number with each successive call. The implementation should set the field name and return the data type of the field.

This function was introduced with the release of HotDocs 2009 to support Unicode strings. Earlier versions of HotDocs will not call this function.

Syntax

```
long GetFieldNameW ( LPCWSTR fieldName, long fieldNum )
```

Parameters	Description
fieldName	The buffer for the field name. It can hold up to a maximum of 50 characters plus a NULL terminator.
fieldNum	The number of the requested field name.

Return Value

Return the data type of the field (see HotDocs Data Types for a list of data types), or zero (0) to indicate there are no more fields to enumerate.

Example (Visual C++)

The following Visual C++ example uses *GetFieldNameW*:

```
long WINAPI GetFieldNameW( LPWSTR fieldName, long fieldNum )
{
  long maxLen = 50;
  if (fieldNum == 0)
    {
      wcscpy_s(fieldName, maxLen, L"First Name");
      return 1;
    }
  else if (fieldNum == 1)
    {
      wcscpy_s(fieldName, maxLen, L"Last Name");
      return 1;
    }
  else
    return 0;
}
```

IsAvailable Function

This function is called by HotDocs to determine whether the answer source DLL has initialized itself properly, is in communication with the answer source application, and is ready to be used. It is called each time HotDocs loads the answer source DLL.

Syntax

```
long IsAvailable ( )
```

Return Value

Return **HD_SUCCESS (1)** if the answer source was able to initialize itself properly and communicate with the answer source application. Otherwise, return **HD_FAILURE (0)**.

Example (Visual C++)

The following Visual C++ example displays a Message Box to indicate whether or not the answer source is initialized properly. In your application, you would typically query a database or perform some other check to see if it is working properly.

```
long WINAPI IsAvailable()

{

  if (MessageBox(NULL, "Is the answer source available?", "Sample Answer
Source", MB_YESNO) == IDYES)

    return HD_SUCCESS;

  else

    return HD_FAILURE;

}
```

OpenRecord Function

This function is called by HotDocs to open the specified record in preparation for accessing it. The answer source should prepare the record for the type of access requested by the mode.

This function was introduced with the release of HotDocs 2005 SP2. Earlier versions of HotDocs will not call this function.

Syntax

```
long OpenRecord ( long recordID, long mode )
```

Parameters	Description
recordID	The record identifier (key) of the record to open. HotDocs assumes that all record IDs returned by an answer source are greater than or equal to zero. That is, negative record IDs are not allowed.
mode	The type of access requested for the opened record. **HD_READONLY (0)** for read-only access (*GetField* calls) or **HD_READWRITE (1)** for read/write access (*SetField* calls).

Return Value

Return **HD_SUCCESS (1)** if successful, and **HD_FAILURE (0)** if otherwise.

Example (Visual C++)

The following Visual C++ example returns HD_SUCCESS:

```
long WINAPI OpenRecord(long recordID, long mode)
{
  MessageBox(NULL, "Click OK to continue.", "OpenRecord", MB_OK);
  return HD_SUCCESS;
}
```

SetField Function

This function is called by HotDocs (if the Unicode version of this function, *SetFieldW*, is not defined) to set a new value for a particular field of a specified record in the answer source.

This function was introduced with the release of HotDocs 2005 SP2. Earlier versions of HotDocs will not call this function. If you need HotDocs to pass a Unicode string to this function, use *SetFieldW*.

Syntax

```
long SetField ( long recordID, LPCSTR fieldName, long typeID, LPCSTR value )
```

Parameters	Description
recordID	The record identifier (key) for the record whose field is to be set. This recordID will be in an opened state ready for read/write access. HotDocs assumes that all record IDs returned by an answer source are greater than or equal to zero. That is, negative record IDs are not allowed.
fieldName	The name of the field that is to have its value set.
typeID	The variable type of the field whose value is being set.
value	The new value to be stored in the field. This string will be formatted as follows for each variable type: • TEXT: As is, no special formatting. • NUMBER: A valid number containing only digits and optionally a decimal point and negative sign. • DATE: YYYY-MM-DD • TRUE/FALSE: "TRUE" or "FALSE" • MULTIPLE CHOICE: CHOICE1|CHOICE2...|CHOICEn If the value in HotDocs is unanswered, HotDocs will pass the following string: **^UNANSWERED^**. This value should be treated logically like a database NULL value (no value exists).

Return Value

Return **HD_SUCCESS (1)** if successful, and **HD_FAILURE (0)** if otherwise.

Example (Visual C++)

The following Visual C++ example returns HD_SUCCESS:

```
long WINAPI SetField(long recordID, LPCSTR szFieldName, long typeID, LPCSTR
value)

{

  return HD_SUCCESS;

}
```

SetFieldW Function

This function is called by HotDocs to set a new value for a particular field of a specified record in the answer source.

This function was introduced with the release of HotDocs 2009 to support Unicode strings. Earlier versions of HotDocs will not call this function.

Syntax

```
long SetField ( long recordID, LPCWSTR fieldName, long typeID, LPCWSTR value )
```

Parameters	Description
recordID	The record identifier (key) for the record whose field is to be set. This recordID will be in an opened state ready for read/write access. HotDocs assumes that all record IDs returned by an answer source are greater than or equal to zero. That is, negative record IDs are not allowed.
fieldName	The name of the field that is to have its value set.
typeID	The variable type of the field whose value is being set.
value	The new value to be stored in the field. This string will be formatted as follows for each variable type: • TEXT: As is, no special formatting. • NUMBER: A valid number containing only digits and optionally a decimal point and negative sign. • DATE: YYYY-MM-DD • TRUE/FALSE: "TRUE" or "FALSE" • MULTIPLE CHOICE: CHOICE1|CHOICE2...|CHOICEn If the value in HotDocs is unanswered, HotDocs will pass the following string: **^UNANSWERED^**. This value should be treated logically like a database NULL value (no value exists).

Return Value

Return **HD_SUCCESS (1)** if successful, and **HD_FAILURE (0)** if otherwise.

Example (Visual C++)

The following Visual C++ example returns HD_SUCCESS:

```
long WINAPI SetFieldW(long recordID, LPCWSTR szFieldName, long typeID, LPCWSTR
value)

{
```

```
  return HD_SUCCESS;

}
```

SupportsBackfill Function

This function is called by HotDocs to determine if the answer source supports back-filling of modified field values. If **HD_FAILURE (0)** is returned then HotDocs will not call *GetFieldAccess* or *GetFieldAccessW* for each field and will not allow any values modified in HotDocs to be written back to the answer source (back-filled).

This function was introduced with the release of HotDocs 2005 SP2. Earlier versions of HotDocs will not call this function.

Syntax

```
long SupportsBackfill ( )
```

Return Value

Return **HD_SUCCESS (1)** if back-filling is supported, and **HD_FAILURE (0)** if otherwise.

Example (Visual C++)

The following Visual C++ example returns HD_SUCCESS to indicate that the answer source supports back-filling:

```
long WINAPI SupportsBackfill()

{

  return HD_SUCCESS;

}
```

Plug-in API

About the HotDocs Plug-in API

What is a HotDocs plug-in?

A HotDocs plug-in allows you to extend the HotDocs user interface by adding or customizing options available at the HotDocs library window. For example, you can:

- Create a plug-in to add a menu to the HotDocs library window that contains commands specific to your integration.
- Add items to shortcut menus accessed when users right-click items in the library.
- Change the way HotDocs commands like **Assemble** work with specific types of files.
- Add an overlay image to icons displayed for items in the library.

HotDocs plug-ins are usually in-process COM components (DLLs) written in any COM-capable language, although they can also be implemented as out-of-process executable files. Plug-ins that implement the *ILibraryWindowIconProvider* interface, however, must be in-process DLLs.

When a user launches HotDocs, HotDocs checks for registered plug-ins and creates an instance of the COM object for each plug-in it finds. If the object is created successfully, HotDocs adds additional menus and shortcut menus to the library window, registers file name extensions to be handled by the plug-in, or changes the library item icons, as determined by the plug-in.

HotDocs Plug-in Interfaces

You can implement one or more of the following interfaces in your plug-in:

Interface	Description
ILibraryWindowContextMenuExtension	This interface lets you create a plug-in that adds a shortcut (context) menu to the HotDocs library window. When users right-click an item in the library, your plug-in can add commands to the shortcut menu. Specifically, this interface lets you add your own submenu to the existing shortcut menu.
ILibraryWindowFileHandlerExtension	This interface lets you create a plug-in that specifies how HotDocs handles certain types of files when they are edited or assembled from the library window. For example, you can change what happens when a user selects an .RTF template in the library and clicks the Edit button. In this case, the plug-in could first locate the template in a document management system, check it out, and open it in the word processor for editing.

ILibraryWindowIconProvider	This interface lets you create a plug-in that changes the icons displayed next to items in the HotDocs library. Specifically, it allows you to place an "overlay" icon on top of the existing icons. For example, if you want to create a version control plug-in, you could use overlay status icons to indicate which templates are checked in, checked out, or not under version control.
ILibraryWindowMenuExtension	This interface lets you create a plug-in that adds a menu to the library window menu bar. For example, you can add a menu containing commands specific to your integration with HotDocs.
IOutputPlugin	This interface enables you to create an output plug-in. Once you have created the plugin, when you are assembling a document, you can see the plug-in by selecting **File** > **Send Document To...** The plug-in also adds a new End of Interview output option. This interface lets you write your own output plug-in integration with HotDocs.
IPluginPreferences	This interface lets you create preferences to be used with any plug-ins that integrate with HotDocs.

How do I create a HotDocs plug-in?

You can create a HotDocs plug-in to extend, or customize, the HotDocs user interface to meet the needs of your integration. There are several steps to creating and registering a HotDocs plug-in:

1. Create a DLL that implements one or more of the HotDocs plug-in interfaces:

Interface	Description
ILibraryWindowContextMenuExtension	This interface lets you create a plug-in that adds a shortcut (context) menu to the HotDocs library window. When users right-click an item in the library, your plug-in can add commands to the shortcut menu. Specifically, this interface lets you add your own submenu to the existing shortcut menu.
ILibraryWindowFileHandlerExtension	This interface lets you create a plug-in that specifies how HotDocs handles certain types of files when they are edited or assembled from the library window. For example, you can change what happens when a user selects an .RTF template in the library and clicks the Edit button. In this case, the plug-in could first locate the template in a document management system, check it out, and open it in the word processor for editing.
ILibraryWindowIconProvider	This interface lets you create a plug-in that changes the icons

	displayed next to items in the HotDocs library. Specifically, it allows you to place an "overlay" icon on top of the existing icons. For example, if you want to create a version control plug-in, you could use overlay status icons to indicate which templates are checked in, checked out, or not under version control.
ILibraryWindowMenuExtension	This interface lets you create a plug-in that adds a menu to the library window menu bar. For example, you can add a menu containing commands specific to your integration with HotDocs.
IOutputPlugin	This interface enables you to create an output plug-in. Once you have created the plugin, when you are assembling a document, you can see the plug-in by selecting **File** > **Send Document To...** The plug-in also adds a new End of Interview output option. This interface lets you write your own output plug-in integration with HotDocs.
IPluginPreferences	This interface lets you create preferences to be used with any plug-ins that integrate with HotDocs.

2. Register the DLL with the operating system (e.g., RegAsm.exe).
3. Register the DLL with HotDocs by passing its CLSID to the *PluginsClass.Register* method.

Click the following link to learn how to create and register a HotDocs plug-in using your desired programming language:

- Create a HotDocs plug-in using Visual C#

How do I create a HotDocs plug-in using Visual C#?

To create a HotDocs plug-in using Visual C#

1. Create a new Visual C# **Class Library** project.
2. Right click on **References** from the Solution Explorer and **Add Reference**. The Add Reference dialog box appears.
3. From the **COM type** libraries, select the **HotDocs 11 Type** Library. Click **OK**.
4. Add a **`using HotDocs;`** statement to the project.

If you are using the example below you will also need to add `using System.Runtime.InteropServices` statement and `using System.Windows.Forms` statement to the project. If you would like to work from the example code you can copy it into the project at this stage. If chose to copy the code then you will need to create your own GUID and replace the example GUID where ever it

appears in the example, generate property stubs for Cmd1 and Cmd2, and replace the sample icon location with the real location of your chosen icon.

5. In the project, select the interface(s) your plug-in will implement from the HotDocs type library.
6. Right click to implement the interface: adds stub functions for each function of the selected interface.
7. Write the necessary code to implement each function of the selected interface.
8. Register the plug-in DLL with HotDocs by passing its CLSID to the PluginsClass.Register method. For example:

```
 HotDocs.Application app = new HotDocs.Application();
 app.Plugins.Register("{12345678-1111-2222-AAAA-DDDDDDDDDDDD}",
"Sample Menu Plugin", 100, 1);
```

9. Right click on the project in the Solution Explorer and select **Properties**. On the **Build** tab check the box to **Register for COM interop**.
10. Change the version to **Release** and build the solution to produce a COM server DLL.
11. Locate the .dll in your visual studio project files and copy the COM server DLL to the folder where HotDocs is installed.
12. In an elevated command line using RegAsm register the DLL with Windows. For example:

```
C:\>Windows\Microsoft.NET\Framework\v4.0.30319\RegAsm.exe
"C:\Program Files (x86)\HotDocs\WindowMenuExtension.dll" /codebase
```

Example

The following Visual C# example implements the ILibraryWindowMenuExtension interface:

```
[ComVisible(true)]
[Guid("12345678-1111-2222-AAAA-DDDDDDDDDDDD")]
public class WindowMenuExt : HotDocs.ILibraryWindowMenuExtension
{
    public void Command(string libraryPath, LibraryEntity
caretEntry, int commandId)
    {
        if (commandId == cmd1)
            System.Diagnostics.Process.Start("http://www.hotdocs.co
m");
        if (commandId == cmd2)
            MessageBox.Show("Menu sample w/icon selected");
    }

    public void DisplayMenuInitialize(string libraryPath,
LibraryEntity caretEntry)
    {
        MessageBox.Show(caretEntry.Title);
    }

    public void GetMenuEntry(int menuPosition, ref string menuText,
ref Icon Icon, ref bool enabled, ref bool callAgain, int commandId)
```

```
    {
        HotDocs.Icon icon = new Icon();
        icon.LoadIcon(@"C:\images\MenuEntry.ico");

        switch (menuPosition)
        {
            case 0:
                menuText = "HotDocs Website";
                enabled = true;
                callAgain = true;
                cmd1 = commandId;
                break;
            case 1:
                //Add a separator
                menuText = "-";
                enabled = false;
                break;
            case 2:
                menuText = "Menu Sample w/icon";
                Icon = icon;
                enabled = true;
                callAgain = false;
                cmd2 = commandId;
                break;
            default:
                break;
        }
    }

    public void GetMenuTitle(ref string menuTitle)
    {
        menuTitle = "Menu Ext Plugin";
    }

    public void Initialize()
    {
        //Do any one time initialization
    }

    public void LibraryInitialized()
    {
        MessageBox.Show("The library has been initialized");
    }

    [ComRegisterFunction]
    public static void RegisterPlugin(Type t)
    {
        HotDocs.Application app = new HotDocs.Application();
        app.Plugins.Register("{12345678-1111-2222-AAAA-
DDDDDDDDDDDD}", "Sample Menu Plugin", 100, 1);
        System.Runtime.InteropServices.Marshal.ReleaseComObject(app
);
    }

    [ComUnregisterFunction]
    public static void UnRegisterPlugin(Type t)
    {
```

```
            HotDocs.Application app = new HotDocs.Application();
            app.Plugins.Unregister("{12345678-1111-2222-AAAA-
DDDDDDDDDDDD}");
            System.Runtime.InteropServices.Marshal.ReleaseComObject(app
);

        }

    }
```

ILibraryWindowContextMenuExtension Interface

ILibraryWindowContextMenuExtension Interface

This interface lets you create a plug-in that adds a shortcut (context) menu to the HotDocs library window. When users right-click an item in the library, your plug-in can add commands to the shortcut menu. Specifically, this interface lets you add your own submenu to the existing shortcut menu.

When HotDocs starts up, it attempts to load each registered plug-in. If it finds a plug-in that implements the *ILibraryWindowContextMenuExtension* interface, HotDocs calls *ContextInitialize* to load and initialize the plug-in. Then, whenever the user right-clicks a library item, HotDocs calls *ContextGetMenuTitle*, passing it information about which library entries are selected to determine which, if any, shortcut menus to display. After determining which menus to display, HotDocs calls *ContextGetMenuEntry* a number of times to retrieve the entries for the submenu. Finally, if the user selects an entry from your submenu, HotDocs calls *ContextCommand* and passes it the identifier for the command.

Function	Description
ContextCommand	This function is called when a user selects one of the entries in a custom shortcut menu.
ContextGetMenuEntry	This function is called when a user right-clicks an item in the library to display a shortcut menu. HotDocs calls this function repeatedly to get the name of each entry in the custom submenu and stops when the function fails, or when menuText is an empty string ("").
ContextGetMenuTitle	This function is called when a user right-clicks an item in the library to display a shortcut menu. HotDocs passes information about the selected library item(s), which can be used to determine which entries to include in the submenu, or how submenu commands should operate. The plug-in can then tell HotDocs the name of the submenu, where it should appear in the shortcut menu, and whether it should be preceded or followed by a separator bar. Finally, the plug-in can also enable or disable the submenu as needed.
ContextInitialize	This function is called when HotDocs starts up to determine if it should

load the plug-in. If the function fails, HotDocs will not load the plug-in.

ContextLibraryInitialized This function is called when the library is initialized.

Example (Visual C#)

The following Visual C# example implements the HotDocs.*ILibraryWindowContextMenuExtension* interface:

```
[ComVisible(true)]
[Guid("12345678-1111-2222-AAAA-DDDDDDDDDDDD")]
public class WindowMenuExt : HotDocs.ILibraryWindowContextMenuExtension
{
    static bool initialized = false;

    public void ContextCommand(int commandId)
    {
        if (commandId == cmd1)
            MessageBox.Show("Command 1 selected");
        if (commandId == cmd2)
            MessageBox.Show("Command 2 selected");
    }

    public void ContextGetMenuEntry(int menuPosition, ref string menuText,
ref bool enabled, ref bool callAgain, int commandId)
    {
        switch (menuPosition)
        {
            case 0:
                menuText = "Command 1";
                cmd1 = commandId;
                enabled = true;
                break;
            case 1:
                menuText = "Command 2";
                cmd2 = commandId;
                enabled = true;
                break;
            default:
                break;
        }
    }

    public void ContextGetMenuTitle(string libraryPath, LibraryEntity
caretEntry, ref string menuTitle, ref int menuPosition, ref bool enabled, ref
bool separatorBefore, ref bool separatorAfter)
    {
        menuTitle = "My Submenu";
        menuPosition = -1;  //Adds the new menu t the bottom of the context
menu
    }

    public void ContextInitialize()
    {
    }
```

```
    public void ContextLibraryInitialized()
    {
        if (!initialized)
        {
            //Add Code here
            initialized = true;
        }
    }

    [ComRegisterFunction]
    public static void RegisterPlugin(Type t)
    {
        HotDocs.Application app = new HotDocs.Application();
        app.Plugins.Register("{12345678-1111-2222-AAAA-DDDDDDDDDDD}",
"Context Menu", 100, 1);
        System.Runtime.InteropServices.Marshal.ReleaseComObject(app);
    }

    [ComUnregisterFunction]
    public static void UnRegisterPlugin(Type t)
    {
        HotDocs.Application app = new HotDocs.Application();
        app.Plugins.Unregister("{12345678-1111-2222-AAAA-DDDDDDDDDDD}");
        System.Runtime.InteropServices.Marshal.ReleaseComObject(app);
    }
}
```

ContextCommand Function

This function is called when a user selects one of the entries in a custom shortcut menu.

Syntax

```
ContextCommand ( int commandId )
```

Parameter	Description
commandId	Indicates which command was selected, which the function can use to determine which actions to perform.

To find out which library items are selected when this function is called, refer to the *caretEntry* parameter of the *ContextGetMenuTitle* function.

Example (Visual C#)

The following Visual C# example implements the *ContextCommand* function:

```
public void ContextCommand(int commandId)
{
    if (commandId == cmd1)
    {
        //Code for first command
        MessageBox.Show("Command 1 selected");
    }
    if (commandId == cmd2)
    {
        //Code for second command
        MessageBox.Show("Command 2 selected");
    }
}
```

ContextGetMenuEntry Function

This function is called when a user right-clicks an item in the library to display a shortcut menu. HotDocs calls this function repeatedly to get the name of each entry in the custom submenu and stops when the function fails, or when menuText is an empty string ("").

Syntax

```
ContextGetMenuEntry ( int menuPosition , ref string menuText , ref bool enabled , ref bool callAgain , int commandId )
```

Parameter	Description
menuPosition	A counter that tells the plug-in how many times HotDocs has called this function.
menuText	The text for the menu entry. (If menuText is a hyphen (-), HotDocs adds a separator to the menu.)
enabled	Specifies whether the menu entry should be enabled (TRUE) or disabled (FALSE).
callAgain	Indicates whether HotDocs should call this function again to add additional entries to the submenu.
commandId	An identifier HotDocs passes to the *ContextCommand* function to specify which menu entry is called. The plug-in should save this identifier so it knows which command is selected from the submenu.

Example (Visual C#)

The following Visual C# example implements the *ContextGetMenuEntry* function:

```
public void ContextGetMenuEntry(int menuPosition, ref string menuText, ref
bool enabled, ref bool callAgain, int commandId)
{
    switch (menuPosition)
    {
        case 0:
            menuText = "Command 1";
            cmd1 = commandId;
            enabled = true;
            break;
        case 1:
            menuText = "Command 2";
            cmd2 = commandId;
            enabled = true;
            callAgain = false;
            break;
        default:
            break;
    }
}
```

ContextGetMenuTitle Function

This function is called when a user right-clicks an item in the library to display a shortcut menu. HotDocs passes information about the selected library item(s), which can be used to determine which entries to include in the submenu, or how submenu commands should operate. The plug-in can then tell HotDocs the name of the submenu, where it should appear in the shortcut menu, and whether it should be preceded or followed by a separator bar. Finally, the plug-in can also enable or disable the submenu as needed.

Syntax

```
ContextGetMenuTitle ( string libraryPath , HotDocs.LibraryEntity caretEntry , ref
string menuTitle , ref int menuPosition , ref bool enabled , ref bool separatorBefore
, ref bool separatorAfter )
```

Parameter	Description
libraryPath	The file path for the open library.
caretEntry	A *HotDocs.LibraryEntity* object representing the item selected in the library.
menuTitle	Returns the title of the menu to add.
menuPosition	The position in the shortcut menu where the new menu should be placed. A value of -1 represents the bottom of the menu.
enabled	Specifies whether the menu entry should be enabled (TRUE) or disabled

	(FALSE).
separatorBefore	Specifies whether or not a separator bar will be placed in the shortcut menu before the title.
separatorAfter	Specifies whether or not a separator bar will be placed in the shortcut menu after the title.

Example (Visual C#)

The following Visual C# example implements the *ContextGetMenuTitle* function:

```
public void ContextGetMenuTitle(string libraryPath, LibraryEntity caretEntry,
ref string menuTitle, ref int menuPosition, ref bool enabled, ref bool
separatorBefore, ref bool separatorAfter)
{
    menuTitle = "My Submenu";
    menuPosition = -1;
}
```

ContextInitialize Function

HotDocs API must not be used during a plug-in's initialization. Plug-ins must wait until HotDocs is itself fully initialized (e.g. after all the plugins have loaded) before calling its APIs, or it risks leaving HotDocs running (hidden) after the user closes it.

This function is called when HotDocs starts up to determine if it should load the plug-in. If the function fails, HotDocs will not load the plug-in.

Syntax

```
ContextInitialize ( )
```

When HotDocs calls this function, it has not completely loaded its COM DLLs, which means you cannot call HotDocs COM interfaces in this function. Instead, you can call those interfaces in *ContextLibraryInitialized*, which HotDocs calls after they are made available.

Example (Visual C#)

The following Visual C# example implements the *ContextInitialize* function:

```
public void ContextInitialize()
{
}
```

ContextLibraryInitialized Function

This function is called when the library is initialized.

Syntax

```
ContextLibraryInitialized ( )
```

Example (Visual C#)

The following Visual C# example implements the *ContextLibraryInitialized* function and adds a user defined menu item using *AddUserMenuItem2*:

```
public void ContextLibraryInitialized()
{
static void Main()
    {
        HotDocs.Application app = new HotDocs.Application();

        HotDocs.Icon icon = new HotDocs.Icon();
        icon.LoadIcon(@"C:\images\UserMenuIcon.ico");
        app.AddUserMenuItem2("User Menu Entry #1", HDLIMENU.LI_FILE, 5,
icon);

        Marshal.ReleaseComObject(icon);
        Marshal.ReleaseComObject(app);
    }
}
```

ILibraryWindowFileHandlerExtension Interface

ILibraryWindowFileHandlerExtension Interface

This interface lets you create a plug-in that specifies how HotDocs handles certain types of files when they are edited or assembled from the library window. For example, you can change what happens when a user selects an .RTF template in the library and clicks the **Edit** button. In this case, the plug-in could first locate the template in a document management system, check it out, and open it in the word processor for editing.

When HotDocs starts up, it attempts to load each registered plug-in. If it finds a plug-in that implements the *ILibraryWindowFileHandlerExtension* interface, HotDocs calls *Initialize* to load and initialize the plug-in. After initialization, HotDocs calls *RegisterFileType* to find out which file name extensions will be handled by the plug-in. Then, when a user selects an item in the library with one of these registered file name extensions, HotDocs calls the *Assemble* or *Edit* function (depending on whether the user chooses to assemble or edit the selected file) to determine how it should handle the file.

Functions

Function	Description
Assemble	This function is called when a file with one of the registered file name extensions is selected and assembled from the library window. For example, you could use this plug-in to set command-line options for all templates with a given file name extension. Then, when a user assembles a template, the command-line options for that item in the library can be ignored or substituted with different options.
Edit	This function is called when a file with one of the registered file name extensions is selected and edited from the library window. For example, this function could check a template out of a document management system before it is edited.
Initialize	This function is called when HotDocs starts up to determine if it should load the plug-in. If the function fails, HotDocs will not load the plug-in.
LibraryInitialized	This function is called when the library is initialized.
RegisterFileType	This function registers the file name extensions that will be handled by the library window file handler plug-in. It is called repeatedly until it fails or until callAgain is set to FALSE.

Example (Visual C#)

The following Visual C# example implements the HotDocs.*ILibraryWindowFileHandlerExtension* interface.

```
[ComVisible(true)]
[Guid("12345678-1111-2222-AAAA-DDDDDDDDDDDD")]
public class FileHandlerExt : HotDocs.ILibraryWindowFileHandlerExtension
{
    public void Assemble(string FileName, string switches, ref string
alternateFilename, ref string alternateSwitches, ref bool hotdocsProcess)
    {
        if (switches == "")
            alternateSwitches = switches + " /stw";

        hotdocsProcess = true;
    }

    public void Edit(string FileName, ref string alternateFilename, ref bool
hotdocsProcess)
```

```
    {
        //Add code
        MessageBox.Show(FileName + " is prevented from editing.");
        hotdocsProcess = false;
    }

    public void Initialize()
    {
    }

    public void LibraryInitialized()
    {
    }

    public void RegisterFileType(int callcounter, ref string extension, ref
bool supportEdit, ref bool supportAssembly, ref bool callAgain)
    {
        switch (callcounter)
        {
            case 0:
                extension = ".rtf";
                supportEdit = true;
                supportAssembly = true;
                callAgain = true;
                break;
            case 1:
                extension = ".docx";
                supportEdit = true;
                supportAssembly = true;
                callAgain = true;
                break;
            case 2:
                extension = ".hpt";
                supportEdit = false;
                supportAssembly = true;
                callAgain = false;
                break;
            default:
                break;
        }
    }

    [ComRegisterFunction]
    public static void RegisterPlugin(Type t)
    {
        HotDocs.Application app = new HotDocs.Application();
        app.Plugins.Register("{12345678-1111-2222-AAAA-DDDDDDDDDDD}", "File
Handler", 100, 1);
        System.Runtime.InteropServices.Marshal.ReleaseComObject(app);
    }

    [ComUnregisterFunction]
    public static void UnRegisterPlugin(Type t)
    {
        HotDocs.Application app = new HotDocs.Application();
        app.Plugins.Unregister("{12345678-1111-2222-AAAA-DDDDDDDDDDD}");
        System.Runtime.InteropServices.Marshal.ReleaseComObject(app);
```

```
    }
}
```

Assemble Function

This function is called when a file with one of the registered file name extensions is selected and assembled from the library window. For example, you could use this plug-in to set command-line options for all templates with a given file name extension. Then, when a user assembles a template, the command-line options for that item in the library can be ignored or substituted with different options.

This function only checks templates with file types registered using *RegisterFileType*.

Syntax

```
Assemble ( string FileName , string switches , ref string alternateFilename , ref
string alternateSwitches , ref bool hotdocsProcess )
```

Parameter	Description
FileName	Complete path and file name for the selected library item.
switches	Command-line options (switches) for the selected library item.
alternateFilename	Alternate file name for the selected library item. If the fileName is a document identifier, for example, this parameter could return the actual file name of the template file.
alternateSwitches	Alternate command-line options for the selected library item. Using this parameter, the plug-in could be written to ignore any command-line options in a library and only use command-line options specified in this function.
hotdocsProcess	If this parameter is TRUE, HotDocs will assemble the template. If it is FALSE, HotDocs will do nothing.

Example (Visual C#)

The following Visual C# example checks the command-line options of the templates being assembled to see if the /stw switch is included. If not, /stw is added so that all assembled templates are automatically sent to the word processor:

```
public void Assemble(string FileName, string switches, ref string
alternateFilename, ref string alternateSwitches, ref bool hotdocsProcess)
{
    if (switches != "/stw")
    {
        alternateSwitches = switches + " /stw";
```

```
    }

    hotdocsProcess = true;
}
```

Edit Function

This function is called when a file with one of the registered file name extensions is selected and edited from the library window. For example, this function could check a template out of a document management system before it is edited.

Syntax

```
Edit ( string fileName , ref string alternateFilename , ref bool hotdocsProcess )
```

Parameter	Description
fileName	Complete path and file name for the selected library item.
alternateFilename	Alternate file name for the selected library item. If the fileName is a document identifier, for example, this parameter could return the actual file name of the template file.
hotdocsProcess	If this parameter is TRUE, HotDocs will open the template for editing. If it is FALSE, HotDocs will do nothing.

Example (Visual C#)

The following Visual C# example prevents the normal HotDocs editing process for registered file types and informs the user with a message:

```
public void Edit(string FileName, ref string alternateFilename, ref bool
hotdocsProcess)
{
    MessageBox.Show(FileName + " is prevented from editing.")
    hotdocsProcess = false;
}
```

Initialize Function

HotDocs API must not be used during a plug-in's initialization. Plug-ins must wait until HotDocs is itself fully initialized (e.g. after all the plugins have loaded) before calling its APIs, or it risks leaving HotDocs running (hidden) after the user closes it.

This function is called when HotDocs starts up to determine if it should load the plug-in. If the function fails, HotDocs will not load the plug-in.

When HotDocs calls this function, it has not completely loaded its COM DLLs, which means you cannot call HotDocs COM interfaces in this function. Instead, you can call those interfaces in *LibraryInitialized*, which HotDocs calls after they are made available.

Syntax

```
void Initialize( )
```

Example (Visual C#)

The following Visual C# example implements the Initialize function:

```
public void Initialize()
{
}
```

LibraryInitialized Function

This function is called when the library is initialized.

Syntax

```
void LibraryInitialized ( )
```

Example (Visual C#)

The following Visual C# example displays a message box when the library is initialized

```
public void LibraryInitialized()
{
    MessageBox.Show("The library has been initialized");
}
```

RegisterFileType Function

This function registers the file name extensions that will be handled by the library window file handler plug-in. It is called repeatedly until it fails or until callAgain is set to FALSE.

Syntax

```
RegisterFileType ( int callcounter , ref string extension , ref bool supportEdit ,
ref bool supportAssembly , ref bool callAgain )
```

Parameter	Description
callcounter	A 0-based counter that tells the plug-in how many times HotDocs has called this function.
extension	File name extension to register (with or without a preceding period).
supportEdit	Specifies whether the plug-in will handle the **Edit** command for the extension file type.
supportAssembly	Specifies whether the plug-in will handle the **Assemble** command for the extension file type.
callAgain	Indicates whether HotDocs should call this function again.

Example (Visual C#)

The following Visual C# example registers several file types to be handled by the file handler plug-in. .RTF and .DOCX templates will have edit and assembly support, but the third file name extension, .HPT, will only have support for assembly. HotDocs calls this function repeatedly, incrementing callcounter each time, until callAgain is FALSE:

```
public void RegisterFileType(int callcounter, ref string extension, ref bool
supportEdit, ref bool supportAssembly, ref bool callAgain)
{
    switch (callcounter)
    {
        case 0:
            extension = ".rtf";
            supportEdit = true;
            supportAssembly = true;
            callAgain = true;
            break;
        case 1:
            extension = ".docx";
            supportEdit = true;
            supportAssembly = true;
            callAgain = true;
            break;
        case 2:
            extension = ".hpt";
            supportEdit = false;
```

```
            supportAssembly = true;
            callAgain = false;
            break;
        default:
            break;
    }
}
```

ILibraryWindowIconProvider Interface

ILibraryWindowIconProvider Interface

This interface lets you create a plug-in that changes the icons displayed next to items in the HotDocs library. Specifically, it allows you to place an "overlay" icon on top of the existing icons. For example, if you want to create a version control plug-in, you could use overlay status icons to indicate which templates are checked in, checked out, or not under version control.

When HotDocs starts up, it attempts to load each registered plug-in. If it finds a plug-in that implements the *ILibraryWindowIconProvider* interface, HotDocs calls:

1. *Initialize* to load and initialize the plug-in. (If Initialize returns a failure code, HotDocs will stop calling plug-in functions.)
2. *LibraryInitialized* to complete initialization of the plug-in.
3. *UpdateLibraryEntry* to update the overlay icons for each item in the library as determined by the plug-in.

This interface is very complicated and difficult to implement; it is only recommended for the most advanced integrators.

Plug-ins that implement the *ILibraryWindowIconProvider* interface must be in-process DLLs.

Functions

Function	Description
Initialize	This function is called when HotDocs starts up to determine if it should load the plug-in. If the function fails, HotDocs will not load the plug-in.
LibraryInitialized	This function is called when the library is initialized.
UpdateLibraryEntry	This function is called each time HotDocs updates the list of items in the library window. For example, when an item is added or removed from the library, HotDocs updates the list of items and calls *UpdateLibraryEntry* for each item in the library.

Example (Visual C#)

The following Visual C# example implements the ILibraryWindowIconProvider interface and adds an overlay icon to each RTF template in the library.

```
[ComVisible(true)]
[Guid("12345678-1111-2222-AAAA-DDDDDDDDDDDD")]
public class IconProvider : HotDocs.ILibraryWindowIconProvider
{
    public void Initialize()
    {
    }

    public void LibraryInitialized()
    {
        MessageBox.Show("The library has been initialized");
    }

    public void UpdateLibraryEntry(LibraryEntity pItem, bool bMultithreaded)
    {
        HotDocs.Icon icon = new HotDocs.Icon();
        icon.LoadIcon(@"C:\images\hptIcon.ico");
        HotDocs._LibraryEntity2 pItem2 = (HotDocs._LibraryEntity2)pItem;

        if (System.IO.Path.GetExtension(pItem2.TemplateFullPath.ToLower()) ==
".hpt")
        {
            pItem2.OverlayIndex = 1;
        }
        else
            pItem2.OverlayIndex = -1;

        System.Runtime.InteropServices.Marshal.ReleaseComObject(icon);
    }

    [ComRegisterFunction]
    public static void RegisterPlugin(Type t)
    {
        HotDocs.Application app = new HotDocs.Application();
        app.Plugins.Register("{12345678-1111-2222-AAAA-DDDDDDDDDDDD}", "Icon
Provider", 100, 1);
        System.Runtime.InteropServices.Marshal.ReleaseComObject(app);
    }

    [ComUnregisterFunction]
    public static void UnRegisterPlugin(Type t)
    {
        HotDocs.Application app = new HotDocs.Application();
        app.Plugins.Unregister("{12345678-1111-2222-AAAA-DDDDDDDDDDDD}");
        System.Runtime.InteropServices.Marshal.ReleaseComObject(app);
    }
}
```

Initialize Function

HotDocs API must not be used during a plug-in's initialization. Plug-ins must wait until HotDocs is itself fully initialized (e.g. after all the plugins have loaded) before calling its APIs, or it risks leaving HotDocs running (hidden) after the user closes it.

This function is called when HotDocs starts up to determine if it should load the plug-in. If the function fails, HotDocs will not load the plug-in.

When HotDocs calls this function, it has not completely loaded its COM DLLs, which means you cannot call HotDocs COM interfaces in this function. Instead, you can call those interfaces in *LibraryInitialized*, which HotDocs calls after they are made available.

Syntax

```
void Initialize( )
```

Example (Visual C#)

```
public void Initialize()
{
}
```

LibraryInitialized Function

This function is called when the library is initialized.

Syntax

```
void LibraryInitialized ( )
```

Example (Visual C#)

The following Visual C# example displays a message box when the library is initialized.

```
public void LibraryInitialized()
{
    MessageBox.Show("The library has been initialized");
}
```

UpdateLibraryEntry Function

This function is called each time HotDocs updates the list of items in the library window. For example, when an item is added or removed from the library, HotDocs updates the list of items and calls *UpdateLibraryEntry* for each item in the library.

Syntax

```
UpdateLibraryEntry ( LibraryEntity pItem , bool bMultithreaded )
```

Parameters	Description
pItem	The library item to update.
bMultithreaded	Indicates whether the updates should occur on separate threads or consecutively on the same thread.

Example (Visual C#)

The following Visual C# example adds an overlay icon to each HPT template in the library.

```
public void UpdateLibraryEntry(LibraryEntity pItem, bool bMultithreaded)
{
    HotDocs.Icon icon = new HotDocs.Icon();
    icon.LoadIcon(@"C:\images\hptIcon.ico");
    HotDocs._LibraryEntity2 pItem2 = (HotDocs._LibraryEntity2)pItem;

    if (System.IO.Path.GetExtension(pItem2.TemplateFullPath.ToLower()) ==
".hpt")
    {
        pItem2.OverlayIndex = 1;
    }
    else
        pItem2.OverlayIndex = -1;

    System.Runtime.InteropServices.Marshal.ReleaseComObject(icon);
}
```

ILibraryWindowMenuExtension Interface

ILibraryWindowMenuExtension Interface

This interface lets you create a plug-in that adds a menu to the library window menu bar. For example, you can add a menu containing commands specific to your integration with HotDocs.

When HotDocs starts up, it attempts to load each registered plug-in. If it finds a plug-in that implements the *ILibraryWindowMenuExtension* interface, HotDocs calls:

1. *Initialize* to load and initialize the plug-in. (If *Initialize* returns a failure code, HotDocs will stop calling plug-in functions.)
2. *LibraryInitialized* to complete initialization of the plug-in.
3. *GetMenuTitle* to retrieve the menu title.

When a user selects your plug-in's menu, HotDocs then calls:

1. *DisplayMenuInitialize* to perform any initialization your menu requires.
2. *GetMenuEntry* as many times as needed to retrieve the name of each command in the menu.

Finally, when a user selects an entry from your plug-in's menu, HotDocs calls *Command* and passes it information about which library entries are selected (if any).

Functions

Function	Description
Command	This function is called when a user selects one of the entries in your plug-in's library window menu. If the user has a library entry selected when this function is called, HotDocs passes information about the selected entry to this function so your plug-in can perform an action based on the selected entry as needed.
DisplayMenuInitialize	This function initializes your plug-in's library window menu. HotDocs calls this function each time the user accesses your plug-in's menu, which allows it to behave differently depending on which library is currently open or which items are currently selected in the library.
GetMenuEntry	This function retrieves the commands that will appear on your plug-in's library window menu. HotDocs calls it repeatedly until it fails, or until menuText is an empty string ("").
GetMenuTitle	This function retrieves the title for your plug-in's library window menu. If it fails, or if the text returned is an empty string (""), HotDocs will not display the menu.
Initialize	This function is called when HotDocs starts up to determine if it should load the plug-in. If the function fails, HotDocs will not load the plug-in.
LibraryInitialized	This function is called when the library is initialized.

Example (Visual C#)

The following Visual C# example implements the ILibraryWindowMenuExtension interface to create a custom library window menu bar plug-in.

```
[ComVisible(true)]
[Guid("12345678-1111-2222-AAAA-DDDDDDDDDDDD")]
public class WindowMenuExt : HotDocs.ILibraryWindowMenuExtension
{
    public void Command(string libraryPath, LibraryEntity caretEntry, int
commandId)
    {
        if (commandId == cmd1)
            System.Diagnostics.Process.Start("http://www.hotdocs.com");
        if (commandId == cmd2)
            MessageBox.Show("Menu sample w/icon selected");
    }

    public void DisplayMenuInitialize(string libraryPath, LibraryEntity
caretEntry)
    {
        MessageBox.Show(caretEntry.Title);
    }

    public void GetMenuEntry(int menuPosition, ref string menuText, ref Icon
Icon, ref bool enabled, ref bool callAgain, int commandId)
    {
        HotDocs.Icon icon = new Icon();
        icon.LoadIcon(@"C:\images\MenuEntry.ico");

        switch (menuPosition)
        {
            case 0:
                menuText = "HotDocs Website";
                enabled = true;
                callAgain = true;
                cmd1 = commandId;
                break;
            case 1:
                //Add a separator
                menuText = "-";
                enabled = false;
                break;
            case 2:
                menuText = "Menu Sample w/icon";
                Icon = icon;
                enabled = true;
                callAgain = false;
                cmd2 = commandId;
                break;
            default:
                break;
        }
    }

    public void GetMenuTitle(ref string menuTitle)
    {
```

```
        menuTitle = "Menu Ext Plugin";
    }

    public void Initialize()
    {
        //Do any one time initialization
    }

    public void LibraryInitialized()
    {
        MessageBox.Show("The library has been initialized");
    }

    [ComRegisterFunction]
    public static void RegisterPlugin(Type t)
    {
        HotDocs.Application app = new HotDocs.Application();
        app.Plugins.Register("{12345678-1111-2222-AAAA-DDDDDDDDDDD}",
"Sample Menu Plugin", 100, 1);
        System.Runtime.InteropServices.Marshal.ReleaseComObject(app);
    }

    [ComUnregisterFunction]
    public static void UnRegisterPlugin(Type t)
    {
        HotDocs.Application app = new HotDocs.Application();
        app.Plugins.Unregister("{12345678-1111-2222-AAAA-DDDDDDDDDDD}");
        System.Runtime.InteropServices.Marshal.ReleaseComObject(app);
    }
}
```

Command Function

This function is called when a user selects one of the entries in your plug-in's library window menu. If the user has a library entry selected when this function is called, HotDocs passes information about the selected entry to this function so your plug-in can perform an action based on the selected entry as needed.

Syntax

```
void Command(string libraryPath, HotDocs.LibraryEntity caretEntry, int commandId)
```

Parameters	Description
libraryPath	The file path for the open library.
caretEntry	The currently selected library entry.
commandId	An identifier for the command, which your plug-in can use to determine

which command was selected. (This is the same as *commandId* in *GetMenuEntry*.)

Example (Visual C#)

The following Visual C# example implements the *Command* function.

```
public void Command(string libraryPath, LibraryEntity caretEntry, int
commandId)
{
    if (commandId == cmd1)
        System.Diagnostics.Process.Start("http://www.hotdocs.com");
    if (commandId == cmd2)
        MessageBox.Show("Menu sample w/icon selected");
}
```

DisplayMenuInitialize Function

This function initializes your plug-in's library window menu. HotDocs calls this function each time the user accesses your plug-in's menu, which allows it to behave differently depending on which library is currently open or which items are currently selected in the library.

Syntax

```
void DisplayMenuInitialize( string libraryPath, HotDocs.LibraryEntity caretEntry)
```

Parameters	Description
libraryPath	The file path for the currently open library.
caretEntry	The currently selected library entry.

Example (Visual C#)

The following Visual C# example displays a message when accessing your plug-in menu, showing the currently selected library item.

```
public void DisplayMenuInitialize(string libraryPath, LibraryEntity
caretEntry)
{
    MessageBox.Show(caretEntry.Title);
}
```

GetMenuEntry Function

This function retrieves the commands that will appear on your plug-in's library window menu. HotDocs calls it repeatedly until it fails, or until *menuText* is an empty string ("").

Syntax

```
void GetMenuEntry( int menuPosition, ref string menuText, ref HotDocs.Icon Icon, ref
bool enabled, ref bool callAgain, int commandId )
```

Parameters	Description
menuPosition	A 0-based counter that tells your plug-in code how many times HotDocs has called *GetMenuEntry*. For example, when this counter is **1**, your code should return the command you want to appear as the second entry in your plug-in's library window menu.
menuText	The text for the menu entry. If *menuText* is a hyphen (-), HotDocs will add a separator to the menu.
Icon	The icon that will appear next to the menu entry.
enabled	Indicates if the menu entry should be enabled (TRUE) or disabled (FALSE).
callAgain	Indicates if the function should be called again to retrieve the next menu entry (TRUE), or if your plug-in is finished adding entries to the menu (FALSE).
commandId	An identifier for the menu entry. When a user selects an item from your plug-in's menu, HotDocs will pass this identifier to the *Command* function so your plug-in knows which command was selected.

Example (Visual C#)

The following Visual C# example implements the *GetMenuEntry* function:

```
public void GetMenuEntry(int menuPosition, ref string menuText, ref Icon
Icon, ref bool enabled, ref bool callAgain, int commandId)
{
    HotDocs.Icon icon = new Icon();
    icon.LoadIcon(@"C:\images\MenuEntry.ico");

    switch (menuPosition)
    {
        case 0:
            menuText = "HotDocs Website";
            enabled = true;
            callAgain = true;
            cmd1 = commandId;
            break;
```

```
        case 1:
            //Add a separator
            menuText = "-";
            enabled = false;
            break;
        case 2:
            menuText = "Menu Sample w/icon";
            Icon = icon;
            enabled = true;
            callAgain = false;
            cmd2 = commandId;
            break;
        default:
            break;
    }
}
```

GetMenuTitle Function

This function retrieves the title for your plug-in's library window menu. If it fails, or if the text returned is an empty string (""), HotDocs will not display the menu.

Syntax

```
void GetMenuTitle( ref string menuTitle )
```

Parameters	Description
menuTitle	The title of your plug-in's library window menu.

Example (Visual C#)

The following Visual C# example implements the *GetMenuTitle* function:

```
public void GetMenuTitle(ref string menuTitle)
{
    menuTitle = "Menu Ext Plugin";
}
```

Initialize Function

HotDocs API must not be used during a plug-in's initialization. Plug-ins must wait until HotDocs is itself fully initialized (e.g. after all the plugins have loaded) before calling its APIs, or it risks leaving HotDocs running (hidden) after the user closes it.

This function is called when HotDocs starts up to determine if it should load the plug-in. If the function fails, HotDocs will not load the plug-in.

When HotDocs calls this function, it has not completely loaded its COM DLLs, which means you cannot call HotDocs COM interfaces in this function. Instead, you can call those interfaces in *LibraryInitialized*, which HotDocs calls after they are made available.

Syntax

```
void Initialize( )
```

Example (Visual C#)

The following Visual C# example implements the *Initialize* function.

```
public void Initialize()
{
}
```

LibraryInitialized Function

This function is called when the library is initialized.

Syntax

```
void LibraryInitialized ( )
```

Example (Visual C#)

The following Visual C# example displays a message box when the library is initialized.

```
public void LibraryInitialized()
{
    MessageBox.Show("The library has been initialized");
}
```

IOutputPlugin Interface

IOutputPlugin Interface

This interface enables you to create an output plug-in. Once you have created the plugin, when you are assembling a document, you can see the plug-in by selecting **File** > **Send Document To...** The plug-in also adds a new End of Interview output option. You can, for example, create a plug-in that outputs documents to places like Google Drive or SkyDrive.

When HotDocs starts up, it attempts to load each registered plug-in. If it finds a plug-in that implements the *IOutputPlugin* interface, HotDocs calls:

1. *Initialize* to load and initialize the plug-in. (If *initialize* returns a failure code, HotDocs will stop calling plug-in functions.)

2. *LibraryIntialized* to complete initialization of the plug-in.

3. *GetPlugInfo* to retrieve the plug-in information.

When a user selects your plug-in's menu, HotDocs then calls:

4. *DocumentAssembled* to perform output options on the assembled template.

Functions

Function	
DocumentAssembled	After document is assembled, this is called to output the plug-in to its destination repository.
GetPlugInfo	This function is called to get information about plug-in functionality.
Initialize	This function is called when HotDocs starts up and add-in is loaded.
LibraryInitialized	This function is called when HotDocs starts up and add-in is loaded and the main window is created.

Properties

Property	Description
CommandId	This property is called to get the command Id (such as menu ID) associated with this plug-in.

Example

The following Visual C# example implements the *HotDocs.IOutputPlugin* interface along with using the *HotDocs.IPluginPreferences* interface:

```
[ComVisible(true)]
[Guid("12345678-1111-2222-AAAA-DDDDDDDDDDDD")]
public class SendToFolder : HotDocs.IOutputPlugin, HotDocs.IPluginPreferences
{
    private string _dlgfolderPath;
    private int _commandId = 0;
    private string _outputPath = "OutputFolder";

    //Copy assembled template to folder of user choice.
    public bool DocumentAssembled(string filePath, bool bIsTempFile, string
templateTitle, string templateFilePath)
    {
        RegistryKey getKey =
Microsoft.Win32.Registry.CurrentUser.OpenSubKey(_outputPath, true);
        if (getKey != null)
        {
            string folderPath = (string)getKey.GetValue(_outputPath);
            string getExt = System.IO.Path.GetExtension(filePath);

            if (System.IO.File.Exists(System.IO.Path.Combine(folderPath,
templateTitle) + getExt))
            {
                int i = 0;
                while
(System.IO.File.Exists(System.IO.Path.Combine(folderPath, templateTitle) + i
+ getExt))
                {
                    i++;
                }
                System.IO.File.Copy(filePath,
System.IO.Path.Combine(folderPath, templateTitle) + i + getExt);
            }
            else
            {
                System.IO.File.Copy(filePath,
System.IO.Path.Combine(folderPath, templateTitle) + getExt);
            }
        }
        else
        {
            createFolderPath();
        }
        return true;
    }

    public void GetPluginInfo(ref string UI_Name, ref string pluginToken, ref
bool bIncludeAtInterviewEnd, ref Icon Icon, ref string
supportedFileExtensions)
    {
        UI_Name = "Output Folder";
        pluginToken = "HDSendToFolderOutputPlugin";
        bIncludeAtInterviewEnd = true;
```

```
        Icon.LoadIcon(@"C:\images\SendToFolder.ico");
        supportedFileExtensions = ".docx;.rtf;.pdf";
    }

    public void Initialize()
    {
    }

    public void LibraryInitialized()
    {
    }

    public int commandId
    {
        get
        {
            return _commandId;
        }
        set
        {
            _commandId = value;
        }
    }

    //Preference: Calls method, opens a dialog and allows users to set a
folder path that is written to the registry.
    public void Edit(IntPtr parentWindowHandle)
    {
        createFolderPath();
    }

    //Opens dialog form to allow user to set preferences.
    public void createFolderPath()
    {
        dlgChooseFolder dlg = new dlgChooseFolder();
        dlg.Show();
        _dlgfolderPath = dlg.folderPath;
    }

    [ComRegisterFunction]
    public static void RegisterPlugin(Type t)
    {
        HotDocs.Application app = new HotDocs.Application();
        app.Plugins.Register("{12345678-1111-2222-AAAA-DDDDDDDDDDD}",
"SendToFolder Output Plugin", 100, 1);
        System.Runtime.InteropServices.Marshal.ReleaseComObject(app);
    }

    [ComUnregisterFunction]
    public static void UnRegisterPlugin(Type t)
    {
        HotDocs.Application app = new HotDocs.Application();
        app.Plugins.Unregister("{12345678-1111-2222-AAAA-DDDDDDDDDDD}");
        System.Runtime.InteropServices.Marshal.ReleaseComObject(app);
}
```

DocumentAssembled Function

After the document is assembled, this function is called to output the plug-in to its destination repository.

Syntax

```
bool DocumentAssembled (string filePath, bool bIsTempFile, string TemplateTitle,
string templateFilePath)
```

Example (Visual C#)

The following Visual C# example implements the *DocumentAssembled* function. This example takes advantage of the *IPluginPreferences* interface to save users plug-in preferences and stores those preferences to the registry:

```
//Copy assembled template to folder of user choice.
private string _dlgfolderPath;
private string _outputPath = "OutputFolder";

public bool DocumentAssembled(string filePath, bool bIsTempFile, string
templateTitle, string templateFilePath)
{
    RegistryKey getKey =
Microsoft.Win32.Registry.CurrentUser.OpenSubKey(_outputPath, true);
    if (getKey != null)
    {
        string folderPath = (string)getKey.GetValue(_outputPath);
        string getExt = System.IO.Path.GetExtension(filePath);

        if (System.IO.File.Exists(System.IO.Path.Combine(folderPath,
templateTitle) + getExt))
        {
            int i = 0;
            while (System.IO.File.Exists(System.IO.Path.Combine(folderPath,
templateTitle) + i + getExt))
            {
                i++;
            }
            System.IO.File.Copy(filePath, System.IO.Path.Combine(folderPath,
templateTitle) + i + getExt);
        }
        else
        {
            System.IO.File.Copy(filePath, System.IO.Path.Combine(folderPath,
templateTitle) + getExt);
        }
    }
    else
```

```
    {
        createFolderPath();
    }

    return true;
}

//Open dialog and allow user to set preferences.
public void createFolderPath()
{
    dlgChooseFolder dlg = new dlgChooseFolder();
    dlg.Show();
    _dlgfolderPath = dlg.folderPath;
}
```

GetPlugInfo Function

This function is called to get information about plug-in functionality.

Syntax

```
void GetPluginInfo ( ref string UI_Name, ref string pluginToken, ref bool
bIncludeAtInterviewEnd, ref HotDocs.Icon Icon, ref string supportedFileExtensions)
```

Example (Visual C#)

The following Visual C# example implements the *GetPlugInfo* function:

```
public void GetPluginInfo(ref string UI_Name, ref string pluginToken, ref
bool bIncludeAtInterviewEnd, ref Icon Icon, ref string
supportedFileExtensions)
{
    UI_Name = "Output Folder";
    pluginToken = "HDSendToFolderOutputPlugin";
    bIncludeAtInterviewEnd = true;
    Icon.LoadIcon(@"C:\images\SendToFolder.ico");
    supportedFileExtensions = ".docx;.rtf;.pdf";
}
```

Initialize Function

HotDocs API must not be used during a plug-in's initialization. Plug-ins must wait until HotDocs is itself fully initialized (e.g. after all the plugins have loaded) before calling its APIs, or it risks leaving HotDocs running (hidden) after the user closes it.

This function is called when HotDocs starts up and add-in is loaded.

When HotDocs calls this function, it has not completely loaded its COM DLLs, which means you cannot call HotDocs COM interfaces in this function. Instead, you can call those interfaces in *LibraryInitialized*, which HotDocs calls after they are made available.

Syntax

```
void Initialize()
```

Example (Visual C#)

The following Visual C# example implements the *Initialize* function.

```
public void Initialize()
{
}
```

LibraryInitialized Function

This function is called when HotDocs has loaded, an add-in has loaded, and the main window has been created.

Syntax

```
void LibraryInitialized()
```

Example (Visual C#)

The following Visual C# example displays a message box when the library is initialized.

```
public void LibraryInitialized()
{
    MessageBox.Show("The library has been initialized");
}
```

CommandId Property

This property is called to get the command Id (such as menu ID) associated with this plug-in.

Syntax

```
int commandId [ set; get; ]
```

Example (Visual C#)

The following Visual C# example implements the *CommandId* property:

```
private int _commandId = 0;
public int commandId
{
    get
    {
        return _commandId;
    }
    set
    {
        _commandId = value;
    }
}
```

IPluginPreferences Interface

IPluginPreferences Interface

This interface lets you set preferences for all Output or Plug-in interfaces implemented.

Function	Description
IPluginPreferences	This function is called when a user selects the preferences button found by navigating to the **Tools** menu > **Options**. When the HotDocs Options dialog opens select the **Plugins** folder and navigate to the plugin you want to edit on the Plugins list.

Example

The following Visual C# example implements the *IPluginPreferences* interface calling a method that opens a new form allowing the user to set preferences for the plugin. The example shown is used in conjunction with the *IOutputPlugin* interface :

```
//IPluginPreferences: Calls custom method.
public void Edit(IntPtr parentWindowHandle)
{
    createFolderPath();
}

//Opens dialog form allowing user to set preferences.
public void createFolderPath()
{
    dlgChooseFolder dlg = new dlgChooseFolder();
    dlg.Show();
    _dlgfolderPath = dlg.folderPath;
}

//**Code for the custom dlgChooseFolder form **
public partial class dlgChooseFolder : Form
{
    string OutFolder = "OutputFolder";

    public dlgChooseFolder()
    {
        InitializeComponent();

        RegistryKey getKey =
Microsoft.Win32.Registry.CurrentUser.OpenSubKey(OutFolder, true);
        if (getKey != null)
        {
            string pathName = (string)getKey.GetValue(OutFolder);
            lblPath.Text = pathName;
        }
    }

    private void btnChooseFolder_Click(object sender, EventArgs e)
    {
        FolderBrowserDialog fldPath = new FolderBrowserDialog();
        fldPath.SelectedPath = lblPath.Text; fldPath.ShowNewFolderButton =
true;
        if (fldPath.ShowDialog() == DialogResult.OK)
        {
            lblPath.Text = fldPath.SelectedPath;
            RegistryKey key;
            key =
Microsoft.Win32.Registry.CurrentUser.CreateSubKey(OutFolder);
            key.SetValue(OutFolder, fldPath.SelectedPath);
            key.Close();
        }
        else
        {
            MessageBox.Show("Operation canceled");
        }
    }

    public string folderPath
    {
        get
        {
```

```
            return lblPath.Text;
        }
        set
        {
            lblPath.Text = value;
        }
    }

    private void btnSave_Click(object sender, EventArgs e)
    {
        this.Close();
    }
}
```

IPluginPreferences Function

This function is called when a user selects the preferences button found by navigating to the **Tools** menu > **Options**. When the HotDocs Options dialog opens select the **Plugins** folder and navigate to the plugin you want to edit on the Plugins list.

Syntax

```
void Edit( System.IntPtr parentWindowHandle)
```

Example [Visual C#]

The following Visual C# example implements the *IPluginPreferences* function calling a method that opens a new form allowing the user to set preferences for the plugin.

```
//IPluginPreferences: Calls custom method.
public void Edit(IntPtr parentWindowHandle)
{
    createFolderPath();
}
//See IPluginPreferences Interface for the remaining code
```

Contact HotDocs Sales and Support

HotDocs Technical Support

Support for customers with technical support agreements is available by calling the numbers below. To expedite your call, please be at the computer on which the program is running.

Outside the European Union:

Method of Contact	Information
Telephone	(800) 828-8328 (U.S.) +1 801 615 2200 (International) U.S. technical support is available from 7:00am to 6:00pm (MST), Monday through Friday.
E-mail	support@hotdocs.com
Web Site	http://www.hotdocs.com/support/

Inside the European Union:

Method of Contact	Information
Telephone	0870 0100 676 (U.K.) +44 131 220 9027 (International) U.K. technical support is available from 9:00am to 5:00pm (GMT), Monday through Friday.
E-mail	tech@hotdocs.co.uk
Web Site	http://www.hotdocs.com/support/

You may also find answers or solutions to questions you have in the HotDocs Wiki.

Click here for information on providing the HotDocs Publications team with documentation feedback.

HotDocs Sales Support

Experienced HotDocs consultants are available to help you with a variety of services, including integrating HotDocs with other products, building a template library, or providing training. Please contact your sales representative to learn more.

Outside the European Union:

Method of Contact	Information
Telephone	(800) 500-3627 (U.S. Sales) (801) 615-2200 (U.S. Business) +44 131 226 3999 (International)
Fax	(877) 356-3627 (U.S.) (801) 868-3627 (International)
E-mail	sales@hotdocs.com
Web Site	http://www.hotdocs.com http://www.hotdocs.com/products http://www.hotdocs.com/services
Address	387 South 520 West Suite 210 Lindon, UT 84042

Inside the European Union:

Method of Contact	Information
Telephone	0870 606 6050 (U.K.) +44 131 226 3999 (International)
Fax	0131 220 9024 (U.K.) +44 131 220 9024 (International)
E-mail	info@hotdocs.co.uk
Web Site	http://www.hotdocs.co.uk
Address	14 South Charlotte Street Edinburgh, EH2 4AX Scotland

Documentation Feedback

To improve the quality of the help file, we invite you to make comments or suggestions. When doing so, please include as much information about your experience using the documentation as possible. For example, if commenting about a specific topic, include the name of the topic in your feedback.

E-mail your comments and suggestions to publications@hotdocs.com.

The HotDocs Publications team cannot respond to technical support or project consulting issues. We are mainly interested in problems with the documentation itself—such as erroneous information, grammatical and spelling errors, or suggestions for topics to include in the next release of the software.

Glossary

.

.ANS: File name extension that designates that the file is a HotDocs answer file. Starting with the release of HotDocs 2009, all answers files (even those with the .ANS file name extension) are saved in XML format. The .ANS file name extension is retained to ensure backwards compatibility with HotDocs 2008.

.ANX: File name extension that designates a HotDocs answer file.

.CMP: File name extension that designates that the file is a component file.

.DOCX: File name extension that designates that the file is a Microsoft Word document. (See text document.)

.DOT: File name extension that designates that the file is a Microsoft Word DOT template. (See text document.)

.HDA: File name extension that designates that the file is a HotDocs auto-assemble file.

.HDI: File name extension that designates that the file is HotDocs auto-install file.

.HFD, .HPD: File name extension that designates that the file is a HotDocs form document.

.HFT, .HPT: File name extension that designates that the file is a HotDocs form template.

.PDF: File name extension that designates that the file is a Portable Document Format file, a format created and supported by Adobe. PDFs are a useful way of distributing documents in a format most users can view—as long as they have Adobe Acrobat, Adobe Reader, or HotDocs Filler (with HotDocs PDF Advantage) installed. With PDF Advantage, template developers can also create PDF-based form templates. They can also create PDFs from assembled documents.

.RTF: File name extension that designates that the template file is a Word RTF file. (See text template.)

.WPD: File name extension that designates that the file is a WordPerfect document. (See text document.)

.WPT: File name extension that designates that the file is a WordPerfect template. (See text template.)

A

accelerator: A key or key combination that quickly performs routine tasks in HotDocs. For example, rather than click the Print button, a user can press Ctrl+P and the document will be printed. Accelerators are useful when users don't want to use the mouse.

addendum: The last section of a form document that contains answers that don't fit in their allotted fields on the actual form. (See also overflow.)

additional text: See dialog element.

ADO: Short for ActiveX Data Objects, it's a data presentation layer that lets HotDocs communicate with a database so HotDocs can retrieve data from it and use it to assemble a document. (See also ODBC.)

answer: Data users enter during an interview, or data provided by your integration. Answers are usually merged into the document, but sometimes they are used to calculate other answers that are used in the document.

answer collection: A file that contains the answers associated with the variables in one or more HotDocs templates. (See also answer file.)

answer file: A saved file that contains the answers entered during an interview. Often users save their answers in a file so they can use them to assemble other similar documents. (See also answer collection.)

Answer File Manager: The library used to manage answer files. With Answer File Manager, users can group answer files, view histories of their usage, and so forth. (The alternative is using Windows Explorer to find, view, and use answer files.)

answer library: See library and Answer File Manager.

answer management: The system of using Answer File Manager to store and manage answer files. (The alternative is using Windows Explorer to find, view, and use answer files.)

answer sharing: The process of creating and using same-named variables in multiple templates so that a user can use the same answer file to assemble multiple documents. (Can also be called variable flow-through.)

answer source: An answer file or DLL that is linked to a specific dialog in an interview. Users can enter their answers in an answer source and have those answers available to them on demand. (During an interview, a Select From Answer SourceSelect button appears on the dialog. The user clicks this button and has access to the answers in the answer source.)

Answer Summary: A brief report HotDocs generates that lists the questions asked during an interview, followed by the answers that were entered. (See also Question Summary.)

answer wizard: A button attached to a form field that users can click during direct-fill assembly. When they click this button (Answer Wizard), a pop-up interview appears, asking one or more questions that are required in order for an answer to be merged in the field. Frequently, answer wizards are assigned to inactive fields in a form.

API: See HotDocs API.

ascend: The process of sorting answers in alphanumeric order, from 1 to 9, and from A to Z. (See also descend.) You can also sort items in a template library, clause library, and answer library.

ASK instruction: An instruction that forces a dialog to be asked at a specific location in the script or template. Frequently, ASK instructions are used when creating an interview component. They allow developers to control the order in which dialogs are asked during the interview.

assemble, assembly: See document assembly.

Assembly Queue: A dialog box that shows a list of assemblies—pending, current, and completed. Users can open the Assembly Queue by clicking its button (Assembly Queue) in the assembly window toolbar. It is most useful when users have selected multiple templates for assembly.

assembly window: The window that appears when a user selects a template to assemble. By default, it includes the Interview tab, the Document tab, the Question Summary tab, and Answer Summary tab. Each of these tabs displays something unique about the document being assembled, such as the questions that are required to customize the document or the assembled document itself.

auto-assemble file: A self-executable file that contains one or more templates and their related files. When packaged in an auto-assemble file (or HDA), the files are temporarily extracted and used to assemble the document. Once assembly is complete, the extracted files are deleted. Auto-assemble files are useful if template developers don't want users to have editing access to the template files themselves.

auto-install file: A self-executable file that contains one or more templates and their related files. When extracted, the files are saved to disk and references to them are added to a library. Auto-install files provide a useful way to distribute templates or updates to template sets.

automate, automation: The process of converting any document (text or form) into an interactive template. At its very core, automation is replacing changeable text in the document with variables. Additional automation steps include making text in the template conditional, repeating sections of the template so multiple answers can be entered, and inserting other boilerplate text into the template.

Automator: See HotDocs Automator.

B

bar code: A format for an answer or a group of answers so that data can be quickly scanned using an optical scanner. Bar codes are supported in both form templates and text templates. (In form templates, developers assign the PDF417 property to the field. In text templates, developers assign the preferred bar code font at the Advanced group of the Variable Field dialog box.)

binary files: In versions of HotDocs prior to HotDocs 2009, represents the format HotDocs-specific files—such as library files, component files, and so forth—were saved in. Binary file formats are common in most software applications. They allow information about the files to be encoded for storage and processing purposes. However, one limitation of storing information in HotDocs in binary format is that third-party application developers aren't able to inspect, edit, or otherwise make use of information contained in the files. Another limitation is that the binary formats used in HotDocs do not support the use of foreign characters (for example, international characters that are not represented in your computer system's default language).

browser: A window that allows users to view HTML documents. When working with HotDocs Server, interviews are displayed in a browser window rather than the regular HotDocs assembly window.

built-in variable: A predefined variable that performs a special function in a template, such as inserting either today's date or the name of the current answer file. Built-in variables include TODAY, PN#, ANSWER FILE NAME, and COUNTER.

C

century rollover: A HotDocs setting that controls whether years entered as two digits appear as 1900-century years or 2000-century years.

check-box field: A type of form template field that represents some sort of pre-existing option a user must select, such as a true/false value or a multiple-choice value.

chevrons: The double-angle brackets (« ») that surround a variable in a text template. Together, the chevrons and variable name make up the variable field, for example, «Client Name».

child dialog: A dialog that is inserted within another dialog. When it's inserted, it becomes linked to that dialog—users can't answer questions in it without first viewing the parent dialog. Usually the two dialogs are related in content or purpose.

clause: Predefined sections of text that can be selected and added to an assembled document. Usually clauses are grouped together in a clause library so users can choose which ones they want to insert, although some clauses are merged in the document automatically

clause archive: A compressed file that contains all of the clauses for a given template or clause library. During assembly, clauses in the archive are extracted so they can be selected and added to an assembled document.

Client: The application or integration that holds a reference to objects hosted by the server.

command-line option: An instruction used to control the operation of HotDocs. These instructions, or commands, are added to any command line that causes HotDocs to run. They can alter the operation of specific templates, or they can affect the overall operation of HotDocs.

comments: Notes or thoughts entered by the template developer either in a script or in a template. Comments are one way to document processes within the template. If entered correctly, they will not be visible to users in the assembled document.

component: An element in a HotDocs template that displays or stores information about the answers that are merged. Examples of components include variables, dialogs, dialog elements, merge text groups, and formats.

component file: The file that stores all of the components used in a template. The component file and template file are both necessary for template development and document assembly to work correctly. Developers use Component Manager to work with components.

Component Manager: The tool used to coordinate component usage in a template. Component Manager shows all of the components used in the template and provides options for working with those components.

Computation variable: A type of component that performs calculations or executes other instructions within the template. Computation variable scripts are created using the HotDocs scripting language.

conditional text: Text in the template that should be included in the assembled document only under certain circumstances. Conditions are controlled using IF instructions and expressions.

control field: A type of form template field that is used for behind-the-scenes tasks, including inserting related templates and assigning values to variables, just to name a few.

COUNTER: An expression that keeps track of the current number of repetitions in a repeated dialog. Each time a new repetition is added, the COUNTER is increased.

custom interview: A script that controls how and the order in which variables and dialogs are asked during an interview. The template developer creates this script.

D

database: A file that contains a collection of data. Template developers can map variables in templates to fields in a database table so that answers can be retrieved from it and merged in the assembled document.

Database Connection: See HotDocs Database Connection.

date detection: The HotDocs setting that controls how HotDocs interprets and merges dates entered during the interview—for example, whether the date appears as DAY MONTH YEAR (British), or MONTH DAY YEAR (United States).

Date variable: A type of component that merges a date in the document.

DEBUG: An instruction developers can insert in a template or script that lets them troubleshoot problems they are experiencing with their automation. While testing the script or template in debugging mode, HotDocs walks the developer through it, step by step, so he or she can see exactly how the script or template is producing the unexpected result.

default interview: The interview HotDocs automatically generates based on the order variables are asked in the template.

default word processor: When multiple word processors (for example, Word and WordPerfect) or when multiple versions of a single word processor (for example, Word 2000 and Word XP) are installed, the word processor HotDocs uses by default for automation and document assembly.

delimiter: A character, such as a tilde (~) or vertical bar (|), that delineates answers or values in a script or instruction.

descend: The process of sorting answers in reverse alphanumeric order, from 9 to 1, and from Z to A. (See also ascend.)

detect: In a form template, the process of aligning a variable field with its surrounding field borders.

developer: See template developer.

dialog: In template development, represents the component in which the developer groups variables and other components. In document assembly, represents the group of questions in the Interview tab of the assembly window where users enter their answers.

dialog element: A component that lets developers more easily add additional text, hyperlinks, buttons, graphics, lines, and spacing to dialogs. These can help make the dialog more visually pleasing and informative.

direct-fill assembly: The process of entering answers directly at the Form Document tab of the assembly window rather than answering questions at the Interview tab.

document: The file that is created after a template has been assembled.

document assembly: The process HotDocs goes through as it processes scripts in the template and merges answers into the document. At the end of the assembly process, the user has a document tailored to his or her needs.

document manager: A third-party application that stores various data files, including documents and answer files. Using a document manager, users can track versions and show histories of the document as well as enter other physical data about the files being stored, such as the date they were created, who created them, and so forth.

Document Preview tab: A tab of the assembly window that shows how the text document has been assembled using the answers entered in the interview. (See also Form Document tab and Interview tab.)

double-angle bracket: See chevrons.

duplicate: The process of copying a variable to create a new one.

E

Edit field: A type of form template field that is used for entering text, dates, and numbers. It is the most commonly used type of field on a form.

ELSE IF / ELSE: See IF instruction.

End of Interview dialog: The last dialog displayed in an interview, which contains a report of the number of questions that are still unanswered. It also provides options for working with the assembled document.

example format: A predefined format for how an answer should look when it is merged in the assembled document. This allows the user to enter the answer however they want in the interview, but forces it to appear a specific way in the finished document.

explicit index: A reference to a specific answer in a list of answers. For example, to merge the third answer from a list, a template developer would assign the index number of [3] to the variable that is being merged, like this: «Service Date[3]». The third date in the list would then be merged.

expression: A command in a script that retrieves a special value. Expressions help calculate dates, sums, and so forth.

F

field: A place in the template that denotes where users' answers should be merged, or where a specific instruction should be executed. In a text template, a field is denoted by chevrons. In a form template, a field is denoted by a colored box that is overlaid on the form's static text.

file name extension: Three characters appended to a file name that identify the type of file so Windows knows what program to use to work with the file.

fill: The process of assembling a form document.

fillable field: In Adobe Acrobat or Reader, represents a dynamic field in which a user can enter data while viewing the document. Using HotDocs, users can create fillable PDF templates from these PDF documents that contain fillable fields.

Filler: See HotDocs Filler.

filter: A script that removes unrelated or unwanted answers from a list of answers. For example, perhaps there is a list of a client's children but only the names of minor children should be merged. A filter can extract just this data from the list.

fixed value: A predefined answer, such as a date, number, or string of text. When working with instructions and expressions, placeholders are replaced either with fixed values or with variables.

foreign language DLL: A file that allows template developers and users to access Date variable and Number variable formats in languages other than English. This allows these dates and numbers to be formatted correctly in the assembled document. Supported languages include French, Spanish, German, Swiss German, Austrian German, Dutch, and Italian.

form document: The file that is created from an assembled form template. Form documents are distinguished from text documents by the design of the document—forms are static in nature,

meaning the underlying text of the document cannot be changed or modified. (See also text document.)

Form Document tab: A tab of the assembly window that shows how the form document has been assembled using the answers entered in the interview. When viewing the Form Document tab, users can enter or change their answers by clicking on the form fields and changing the answer.

form template: A template that is created and automated in HotDocs Automator. It is distinguished from a text template by the fact that the underlying text cannot be modified because it is static. (See also form document.)

format example: See example format.

H

HotDocs API: The HotDocs Application Programming Interface. It contains the functions you use to integrate your application with HotDocs.

HotDocs Automator: The tool used to automate form templates, or those templates whose underlying static text cannot be changed. Examples of form templates include tax preparation forms, applications, and so forth.

HotDocs Compare: Starting with the release of HotDocs 2009, HotDocs Compare is no longer available. It was a HotDocs add-in tool that is used to compare different versions of an assembled document. Using HotDocs Compare, users could take a "snapshot" of an assembled document, change some answers in the interview, and then compare the two versions.

HotDocs Database Connection: A tool that provides the mapping needed to connect templates to a database. Answers can be retrieved from the database during the interview, which keeps users from manually having to enter their answers. (Starting with the release of HotDocs 2008, HotDocs Database Connection (the separate product) was fully integrated into all editions of HotDocs.)

HotDocs Filler: The application used to view assembled form documents.

HotDocs Options: A section of the software where template developers and end users can set their preferences for working with HotDocs.

HotDocs PDF Advantage: A HotDocs add-in tool that allows the creation and automation of PDF-based form templates. PDF Advantage can also be used to save most types of documents as PDF (assembled or otherwise).

HotDocs Player Edition: A version of HotDocs that is used for assembling published (and registered) templates.

HotDocs Professional Edition: A version of HotDocs, now known as HotDocs Developer, that contains the tools necessary to automate a simple to highly complicated set of both text and form templates. It is also used to assemble both text and form templates. (See also HotDocs Standard Edition.)

HotDocs Server: The Web-based version of HotDocs. When using HotDocs Server, interviews are presented in a user's Web browser. Answers are then sent back to a server where the document can be assembled. HotDocs Server allows users to create documents and answer files without requiring them to have HotDocs installed on their desktop.

HotDocs Standard Edition: A version of HotDocs, now known as HotDocs Developer LE, that contains the tools necessary to automate a simple to moderately complicated set of text templates. HotDocs Standard can also be used to assemble text and form documents. (See also HotDocs Professional Edition.)

HotDocs Variable Mapping dialog box: A tool in HotDocs that allows source fields to be mapped to HotDocs variables.

I

IF instruction (also ELSE IF, ELSE, END IF): A set of instructions and expressions that control the inclusion and exclusion of optional text in a document. IF instructions are based on either True/False variables or true/false expressions. IF instructions can also be used to control whether certain instructions or expressions are processed in computation or dialog scripts.

import: When working with libraries, the process of copying template files into the currently viewed library. These files can be imported for assembly only or for editing and assembly. When working with answer files, the process of copying an answer file to the default Answers folder and then adding it to the answer library.

inactive field: A form document field on which the user cannot directly enter an answer. Fields can be inactive for any number of reasons. For example, the field may be conditioned or it may contain a Computation variable. Frequently, a template developer provides an answer wizard to help the user answer all of the questions that will make the field active.

infinite loop: The process of a HotDocs script repeatedly reprocessing itself until HotDocs stops responding. For example, a computation can repeatedly scan a text string, character by character, for a specific value. As HotDocs searches for this value, it adds information to what is called the processing stack. If too much information gets added to this stack, HotDocs may get into an infinite loop and stop responding.

INSERT instruction: An instruction that inserts one template into another. For example, if boilerplate text needs to be used in multiple documents, a template that contains that text can be created and inserted in each template that requires it (via an INSERT instruction). This way, if changes need to be made to the text, the change has to be made in only one template.

inserted dialog: See child dialog.

inserted template: A template that is inserted into another template using an INSERT instruction.

instant update: A command in the HotDocs assembly window that, when selected, updates the interview every time a user enters or changes an answer in the interview. Sometimes this updating may cause HotDocs to behave sluggishly as users move between answer fields. In such cases, the user can turn the instant update command off. Then HotDocs will update the interview only as it needs to.

instruction: A command in a script or template that performs a special task, such as inserting a template or asking a dialog at a specific place in the interview.

intake interview: See interview template.

interview: A presentation of questions that must be answered in order to create an assembled document. The interview is viewable by clicking the Interview tab of the assembly window.

interview component: A computation script that defines how a custom interview will look and function. An interview component usually includes ASK instructions to ask all of the dialogs/variables in the interview. The script frequently includes other instructions, such as REPEAT instructions and INSERT instructions as well as conditions for using these instructions. The name of this component is defined at the Component File Properties dialog box.

interview outline: The leftmost pane of the assembly window that lists all of the dialogs in the interview. Viewing the outline shows the natural progression of the interview. Icons in the outline also indicate whether questions in the associated dialog are completely answered, partially answered, or not answered at all.

Interview tab: A tab of the assembly window that shows the outline of questions in the interview as well as the dialogs that contain the questions. Users enter answers while viewing the Interview tab. (See also interview.)

interview template: A template that contains a series of interview questions designed to gather information about a person (or persons) or matter. Answers are saved in an answer source file, which can then be linked to a dialog in a template that requires the same information. Generally, interview templates can be used to create a list of possible answers so users have more options to choose from.

iteration: One instance of a repeated dialog.

J

JS files: Stands for JavaScript files, which are used to display interviews in a Web browser. When templates are published for use with HotDocs Server, HotDocs generates these JavaScript files for the interview.

K

keywords: A broad term used to describe scripting instructions, expressions, and operators. Keywords are used in a script and generate values or perform certain tasks.

L

label: In a text template, an identification assigned to a REPEAT, IF, or SPAN instruction to help the template developer identify the instruction in relation to other instructions in the template. In a form document, the text that is merged in a field when an answer overflows and is sent to the addendum. (See also reference.)

library: A window used to display and organize templates. The library does not store the actual files—instead, it contains shortcuts (or links) to the files, which are stored on disk. In addition to the template library, HotDocs also uses an answer library, which is more commonly known as Answer File Manager.

line break: A code in a Word document that indicates that text should appear on a new line within the same paragraph. For example, if the user must enter separate lines in a single paragraph (such as lines in an address), a line break should be used. (See also paragraph mark.)

linked field: Represents a HotDocs field in a fillable PDF template that is associated with an Adobe fillable field. By creating this association between a HotDocs field and a fillable field, template developers can create HotDocs fields that precisely match fields in the underlying PDF. Users who assemble the document can then edit answers associated with linked/fillable fields in the saved PDF.

list: Two or more answers to one question merged in the document.

M

manual index: See explicit index.

map file: A file used to store the associations between source fields and HotDocs variables. Your application will have one map file.

map, mapping: See variable mapping.

mark up, markup: The formatting applied to a Word template or an assembled Word document that shows simplified template development marks. For example, when viewed in Markup View, variables in a template appear between brackets rather than chevrons. (See also Markup View.)

Markup View: A view that shows a simplified version of a Word template or an assembled document. This simplified view may be useful if a non-HotDocs user must review the template or document. When viewing a template or document in Markup View, variable and answer fields are marked using brackets.

merge field: During template development, the place in the template where a variable is inserted. During document assembly, the place where the user's answer will be inserted.

merge text: The text that will be merged in a document if a user chooses a specific Multiple Choice variable option. For example, if a user chooses Male as the option, a masculine pronoun such as he or his can be merged instead of Male.

model: A tool in the script editor that template developers can use in writing scripts. A model shows the full instruction or expression—including any placeholders that must be replaced for the script to work correctly. Developers can drag these models from their respective lists and then replace the placeholders with the appropriate values.

Multiple Choice variable: A type of component that merges a predefined answer in the document.

N

navigation bar: In an interview (at the Interview tab), the toolbar used to move from dialog to dialog. In a document (at the Document tab), the toolbar used to move between merged answers in a document.

non-breaking space / hyphen: A property that can be assigned to a variable that keeps the answer from being split across two lines in the assembled document.

notation: An identification assigned to a variable name to help identify what type of variable it is. For example, Client Name TE would indicate that the variable is a Text variable. (Typical component notations include TE (Text), DA (Date), NU (Number), MC (Multiple Choice), TF (True/False), CO (Computation), and DI (dialog).)

Number variable: A type of component that merges a numeric value in the document.

O

ODBC: Short for Open Database Connectivity, it's a data presentation layer that lets HotDocs communicate with a database so HotDocs can retrieve data from it and use it to assemble a document. (See also ADO.)

operator: A symbol or word that causes either an operation (such as addition) or a comparison to be performed in a computation script or expression.

order: The process of designating the sequence in which form template fields are asked in the tab order. Establishing this order in a form is important for users who directly fill the form document.

outline: See interview outline.

overflow: Answers in a form document that do not fit in the allotted field space. Overflowing answers are usually sent to the addendum.

overlay: The process of using the Overlay Answers command to merge existing answers into the current answer file. When answers are overlaid, the answers become a part of the current answer file. They also overwrite any existing answers in the interview.

P

paragraph mark: A code in a Word document that indicates that text following the mark should appear in a new paragraph. (See also line break.)

parent dialog: A dialog that contains a child dialog.

pattern: Determines how a Text variable will be displayed and formatted in the interview and in the assembled document. By default, HotDocs includes three patterns in all new templates (Social Security number, telephone number, and time of day), but template developers can create custom patterns.

PDF417: The two-dimensional bar code format used in HotDocs Automator and HotDocs Filler.

Personal Information variable: A type of component that stores basic information about a user, such as a name, a company name, and a phone number. This information is saved in the Current User key of the Windows System Registry. Once answered, users won't be prompted to enter it again.

pick list: See answer source.

placeholder: A marker in an instruction or expression model that indicates where a value must be substituted. This value must be a literal value or a variable. Instruction and expression models help the developer use the correct syntax in a script.

pointed component file: When sharing components across multiple templates, represents the template's own component file, which, in turn, points to the shared component file.

pop-up interview: A dialog a user can display during an interview. Usually a pop-up interview shows a different view of the dialog. For example, if a user is entering answers in a spreadsheet, he or she can click the Edit RowEdit Row button and a pop-up interview appears that shows just the questions (and answers) from that particular row in the spreadsheet.

processing stack: A sequential list of templates and components HotDocs is processing at any given time. Each time a new component is processed, it is added to the stack. (Once processing is finished, it is removed.) In some instances where recursion is used in a script, the same

component is repeatedly added to the list. If the number of components exceeds the stack limit, an infinite loop error will occur. (The stack limit can be changed at the Component File Properties dialog box.)

prompt: Text that can be assigned to a variable to help the user better understand how to answer the question.

punctuate: The process of formatting a REPEAT instruction so that a list of answers will appear in sentence format, like this: The client owns real estate in New York, Pennsylvania, and Montana. (New York, Pennsylvania, and Montana are the list items. The punctuation adds the commas and the conjunction and).

Q

Question Summary: A brief report HotDocs generates that lists questions asked during an interview. The summary includes blank lines for handwritten answers. (See also Answer Summary.)

queue: See Assembly Queue.

R

reference: In a form document, the text that is added to the addendum to identify any overflow answers. (See also label.)

reference path: A folder path for a template in which the drive letter and some or all of the folder names are represented by a keyword. At runtime, this keyword is mapped to an actual path on the user's computer so that when the user accesses the template, the keyword is replaced by the path. This allows templates saved in one central location to work on multiple workstations regardless of how the drives on the workstation are mapped.

REPEAT instruction: An instruction that repeatedly asks the same variable(s) so that two or more answers can be entered. REPEAT instructions are used to create lists of answers in the document.

repeated dialog: A dialog that contains the variables that need to be repeated so that multiple answers can be entered. (See REPEAT instruction.)

repeated series dialog: One of two representations of a dialog that is repeated. With a repeated series, the dialog is asked repeatedly until all answers in the list have been entered. (See also spreadsheet dialog.)

resource: Supplemental help that can be included with a variable or dialog to help users better understand how to answer the questions they are viewing. Resources appear in the resource pane of the assembly window.

S

script: One or more instructions and/or expressions that generate a value or execute some kind of procedure.

script editor: The tool used to write a script. The script editor includes several options to make the script-writing process easier, including color-coding, auto-complete lists, and a toolbar for completing other tasks.

selection grouping: A dialog property assigned to True/False variables, clauses, and child dialogs which presents these options as check boxes (multiple-select) or option buttons (single-select).

Send to Word Processor command: A command that opens the word processor and copies the assembled document into it. Once opened in the word processor, the user can make any changes necessary to the document.

SET instruction: An instruction in a template or script that assigns a value to a variable. Variables that have their values set should not be asked again in the interview.

shared component file: A common component file to which several related templates are linked. To use a shared component file, the template's own component file must be pointed to the shared file. Changes to components in the file are reflected in all templates that use it. (See also pointed component file.)

SHOW: An instruction used in a dialog script to show variables that have been hidden in the dialog. (See also .) Usually this instruction is conditioned so that variables hide and show dynamically, based on answers the user enters.

sort: The process of alphabetizing answers in a repeated list or items in a library. Sorting can be done in ascending or descending order.

spreadsheet dialog: One representation of a repeated dialog. Each row in a spreadsheet represents one repetition in a dialog. (See also repeated series dialog.)

static text: The underlying text in a form template or document that does not change. To enter answers on a form, form fields must be created and overlaid on the static text.

strike-through field: A type of form template field that is used for crossing out static text on the form.

summary: See Question Summary and Answer Summary.

supplemental component: A term used to define components such as patterns, example formats, dialog elements, and merge text. Supplemental components are associated with regular components, but they can be created and edited as standalone components.

syntax: The language used in writing scripts. For a script to work properly, the script must be written in a way that HotDocs can understand. This language consists of instructions, expressions, operators, and values (such as text, numbers, dates, or answers users enter).

T

template: A word processor or form document that has been converted to HotDocs format so that it can be automated. When in template format, changeable text in the template can be replaced with variables. Other instructions can be added as well, such as instructions that create lists, condition text, and insert other templates.

template developer: The person responsible for automating the templates in the set. The template developer creates and inserts the variables in the template, arranges variables in dialogs, and performs other custom tasks in the template. (See also user.)

template development: See automate, automation.

test: The process of testing a variable or other component to make sure it looks right and works correctly.

test assemble: The process of assembling a document for the purpose of ensuring the interview works correctly and the automation within the template produces a correctly assembled document. During a test assembly, developers can easily edit components and have the test assembly window updated with changes.

text document: A document that is viewed in either Word or WordPerfect. It can represent a document before it is automated as well as a document after it has been assembled. When in document format, it is not associated with (or linked to) HotDocs in any way. (See also text template.)

text template: A template that is created and automated in Microsoft Word or WordPerfect. It is distinguished from a form template by the fact that the underlying text of the template can be modified. (See also text document.)

Text variable: A type of component that merges text in the document.

title: A property of a variable or dialog that specifies a more user-friendly name for the component. For example, if project standards require components be named using notations, names like Employee Name TE may not make sense to a user. However a title like Employee Name can be used instead.

True/False expression: A script that must result in either true or false. Expressions are used for merging or excluding optional text in a document. They are also used for determining which parts of a script will be executed, based on answers or other values entered by a user. Expressions are often used when a simple True/False variable doesn't convey the condition needed. (See IF Instruction.)

True/False variable: A type of component that determines a true/false status of some condition and then merges the appropriate answer or text.

U

unanswered text: Text in a text document that indicates that a question is unanswered. By default, unanswered questions appear as ***Variable Name***, but this can be customized.

Unicode: Computer specification that makes it possible for computers to represent and manipulate characters used in most of the world’s written languages. Unicode support in HotDocs makes it possible to automate and assemble documents in non-native, left-to-right-reading languages. This includes automating and assembling Microsoft Word templates as well as PDF-based form templates.

user: The customer, client, or person who assembles documents from templates. (See also template developer.)

V

value: In an interview, it represents a user's answer. In a script, it represents data that must be used in executing the script. (The value can either be a literal value or a user's answer.)

variable: A component that is used to represent changeable text (such as names, dates, numbers, etc.) in the template. Types of variables include Text, Date, Number, True/False, Multiple Choice, Computation, and Personal Information.

variable flow-through: See answer sharing.

variable mapping: The process of associating two HotDocs variables so that they can share answers. In some cases, this mapping defines the relationship between a HotDocs variable and a field in a third-party application file, such as a database table or a field in an Outlook Contacts list.

X

XML: Stands for eXtensible Markup Language. It is a computer language designed to store and transmit data between applications. Like HTML (HyperText Markup Language), it contains customized markers, or tags, that identify the information in an XML file. However, while HTML describes the way a page looks, XML controls the way data is structured, making it easy for diverse programs to access the same information. (For example, in HTML, to indicate a book title, you would italicize it using the <i> tag. In XML, you could mark the title using a <booktitle> tag. The HTML tag simply formats the text (making it italic), while the XML tag actually defines what the text is (a book title).) In HotDocs, you can save libraries, component files, and answer files in XML format.

Index

/

/af 22

/as 22

/cl 23

/da 23

/db 24

/df 23

/ed 24

/ex 24

/fia 24

/ha 25

/hi 25

/hl 25

/if 26

/is=u 26

/kig 27

/la 28

/lf 27

/ll 28

/mo 26

/na 28

/ni 29

/nw 29

/nx 29

/of 30

/ov 30

/pa 32

/pb 33

/pc 33

/pd 33

/po 34

/pr 32

/ps 31

/pt 31

/pw 34

/qs 34

/regserver 35

/sa 35

/sap 35

/si 38

/sig 37

/sl 36

/ss 37

/ssn 38

/sto 36

/stw 36

/sw 38

/tf 39

/unregserver 39

A

About

command-line options 13

Adding

assemblies to assembly queue 161, 162

assemblies to assembly queue at indexed position 163

custom menus to assembly window interface 138

custom menus to library interface 101, 102

folders to a library 201

Multiple Choice values to a Multiple Choice variable 68

new answers to AnswerCollection 85

templates to a library 202

Answer event notifications, firing when finding values for answers 80

Answer File command-line option 22

Answer files (answer collections)

closing after using 86

determining format (XML or ANS) for 92

overlaying 88

prompting to save after assembly 148

retrieving answer from 87

saving 88

uploading to a server 89

using as default 91

Answer object

about 65

events 80

methods 68, 69, 70, 71, 72, 73, 75

properties 76, 77, 78, 80

Answer Source API

about 235, 241

entry points 243

Answer source integrations

about 235

creating 241

functions 246, 247, 248, 249, 250, 252, 254, 255, 256, 257, 258, 259, 260, 261, 262

Answer Summaries, saving to specific location 145

Answer Summary command-line option 22

AnswerCollection object

about 82

initializing 86

methods 85, 86, 87, 88, 89

properties 90, 91, 92, 93, 94, 95

Answers

adding to an answer collection 85

creating new 69

determining if marked unanswered 78

returning names of 76

returning types of 77

suppressing warnings if unanswered 150

working with 65

API

about 11

answer source 235, 243

description 11

using Help file 1

Application event notifications

firing when assembly completes 129, 130

firing when assembly starts 130

firing when error occurs 131

firing when user closes library 131

firing when user opens library 132

firing when user selects menu options at library 133

firing when user selects options at library ... 132

firing when user selects template(s) for assembly 132

Application object

about 96

enumerations for 57, 58

events 129, 130, 131, 132, 133

methods 101, 102, 103, 104, 105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115, 116, 118, 119, 121

properties 122, 123, 124, 125, 126, 127, 128, 129

Asking

only unanswered questions 26

Assembled documents, printing 110

Assemblies

adding a single assembly to queue 161

adding at certain location in assembly queue 163

adding multiple assemblies to queue 162

automatically printing after complete 147

choosing a single template for 118, 119

choosing multiple templates for 115, 116

clearing from queue 163

counting number of in queue 165

maintaining reference to answer collection during 144

moving queued 164

printing documents from 110

removing from assembly queue 165

returning object for current 122

showing completed in assembly queue 147

Assembly event notifications

answer file saved 159, 160

answers needed 156

assembly completed 151

assembly started 151

assembly window closed 152

document file saved 156, 157

errors during assembly 152

user chooses custom menu option 159

user chooses interface options 158

Assembly object

about 133

enumerations for 49, 51

events 151, 152, 153, 154, 155, 156, 157, 158, 159, 160

methods 138, 139, 140, 141, 142, 143, 144

properties 144, 145, 146, 147, 148, 149, 150, 151

Assembly queue

adding assemblies to 161, 162

moving assemblies within 164

removing assemblies from 165

returning count of assemblies in 165

returning list of assemblies in 123

showing 124

showing completed assemblies in 147

AssemblyCollectionClass object

about 160

methods 161, 162, 163, 164, 165

properties 165

C

Choosing

a single template for assembly 118, 119

multiple templates for assembly 115, 116

Clause Name command-line option 23

Closing

answer file after using 86

library 196

COM Events

about 44

programming in Visual C# 47

COM objects 65, 82, 96, 133, 160, 166, 174, 183, 186, 188, 190, 192, 195, 200, 206, 210, 214, 218

Command-line options

about 13

Answer File 22

Answer Summary 22

Clause Name 23

Default Answer File 23

Discard Answers 23
Don't Brag 24
Edit Template 24
Exit HotDocs 24
Finish Interview Action 24
full list of 14
Hide Library 25
HotDocs Auto-Assemble File 25
HotDocs Auto-Install File 25
HotDocs Model 26
Installation File 26
Interview Scope 26
Keep Interview Scope 27
Library File 27
Lock Answer File 28
Lock Library 28
New Answer File 28
No Assembly Window 29
No Exit 29
No Interview 29
Output File 30
Overlay Answer File 30
Paper Size 31
Paper Tray 31
Print 32
Print Answers Only 32
Print Both 33
Print Copies 33
Print Duplex 33
Print Form Only 34
Print Without Dialogs 34
Question Summary 34
Register 35
Save Answers 35
Save Answers Prompt 35
Send to Plugin 36
Send to Word Processor 36
Show Library 36
Start Interview Group 37
Suggest Save 37
Suggest Save New 38
Suppress Installation 38
Suppress Unanswered Warning 38
Template File 39
Unregister 39
Component files (component collections)
creating new, empty 176
creating variables in 178

finding number of components in ... 181
opening ... 180
retrieving components from ... 179
returning file location for ... 182
Component object
about ... 166
methods ... 167
properties ... 167, 168, 169, 170, 171, 172, 173
ComponentCollection object
about ... 174
methods ... 176, 177, 178, 179, 180
properties ... 181, 182, 183
ComponentProperties object
about ... 183
methods ... 184, 185
properties ... 185
ComponentProperty object
about ... 186
properties ... 186, 187
Components
creating in component file ... 178
enumeration for types of ... 64
finding dialogs linked to ... 168
finding name of ... 170
finding number of in collection ... 181
finding prompt for ... 171
identifying resources for ... 169
identifying type of ... 172
linking to database fields ... 168
Connection point ... 44
Contacting
technical support ... 303
Copyright ... 3
Counting
answers in a collection ... 90
number of assemblies in queue ... 165
number of components in collection ... 181
repeated values ... 70
Creating
AnswerCollection object ... 144
new answers ... 69
new library ... 196
new, empty component file ... 176
variables in a component file ... 178
Customizing
assembly window menus ... 138
library menus ... 101, 102, 106

D

Database, linking variables in templates to fields in ... 168

Default Answer File command-line option ... 23

Default answer files, using ... 91

Deleting

assemblies from the assembly queue ... 165

custom assembly window menu items ... 139

custom library menu items ... 106

items from the library ... 203

Dependency object

about ... 188

properties ... 189

DependencyCollection object

about ... 190

methods ... 191

properties ... 191

Disabling application features in integration ... 143

Discard Answers command-line option ... 23

Documentation Feedback ... 305

E

Edit Template command-line option ... 24

Editions of HotDocs, determining which is being used ... 126

Enumerations ... 48, 49, 50, 51, 57, 58, 62, 63, 64

for assembly window interface ... 51

for assembly window menus ... 49

for library interface ... 58

for library menus ... 57

for variable types ... 64

Event sink ... 44

Events ... 80, 129, 130, 131, 132, 133, 151, 152, 153, 154, 155, 156, 157, 158, 159, 160

Example scripts

creating an Application object ... 41

Exit HotDocs command-line option ... 24

F

Features, new and enhanced ... 5, 7, 8, 9, 10

Feedback, providing ... 305

File locations

returning for answer files ... 93

returning for component files ... 182

returning for templates ... 150

specifying for answer files ... 88

specifying for answer summaries ... 145

specifying for question summaries ... 148

Finish Interview Action command-line option ... 24

G

Garbage collection ... 42

H

Handles

returning for assembly window interface 145

returning for library interface..... 127

Help file

get help using 1

Hide Library command-line option.....25

HotDocs

contacting 303

HotDocs API

about.....11

HotDocs Auto-Assemble File command-line option 25

HotDocs Auto-Install File command-line option 25

HotDocs editions, determining which is being used..... 126

HotDocs Model command-line option 26

HotDocs plug-ins

about..... 265

HotDocs Plug-ins

creating 266

creating using Visual C++ 267

HotDocs Server, publishing files for..... 110, 111

How To's

creating the HotDocs Application object.....41

I

Icon object

about.....192

methods.....193

properties.....194

Identifying

assembly window using handle.....145

library window using handle.....127

ILibraryWindowContextMenuExtension Interface

about.....270

functions.....272, 273, 274, 275

ILibraryWindowFileHandlerExtension Interface

about.....276

functions..... 278, 280, 281

ILibraryWindowIconProvider Interface

about.....283

functions..... 284, 285

ILibraryWindowMenuExtension Interface

about.....286

functions.....289, 290, 292, 293

Indexes, finding for assemblies.....163

Initializing AnswerCollection object..... 86

Install File command-line option..... 26

Integration, people involved in..... 11

Interface (GUI)

changing for assembly window 51

changing for library window 58

disabling or enabling features of assembly window 143

disabling or enabling features of library window 121

Interview Scope command-line option 26

IOutputPlugin Interface

about....293

functions....296, 298, 299

properties....299

IPluginPreferences Interface

about....300

functions....302

Iterating values for answers 73

J

JavaScript files

building for HotDocs Server interviews 110, 111

K

Keep Interview Group command-line option 27

L

Libraries

adding folders to 201

adding templates to 202

assigning descriptions to 204

assigning titles to 199

closing....196

creating new....196

opening....197

returning root folder in....199

returning specified item 203

saving....198

Library File command-line option....27

Library object

about....195

methods....196, 197, 198

properties....198, 199

LibraryEntity object

about....200

methods....201, 202, 203

properties....203, 204, 205, 206

Lock Answer File command-line option 28

Lock Library command-line option 28

M

Mapping

about....218

adding new items to variable collection....221

adding new source names to SourceNames collection....228

associating variables with source names in integration 222, 223

counting items in mapping..... 232

counting items in variable collection..... 232

counting source names in SourceNames collection 233

opening a HotDocs map file 226

opening component file 226

removing mapping from a variable collection 225

removing mapping from SourceNames collection 230

retrieving items from variable collection..... 221

retrieving source name and type from SourceNames collection..... 229, 230

retrieving variable name and source name from collection 224, 225

saving map file to specific location 226

showing interface for 227

Menus

customizing for assembly window..... 138

customizing for library.....101, 102, 106

enumeration (for assembly window) 49

enumeration (for library).....57

Moving assemblies within the assembly queue 164

Multiple Choice variables, adding options to 68

N

New Answer File command-line option 28

No Assembly Window command-line option... 29

No Exit command-line option 29

No Interview command-line option..... 29

O

Objects.... 65, 82, 96, 133, 160, 166, 174, 183, 186, 188, 190, 192, 195, 200, 206, 210, 214, 218

Opening

a component file 180

a library file 197

libraries for Application object 109

modal library to show templates..... 115, 116

Output File command-line option..... 30

Overlay Answer File command-line option..... 30

Overlaying answer files..... 88

P

Paper Size command-line option..... 31

Paper Tray command-line option 31

Passive answer source integration..... 235, 243

PDFs, saving documents as.....114

Plugin object

about.....206

properties.....207, 208, 209, 210

Plug-ins

about....265
creating....266
description....265
file handler....276
menu bar....286
shortcut menu....270
PluginsClass object
about....210
methods....211, 212, 213
properties....214
Print Answers Only command-line option....32
Print Both command-line option....33
Print command-line option....32
Print Copies command-line option....33
Print Duplex command-line option....33
Print Form Only command-line option....34
Print Without Dialogs command-line option....34
Printing
automatically when assembly completes....147
documents....110
Prompting to save answers after assembly....148
Publishing files for use with HotDocs Server....110, 111

Q

Question summaries, specifying file location where saved....148
Question Summary command-line option....34

R

Recurse values for an answer....73
Register command-line option....35
Removing
assemblies from the assembly queue....165
custom assembly window menu items....139
custom library menu items....106
items from the library....203
Repeat indexes
finding....71
finding the value for the current....78
returning....71
setting answers using....75
Repeated values
counting....71
indexing....71
returning the count of....77
Retrieving
answers from an answer file....87
components from a component file....179
Returning

answer types 77
component file names 182
current assembly object 122
file system path for current template library 126
names of answers 76
number of answers in a collection 90
number of repeated answers 77
objects in assembly queue 123
repeat indexes 71
specified item in library 203
window handles for assembly window interface 146
window handles for library interface 146

S

Save Answers command-line option 35
Save Answers Prompt command-line option 35
Saving
a library 198
answers in an AnswerCollection 88
answers using a prompt 148
document to PDF file 114
question summaries to specific location 148
Send to Plugin command-line option 36
Send to Word Processor command-line option 36
Sending assembled document to word processor 121
Setting
command-line options for use in assembly window 145
command-line options for use in library 125
current values to those specified at repeat index 75
Show Library command-line option 36
Start Interview Group command-line option 37
Status, determining for the assembly object 149
Suggest Save command-line option 37
Suggest Save New command-line option 38
Support
contacting 303
Suppress Installation command-line option 38
Suppress Unanswered Warning command-line option 38
Switches, full list of command-line 14

T

Technical Support 303
Template File command-line option 39
Template library
descriptions for templates in 150
returning path for 126
selecting a single template from 118, 119

selecting multiple templates from 115, 116

TemplateInfo object

about 214

methods 216

properties 216, 217, 218

Templates

adding to Assembly Queue 161, 162

assigning descriptions to 150

assigning file paths for 150

assigning titles to 151

U

Unanswered variables

asking only 26

Unansweredness

determining 78

suppressing warnings for 150

Underlaying answers in a collection 91

Unregister command-line option 39

Uploading answer collections to a URL 89

User interface

enumeration for assembly window 51

enumeration for library 58

manipulating 121

visible status of assembly window 151

visible status of library window 129

V

Variable mapping

allowing users to merge from multiple map files 42

controlling which object is used in assembly 147

linking variables to database fields 168

mapping fields in integration to variables 42

Variables

enumeration for types of 64

finding dialogs linked to 168

finding name of 170

finding prompt for 171

identifying resources for 169

identifying type of 172

linking to database fields 168

returning for types 77

VarMap object

about 218

methods 221, 222, 223, 224, 225, 226, 227, 228, 229, 230

properties 231, 232, 233

Versioning 128

Visual C#

programming COM events in 47

W

Warning, suppressing if questions in interview unanswered 150

Window handles

returning for assembly window interface 146

returning for library interface 127

X

XML Answer sets, specifying format for 92