Developing Plug-ins for Foxit Reader/PhantomPDF

Foxit PhantomPDF Plug-in SDK

Version 1.0

Contents

Prerequisites

Developer Audience

This document is targeted towards C/C++ developers using the SDK to create plugins for Foxit Reader/Phantom. It assumes the developer is familiar with C/C++ and is an experienced user of Foxit Reader/Phantom.

Supported Environments

Platform	Operating System	Compiler
Windows (32-bit)	Windows 2000/XP or later	Microsoft Visual Studio 6.0 or later.

MFC

Dynamically linked libraries (DLL) that link to Microsoft Foundation Classes (MFC) need to use the macro AFX_MANAGE_STATE to switch the MFC module state correctly. This is done by adding the following line of code to the beginning of functions that are exported from the DLL:

```
AFX_MANAGE_STATE(AfxGetStaticModuleState());
```

If this macro is not used, the plugin resources will fail to load in debug mode. The program will work until a function return is encountered. The `AFX_MANAGE_STATE` macro should not be used in regular DLLs that statically link to MFC or in extension DLLs.

Additional Documentation

Please refer to the *Foxit PhantomPDF API Reference* for a detailed description of the Foxit Reader/PhantomPDF Plugin API.

Overview

Purpose

This document covers how to develop Foxit Reader/Phantom plugins with the SDK. Plugins allow users to enhance the functionality of Foxit Reader/PhantomPDF. Additional features that are beyond the basic Foxit Reader/PhantomPDF feature set can be added through plugins. Developers will learn what a plugin is and how it integrates with Foxit Reader/PhantomPDF.

What can Foxit Reader/PhantomPDF Plugins do?

Customize the Foxit Reader/PhantomPDF User Interface

Developers can modify the Foxit Reader/PhantomPDF user interface through plugins. New menu items and tool bar buttons can be added to the existing interface. Custom actions can be programmed for specific user interface commands. Please refer to the "Menu" and "Toolbar" sample projects for examples of custom user interface changes done through plugins.

Manipulate PDF Documents

Here are a few PDF manipulations that can be done through plugins,

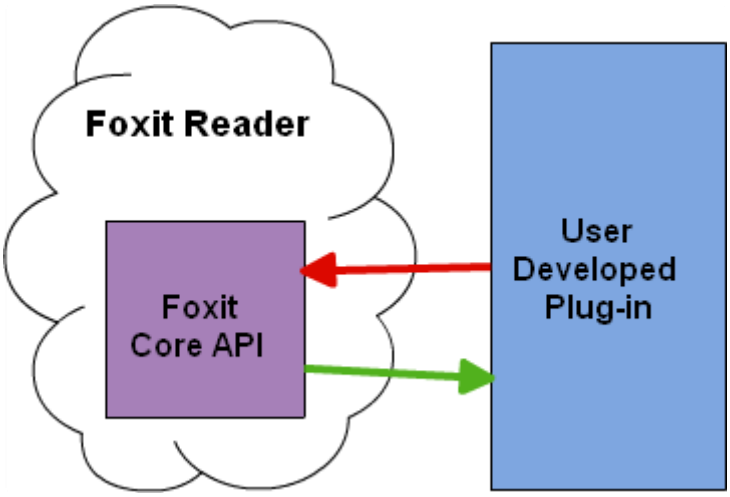
- Open a PDF document in an external dialog.
- Dynamically add and remove pages from an existing PDF document.
- Edit all of the elements contained in a PDF document.
- Set up security policies to control user permissions for PDF documents.

Please refer to the "Document" and "DRM" sample projects for examples of PDF manipulations done through plugins.

Plugin Architecture

Foxit Reader/PhantomPDF provides the host environment in which a plugin application

exists. Foxit Reader/PhantomPDF shares its resources with plugins through the Foxit Reader/PhantomPDF Core API. The Core API has control over items such as the menu bar, page view, and PDF document operations. Through these resources, plugins can make modifications to Foxit Reader/PhantomPDF. The following diagram shows the relationship between Foxit Reader/PhantomPDF and plugins.



Plug-ins are dynamically-linked extensions to Foxit Reader/PhantomPDF and are written using the Core API. The Core API is an ANSI C/C++ library. Plugins are dynamically-linked libraries (DLLs) on the Microsoft Windows platform.

There are two ways to install custom plug-ins for Foxit Reader/PhantomPDF to load it:

1. Place the plug-in in "plugins" directory which is in the same directory of Foxit Reader/PhantomPDF.

2. Place the plug-in in a specified directory and then create registry entries to associate it. The format of the registry entries is as follows:

For 64bit Windows, to make your plug-in available for all Windows users, create a registry key as following:

[HKEY_LOCAL_MACHINE\SOFTWARE\Wow6432Node\Foxit Software\PhantomAddins\YourPlgName]

For 32bit Windows, to make your plug-in available for all Windows users, create a registrykey as following:

[HKEY_LOCAL_MACHINE\SOFTWARE\Foxit Software\PhantomAddins\YourPlgName]

For 64bit Windows and 32bit Windows, to make your plug-in available for current Windows users, create a registry key as following:

[HKEY_CURRENT_USER\Software\Foxit Software\PhantomAddins\YourPlgName]

After created the registry key correctly, set the key-value pair to "YourPlgName" key as following:

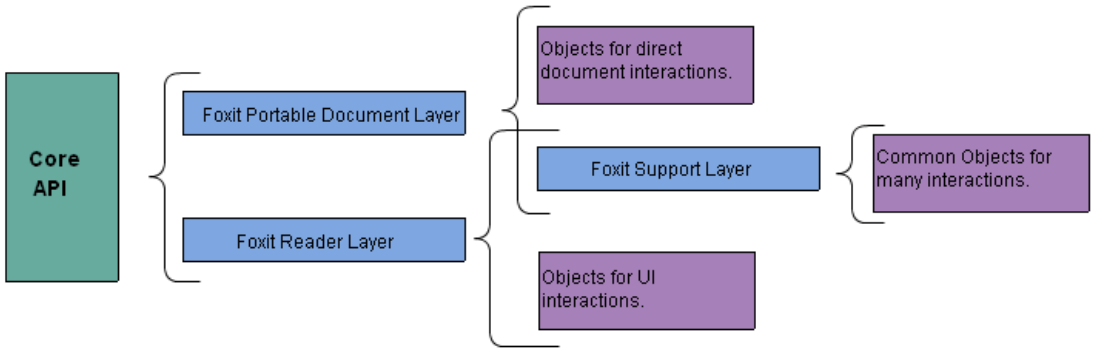
"FriendlyName "="YourPluginName"
"Description" = "The description of your plugin"
"PIPath "="The full path of your plug-in"
"LoadBehavior"= "3"

Then Foxit Reader/PhantomPDF must be restarted in order for the plug-ins to take effect.

The number of plug-ins that Foxit Reader/PhantomPDF can load at any specified time is limited. The host operating system may have a limitation on the total number of Thread Local Storage (TLS) slots available to a single process. This is a limitation of the multi-threaded model used by the Win32 API. More available TLS slots on a system equates to more plug-ins being loaded by Foxit Reader/PhantomPDF. TLS slots are allocated for each DLL or plugin that is loaded by invoking the Windows `LoadLibrary` function. `LoadLibrary` will fail if all of the TLS slots for a given process are filled. Additional information about TLS can be found at http://msdn.microsoft.com/en-us/library/ms686749(VS.85).aspx

Foxit Reader/PhantomPDF Core API

The Foxit Reader/PhantomPDF Core API encapsulates resources (e.g. user interface, basic libraries, etc.) into three separate layers: Foxit Support layer, Foxit Portable Document layer, and Foxit Reader/PhantomPDF layer. Each layer provides method calls for developers to operate on different objects within a PDF document. All of these objects are opaque data types. The Core API is implemented as a standard ANSI C programming library and is supported on Windows 32 bit platforms. The diagram below shows how the three layers of the Core API fit together.



Foxit Reader/PhantomPDF Layer

User interface customizations are done at the Foxit Reader/PhantomPDF (FRD) layer. The

FRD layer encapsulates the user interface objects used in Foxit Reader/PhantomPDF (e.g. `FR_MenuBar`, `FR_Menu`, `FR_PageView`, `FR_ToolBar`). Here are some examples of application-level tasks that can be done with the FRD layer,

- Add menus, menu commands, and toolbar buttons.
- Open and close files.
- Display simple dialog boxes.

Foxit Portable Document Layer

Modifications to PDF documents are handled in the Foxit Portable Document (FPD) layer. The FPD layer encapsulates many of the PDF objects used in Foxit Reader/PhantomPDF (e.g. `FPD_Document`, `FPD_Page`, `FPD_Annot`, `FPD_Dictionary`). Here are some examples of PDF modifications that can be done with the FPD layer,

- Create PDF documents.
- Insert pages into an existing PDF document.
- Add annotations.

Plugins can modify almost all of the data inside a PDF file since the FPD layer provides access to this content.

Foxit Support layer

Data management is handled by the Foxit Support (FS) layer. The FS layer provides platform-independent data types and methods that support the FRD and FDP layers. The FS layer encapsulates the basic common objects used by the other two layers (e.g. `FS_Rect`, `FS_AffineMatrix`, `FS_PtrArray`, `FS_ByteArray`, `FS_ByteString`).

"Hello World" Plugin

In order to cover the basics of plugin development, this manual uses three example demo projects. The "Starter" demo is a functional skeleton for all plugin applications. The "Menu" demo adds a menu item to the functional skeleton. The "Document" demo, makes use of the menu item example and displays "Hello World" on the document. Developers are encouraged to use the demo projects as a base for Foxit Reader/PhantomPDF plugins. Please download the demo projects and have them available for viewing while reading through this guide. This is required for a complete understanding of this section since it makes multiple references to the demo code. The demo projects are available for download in Visual C++ 6.0 (.dsw) format, and can be used with any version of Visual C++ 6.0 and above.

For help with using Integrated Development Environments (IDEs) other than Visual Studio for compiling these demos, please contact support@foxitsoftware.com

Visual Studio Project Settings

In order to properly develop and debug any Foxit Reader/PhantomPDF plugin, Visual Studio must be configured correctly. Below are the default Visual C++ 6.0 project settings used in the demos.

- Open the "Project Settings" dialog box for the demo project by going to **Project > Settings**
- Click on the **C/C++ tab** and select the **Preprocessor category**.
- Replace any "_MBCS preprocessor" definition with "UNICODE", "_UNICODE".
- Click on the **Link tab** and select the **General category**.
- In the **Output File Name** text field, enter the full path to your plugin. The plugin path must match the installation directory of Foxit Reader/PhantomPDF on your system. The default value is "\Program Files\Foxit Software\Foxit Reader(\Foxit PhantomPDF)\plugins". The extension is ".fpi".
- Click on the **Debug tab** and select the **General category**.
- In the **Executable for debug session** text field enter the full path to the Foxit Reader.exe/Foxit PhantomPDF.exe that will load the plugins upon startup. Foxit Reader.exe/Foxit PhantomPDF.exe must be located at the same directory level as the plugins folder.
- Click **OK** to apply the new changes and exit the "Project Settings" dialog.
- Go to **Debug > Rebuild-All**.
- It is now possible to add breakpoints and debug the plugin project like a normal application. Visual Studio will launch Foxit Reader/PhantomPDF when a debug session begins.

Please note that there are special considerations if the plugin DLL needs to link dynamically with MFC DLLs, which can be found in the MFC section.

Compilation Errors with IDEs Older than Visual C++ 6.0

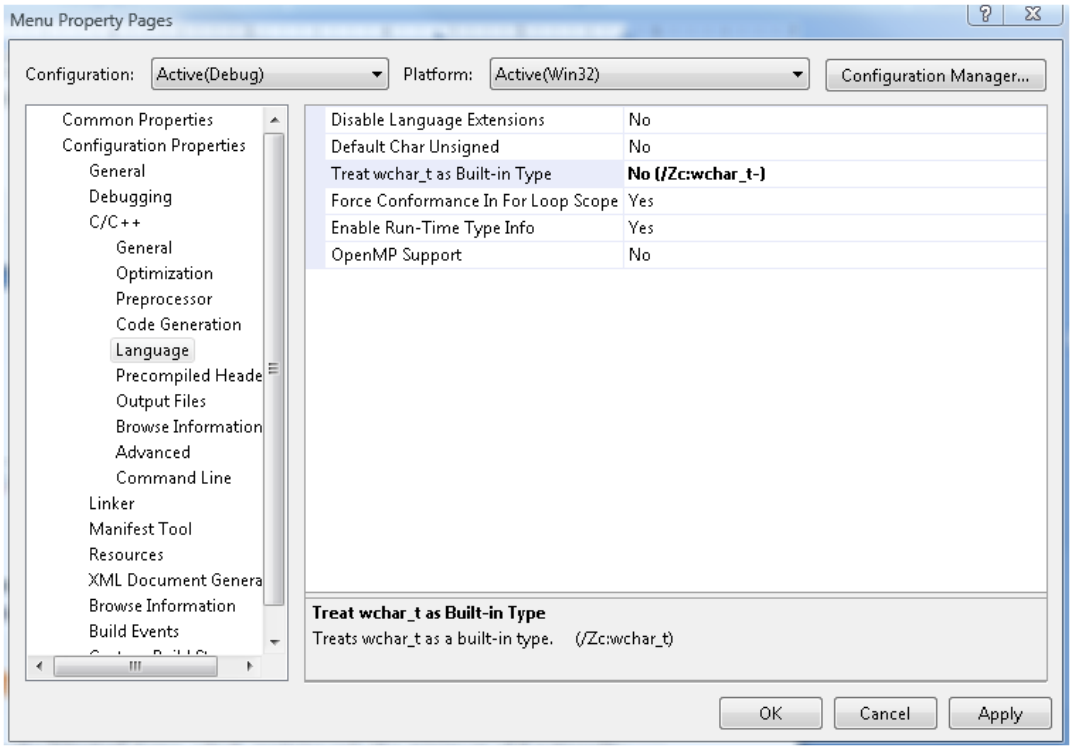
Integrated Development Environments (IDEs) later than Microsoft Visual C++ 6.0 may produce the following compilation errors:

	Description	File	Line	Column	Project
1	error C2664: 'FR_MenuItem (FS_LPCSTR,FS_LPCWSTR,FS_DIBitmap,FS_BOOL,FR_Menu)' : cannot convert parameter 2 from 'const wchar_t [10]' to 'FS_LPCWSTR'	menu.cpp	181		Menu
2	error C2664: 'FR_MenuItem (FS_LPCSTR,FS_LPCWSTR,FS_DIBitmap,FS_BOOL,FR_Menu)' : cannot convert parameter 1 from 'FS_BOOL' to 'FS_LPCSTR'	menu.cpp	181		Menu
3	error C2664: 'FS_BOOL (FR_MenuItem,FS_LPCWSTR)' : cannot convert parameter 2 from 'const wchar_t [18]' to 'FS_LPCWSTR'	menu.cpp	184		Menu

Compilations errors from Visual Studio 2008

To fix these errors, you'll need to change the compilation options such that **Treat wchar_t as a Built-in Type** is set to **No**. Here are instructions on how to configure this in Visual Studio 2008,

Go to Project > [project name] properties > Configuration Properites > C/C++ > Language > and set Treat wchar_t as Built-in Type to No (/Zc:wchar_t-).



Setting Treat wchar_t as Built-in Type in Visual Studio 2008

Getting Started with the Demo Projects

This section covers the "Starter" demo, which contains the minimum functionality each plugin must implement in order to communicate with Foxit Reader/PhantomPDF. The main files for this demo a "Starter.cpp" and "PIMain.cpp". All of the other files are auto generated during the Visual C++ project creation process. Much of the code in "Starter.cpp" and "PIMain.cpp" is also auto generated, and will be familiar to MFC application developers.

The relevant section of code starts with the compiler directive `extern "C" {…}`. Note that this directive encapsulates several functions, which are listed below.

- `PlugInMain`
- `PISetupSDK`
- `PIHandshake`
- `PIExportHFTs`
- `PIImportReplaceAndRegister`
- `PIInit`
- `PIUnload`

These functions make up a "C style" interface (hence the compiler directive) that Foxit Reader/PhantomPDF uses during plugin initialization. Every plugin must maintain this interface. The initialization procedure, or "handshaking", is handled by the `PlugInMain` function, which is the main entry point of plugin DLLs. The following functions in the "Starter" demo have default implementations that provide Foxit Reader/PhantomPDF with pointers to user defined plugin specific functions,

- `PlugInMain`
- `PISetupSDK`
- `PIHandshake`

It is the developer's responsibility to add custom application logic to these plugin specific functions. Since the "Starter" plugin serves as a skeleton plugin application, these functions are left blank. For example, if the "Starter" demo allocated any memory, it would need to release that memory when the user closes Foxit Reader/PhantomPDF. When Foxit Reader/PhantomPDF is closed, the `PIUnload()` function is invoked. The developer would add code in `PIUnload()` to de-allocate the memory.

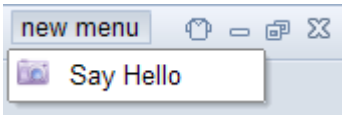
Plug-ins must define a pointer reference to Foxit Reader/PhantomPDF's Core HFT Manager and a pointer to receive the plug-in unique ID. Here is the line of code from Starter.cpp that defines the CORE HFT Manager pointer and the unique ID pointer.

```
FRCoreHFTMgr *_gpCoreHFTMgr = NULL;

void* _gPID = NULL;
```

Adding Menu Options to the Demo

The "Menu" demo covers how to add a new item to an existing Foxit Reader/PhantomPDF menu. It also shows how to add an entirely new menu to Foxit Reader/PhantomPDF.



The functional plugin menu created by the "Menu" project.

Note that "Menu.cpp" starts with the skeleton application provided by "Starter.cpp" and adds application specific logic to functions that were left blank by default. The interface used by Foxit Reader/PhantomPDF to initialize the plugin is the same with the exception of the `PIInit` function. This function now implements user defined functionality upon startup by making calls to `AddMenuItemToExistingMenu` and `AddMenuToMenuBar`.

These two functions both work in a similar way.

1. Retrieve the current Foxit Reader/PhantomPDF menu-bar.
2. Retrieve an index, and then select the menu option at that index (add to existing menu only)
3. Create a new menu item and add an icon if desired, using Plugin SDK defined types.
4. Set menu item attributes such as Title and Description.
5. Set the menu item callbacks to user defined functions (Similar to a message map).
6. Add the menu item to Foxit Reader/PhantomPDF.

These operations are handled by menu specific API functions provided by the Foxit Plugin SDK (e.g. `FRMenuItemSetExecuteProc`). The developer is then left to create the callback function for that particular menu item. Here are the callback functions from the demo,

- `OnClick`
- `IsCheck`
- `IsEnable`

The OnClick callback displays a message box to the user. The `IsCheck` and `IsEnable` are optional callbacks that change the user's view of the menu item.

The last user defined function in the "Menu" demo is `GetBMPFromRes`.This function implements a plugin specific method for retrieving a menu item's image from a PDF resource. This allows the SDK user to add images in a non-system specific way. This means that menu item images do not require full .ico files.

For additional information on the different menu options available through the Foxit Plugin SDK, please refer to the "Menus" section under Additional Demos.

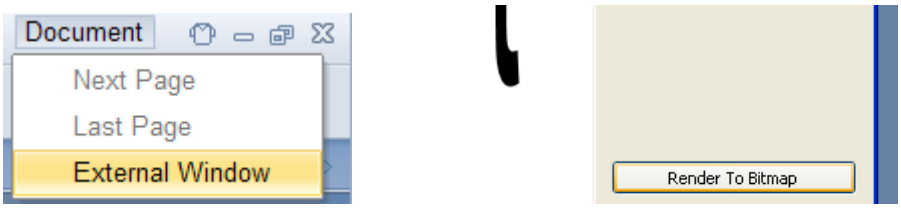
Menus that work directly with PDF documents

The "Document" demo covers PDF operations that allow a developer to re-render PDF pages. This section focuses on creating a "snapshot" or bitmap image file from a PDF page.

Please note that the relevant parts of "Document.cpp" are nearly identical to "Menu.cpp", except that the menu names differ. In the interest of clarity, the similarities between menu callbacks are illustrated below.

"Menu" Project	"Document" Project
`void OnClick`	`void ExternalWindowExecuteProcProc`
`BOOL IsCheck`	`BOOL ExternalWindowComputeMarkedProc`
`BOOL IsEnable`	`BOOL ExternalWindowComputeEnabledProc`

This section is covers the "Document->External Window->Render to Bitmap" menu option.



The "External Window" and "Render to Bitmap" options are only enabled after a document is opened.

When the `ExternalWindowExecuteProcProc` callback is fired, a Plugin SDK API function is invoked to fetch the open document information from Foxit Reader/PhantomPDF.

```
FR_Document frDoc = FRAppGetActiveDocOfPDDoc();
FPD_Document pdfDoc = FRDocGetPDDoc(frDoc);
```

Once the document information is retrieved, the callback launches a modal dialog. This modal dialog is a standard MFC dialog class that is described in "CDisplayPDFDlg.cpp". The dialog class re-renders the PDF in the modal dialog window and allows the user to perform standard reading actions, much like a "mini" Reader/PhantomPDF. The function we are interested in is saving a bitmap image of the current PDF page. This function is presented to the user as a single button that is mapped to an event handler through the message map. The event handler `OnButtonRenderToBMP` then calls `RenderPDFToBitmap`. The `RenderPDFToBitmap` function contains application logic that is common to many operations that can be done to PDF documents using the Plugin SDK. Here's a summary of the steps found in that function,

1. Create a memory bitmap (type: `FS_DIBitmap`) and associate it with a device context.
2. Set up definitions (read: pointers) needed for document operations. This includes things like `FPD_Page` and `FPD_AnnotList`.

3. Retrieve memory locations into previously defined pointers using Foxit Plugin SDK functions like `FPD_DocGetPage`.
4. Render the page and annotations to the previously defined device context with `FPDRenderContextRender`. Note that this function fills the memory bitmap associated with the device context.
5. Pass the now filled memory bitmap to the `SaveAsBmp` function.

Any PDF operations other than rendering can replace step 5, but will require repeating steps 1-4 at a later time to reflect document changes.

The `FS_DIBitmap` is a typedef for a standard Device Independent Bitmap. The `SaveAsBmp` function uses MFC and Windows standard methods to retrieve the size of the `BITMAPINFOHEADER` and `BITMAPFILEHEADER`. These values, which are a part of the DIB, are packaged along with the image data stream into a *.BMP file. The resulting output is a "snapshot" of the current PDF page.

Other PDF document operations can be implemented in a similar fashion, repeating steps 1 through 4 for proper operations. For a complete listing of all available public application and document callbacks, please see the fr_appExpT.h, fpd_docExpT.h, and fpd_renderExpT.h files.

Plugin Basics

Methods Every Plugin Must Have

A plugin must contain a source file that defines the following methods:

Method	Runtime Functionality
PlugInMain	Main entry point for the plugin.
PISetupSDK	Called by the host application to set up the plugin's SDK-provided functionality.
PIHandshake	This routine provides the initial interface between a plugin and the application. It also provides the callback functions to the application that allow it to register the plugin with the application environment.
PIExportHFTs	An extension HFT allows plugins to invoke methods that belong to other plugins. After Foxit Reader/PhantomPDF finishes handshaking with all of the plugins, it invokes each plugin's PIExportHFT callback procedure. Extension HFTs are created and added in the PIExportHFT procedure. Once the extension HFT is added to the extension HFT manager, its methods are available to other plugins. (see Extension HFTs for more information). This callback should only be used for exporting an extension HFT. It should not be used to invoke other Foxit Reader/PhantomPDF Core API methods.
PIImportReplaceAndRegister	This function allows you to, Import extension HFTs. Replace functions in existing HFTs. Register to receive notification events.
PIInit	This is the main initialization method where plugins execute steps to hook into Foxit Reader/PhantomPDF's user interface. This allows the developer to customize the user interface (e.g. add menu items, toolbar buttons, windows, etc). It is also possible to modify Foxit Reader/PhantomPDF's user interface while the plugin is running.

These methods are invoked when Foxit Reader/PhantomPDF attempts to load a plugin. Please refer to "Starter.cpp" in the "Starter" sample project for an example source file that defines all of these methods. The core plugin functionality is defined in the `PIInit` method. It is inside `PIInit` that Core API or extension API function calls can be made.

Plugin Loading

When Foxit Reader/PhantomPDF starts, it scans this directory for plugins and loads them. Foxit Reader/PhantomPDF plugins must be placed in the following directory,

```
\Program Files\Foxit Software\Foxit Reader\plugins
```

Foxit Reader/PhantomPDF must be restarted in order for the plug-ins to take effect.

The `PlugInMain` method invokes four other methods during plugin initialization,

```
PISetupSDK
PIHandshake
PIExportHFTs
PIInit
```

If any of these steps fail or Foxit Reader/PhantomPDF exits, the `PIUnload` routine is invoked. Memory used by the plugin can be de-allocated in `PIUnload`.

Handshaking

During the plugin initialization, the `PIHandshake` routine is invoked to perform a handshake with the host application. Handshaking does four things:

- Complete exporting extension HFTs that can be invoked by other plugins.
- Import extension HFTs
- Invoke all of the HFTs.
- Free allocated memory.

These four methods must be implemented for the handshake process:

```
FS_BOOL PIExportHFTs (void);
FS_BOOL PIImportReplaceAndRegister(void);
FS_BOOL PIInit (void);
FS_BOOL PIUnload (void);
```

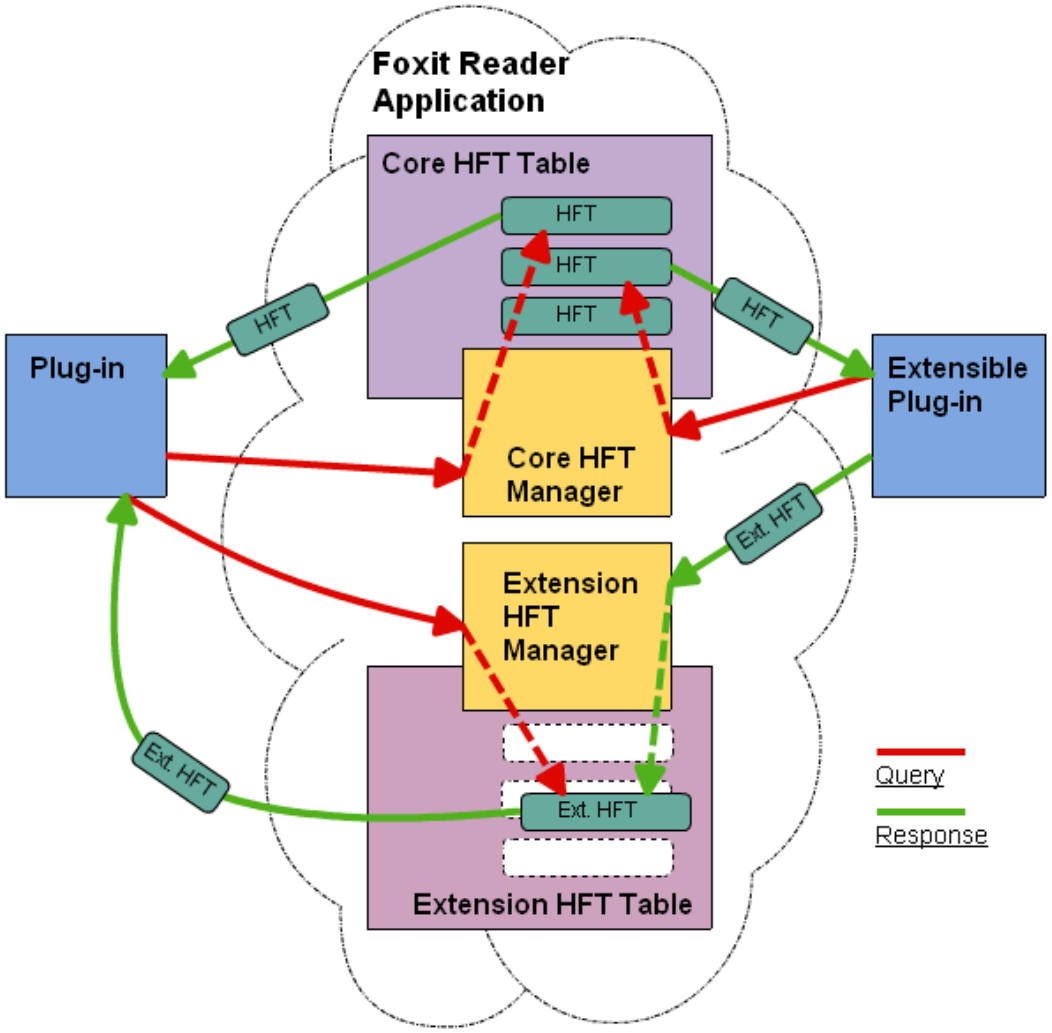
If any of these methods are not implemented, `PIHandshake` will return false and Foxit Reader/PhantomPDF will fail to load the plugin.

Host Function Tables (HFT)

A Host Function Table (HFT) is a mechanism for managing the Core API methods. It is implemented as a pointer array that stores the addresses of Core API methods. The methods are grouped together based on the types of objects they are associated with. Each group of methods has a specific HFT for performing actions on a specific object type. All of these HFTs are managed by the Core HFT manager. The manager indexes the HFTs by category. Foxit Reader/PhantomPDF consists of numerous internal HFTs that provide plugins with an efficient way to invoke their methods. Here is a high level summary of the HFT API method search algorithm,

1. The Core HFT Manager uses a category selector to locate the specific HFT.
2. The manager then uses the method selector to locate the specific method.

In addition to invoking Core API methods, extension HFTs can be created for individual plugins. Extension HFTs allow the methods of a specific plugin to be accessible to all other plugins. The following is a diagram of the HFT mechanism.



Global Core HFT Manager

User-defined plugins must contain a global Core HFT Manager pointer. If this pointer is not defined, the plugin will fail to compile. This pointer is defined in the `PISetupSDK` method. `PISetupSDK` is called by the host application to initialize the plugin.

```
/*Core HFT Manager.*/
FRCoreHFTMgr *_gpCoreHFTMgr = NULL;

FS_BOOL PISetupSDK(FS_INT32 handshakeVersion, void *sdkData)
{
    if(handshakeVersion != HANDSHAKE_V0100) return FALSE;
    PISDKData_V0100 *pSDKData = (PISDKData_V0100*)sdkData;

   /* Points to core HFT manage from Foxit Reader/PhantomPDF */
   _gpCoreHFTMgr = pSDKData->PISDGetCoreHFT();

   /* Set the plugin's handshake routine, which is called next by the host
   application */
   pSDKData->PISDSetHandshakeProc(sdkData, &PIHandshake);
   return TRUE;
}
```

Extension HFTs

Extension HFTs are mechanisms that allow plugins to invoke other plugins. The invoked plugin must create its own extension HFT. This section will cover extension HFTs in more detail. To make a plugins' set of methods accessible to other plugins, an extension HFT should be created to manage these methods. Create the extension HFT by invoking,

`FSExtensionHFTMgrNewHFT`

Add the new extension HFT to the host environment by invoking,

`FSExtensionHFTMgrAddHFT`

The following steps show the entire process. Steps 2 through 4 must be implemented in the plugin basic routine `PIExportHFTs`.

1. Define a group of methods.

```
void Function1();
void Function2();
```

2. Create a new extension HFT

```
FS_HFT extensionHFT = FSExtensionHFTMgrNewHFT(2);
// In step 1, we defined 2 methods, so the value "2" is passed to the function
// to indicate the capacity of the new HFT.
```

3. Add the HFT to the host environment.

```
FSExtensionHFTMgrAddHFT("name", VERSION, extensionHFT);
```

4. Add the address of the methods to the extension HFT.

```
FSExtensionHFTMgrReplaceEntry(extensionHFT, 0, &Function1);
FSExtensionHFTMgrReplaceEntry(extensionHFT, 1, &Function2);
```

5. From another plugin, you can access the method and invoke it in the `PIImportReplaceAndRegister` routine.

```
typedef void (*FunctionPROTO)();
extensionHFT = FSExtensionHFTMgrGetHFT("name", VERSION);
FunctionPROTO pFunction1 = FSExtensionHFTMgrGetEntry(extensionHFT, 0);
pFunction1();
```

For a detailed example, please refer to the sample project "Extension HFT". In this sample project, the methodsTmpl.h header file is a template that describes groups of methods. It is used to generate the method selector that manages the indices of the methods. It also generates the method prototypes.

The methodsCall.h header file contains examples of how to,

- Generate a method selector.
- Generate prototypes for the plugin methods.

Notice that the methods are defined as macros making it easier for other plugins to invoke them. Here are the steps for referencing these macros from other plugins,

- Include the methodsCalls.h header file.
- Invoke the FSExtensionHFTMgrGetHFT to get the HFT by name.
- Reference the macros defined in methodCalls.h

Header Files

Foxit PhantomPDF Plug-in SDK header files must be included in your plugin project. You can find the header files in the following directory:

Foxit PhantomPDF Plug-in SDK\PluginSupport\Headers

The following table lists the SDK header files and gives a simple description. In general, including fr_callsInclude.h and fs_pidata.h header files is enough.

Header file	Description
fr_callsInclude.h	Includes all the xxxCalls.h header files, which define names for referencing Foxit Reader/PhantomPDF Core APIs via the corresponding HFTs. Include this file in your plugin.
fr_common.h	Defines Foxit Reader/PhantomPDF SDK version, core HFT manager which manages referencing Foxit Reader/PhantomPDF Core APIs.
fs_pidata.h	Defines data structure, types, and other things, which are used to build a handshake routine. This file is shared between Foxit Reader/PhantomPDF and plugins.
xxxExpt.h	Contains Types, macros, and structures that are required to use the Host Function Tables.
xxxCalls.h	Defines names for referencing Foxit Reader/PhantomPDF Core APIs via the corresponding HFTs
xxxTempl.h	Catalogs of functions exported.

Core API Terminology

Objects

All the Core API objects are defined as a pointer that represents an internal real object. Objects are obtained by Core API methods. Internal objects are opaque so objects' data cannot be directly accessed. Manipulation of objects is achieved by calling corresponding Core API methods. Objects are passed by reference (vs. passed by value).

Objects names are typically defined in the following structure:

<layer>_<name> (Example: *FPD_Document*)

Layer: identifies the Core API layer (*FPD* = Foxit Portable Document layer)

Name: object's name.

Methods

Most Core API method names are typically defined in the following structure:

<layer><object><action><thing> (example: *FPDDocGetUserPermissions*)

layer: identifies the Core API layer (*FPD* = Foxit Portable Document layer)

object: identifies the object upon which the method acts (*Doc*)

action: specifies an action that the method performs (*Get*)

thing: specific to each method. (*UserPermissions*) May not always be present.

Additional Demos

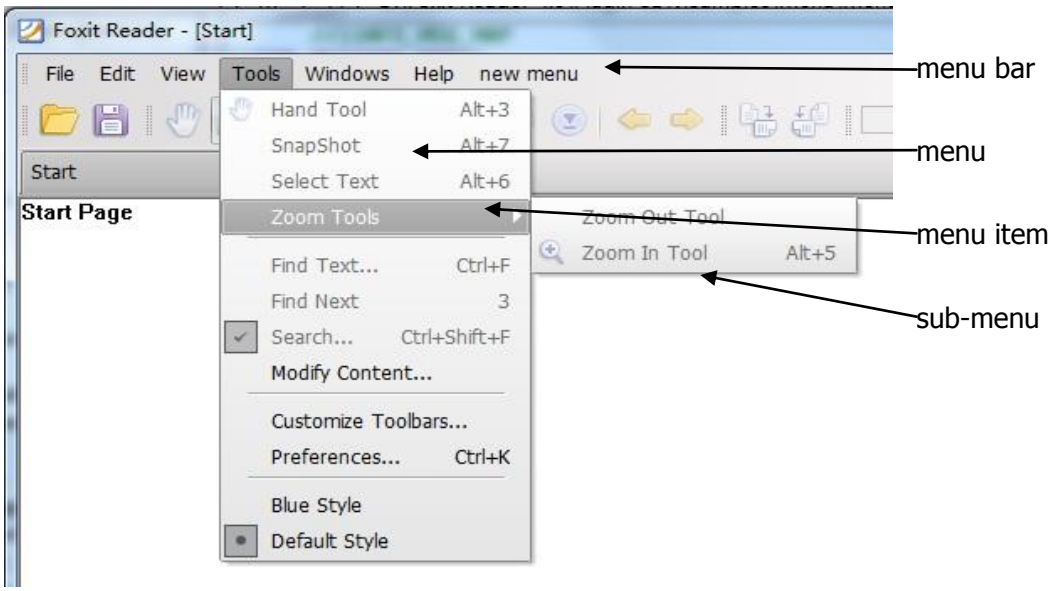
This section provides a detailed introduction to plugin development involving these Foxit Reader/PhantomPDF components,

- menus
- toolbars
- documents
- navigation panels
- event notifications

It describes the individual objects that make up each of these components and the API methods that are available for modifying them. Some of the material is an expanded explanation of topics discussed in the "Hello World" Plugin section. The Core API provides a number of structures and functions for accessing the Foxit Reader/PhantomPDF components and their individual objects (see the *Foxit Reader/PhantomPDF PDF API Reference* for a complete list).

Menus

Menu Objects



The table below lists the API structures and functions that are used to access the menu objects in the picture above.

Structure	Menu Object
FR_MenuBar	Represents the Foxit Reader/PhantomPDF menu bar. Invoke `FRAppGetMenuBar` to access the individual objects within the menu bar.
FR_Menu	Represents the individual menus contained within the menu bar. Invoke `FRMenuBarGetMenuByIndex` to gain access to an existing menu. To create a new menu, invoke the `FRMenuNew` function and add it to the menu bar.
FR_MenuItem	Represents the menu items within a menu. Menu items can also function as separators that partition other menu items. Sub-menu items are considered `FR_Menu` structures.

Menu Item Callback Functions

When creating a custom menu item with the Foxit Reader/PhantomPDF Core API, the developer must also implement the application logic in the menu item callback function. A menu item can invoke three different callback functions through the Core API,

```
typedef void (*FRExecuteProc)(void *clientData);
typedef FS_BOOL (*FRComputeEnabledProc)(void *clientData);
typedef FS_BOOL (*FRComputeMarkedProc)(void *clientData);
```

When a menu item is selected, the `FRExecuteProc` callback is invoked. It is the developer's responsibility to implement application logic for this callback function.

The developer can control whether a menu item is enabled by implementing the `FRComputeEnabledProc` callback. Set this callback function to return true to enable the menu item and false to disable it.

For menu items that have check marks associated with them, the developer can control whether it is checked by implementing the `FRComputeMarkedProc` callback. Set this callback function to return true to check the menu item and false to uncheck it.

Menu Sample Project

The "Menu" sample project is a working example of the steps covered in the following sections. Please refer to it for additional information.

How to Add a Menu Item to an Existing Menu

1. Get the Foxit Reader/PhantomPDF menu bar.

```
FR_MenuBar menuBar = FRAppGetMenuBar();
```

2. Get the last menu in the menu bar.

```
int nCount = FRMenuBarGetMenuCount(menuBar);
FR_Menu menu = FRMenuBarGetMenuByIndex(menuBar, nCount - 1);
```

3. Get the menu item icon.

```
FS_DIBitmap bitmap = GetBmpFromRes(IDR_SAYHELLOICON);
```

4. Create a new menu item.

```
FR_MenuItem menuItem = FRMenuItemNew("Say Hello", L"Say Hello",
                                     bitmap, false, NULL);
```

5. Set the tool tip to the menu item.

```
FRMenuItemSetToolTip(menuItem, L"Say Hello Tooltip");
```

6. Set the description to the menu item.

```
FRMenuItemSetDescribeText(menuItem, L"Say Hello Description");
```

7. Set the execute callback to the menu item.

```
FRMenuItemSetExecutePro(menuItem, &OnClick);
```

8. Set the compute-enabled callback to the menu item.

```
FRMenuItemSetComputeEnabledPro(menuItem, &IsEnable);
```

9. Set the compute-marked callback to the menu item.

```
FRMenuItemSetComputeMarkedProc(menuItem, &IsCheck);
```

10. Add the new menu item to the end of the menu.

```
int nMenuItemCount = FRMenuGetMenuItemCount(menu);
FRMenuAddMenuItem(menu, menuItem, nMenuItemCount);
```

How to Add a New Menu to the Menu Bar

1. Get the Foxit Reader/PhantomPDF menu bar.

```
FR_MenuBar menuBar = FRAppGetMenuBar();
```

2. Create a new menu.

```
FR_Menu menu = FRMenuNew(NULL);
```

3. Add a menu to the end of the menu bar.

```
int nCount = FRMenuBarGetMenuCount(menuBar);
FRMenuBarAddMenu(menuBar, menu, L"new menu", "new menu", nCount);
```

How to Get the Name Associated to Menu Item

All the menus and menu items of Foxit Reader/PhantomPDF use a name attribute as their unique identifier. To get the name of a specified menu item, you should retrieve all the menus or all the menu items if needed.

The following code is an example:

```
// Retrieve the menu bar in Foxit Reader/PhantomPDF
FR_MenuBar menuBar = FRAppGetMenuBar();
if(NULL == menuBar) return;

// Retrieve all the menus in menu bar
FS_INT32 nMenu = FRMenuBarGetMenuCount();
for(FS_INT32 i=0; i<nMenu; i++)
{
    FR_Menu menu = FRMenuBarGetMenuByIndex(menuBar, i);
    FR_MenuItem menuItem = FRMenuGetParentMenuItem(menu);
    FS_ByteString bsMenuName = FSByteStringNew();
    FRMenuItemGetName(menuItem, &bsMenuName);
    //Now, you get the menu name, save it if it is you want.
    ...
    ...
    //end save
    FSByteStringDestroy(bsMenuName);

  // Retrieve all the menu items of the menu
    FS_INT32 nItem = FRMenuGetMenuItemCount(menu);
    for(FS_INT32 n=0; n<nItem; n++)
    {
        FR_MenuItem mItem = FRMenuGetMenuItemByIndex(menu, n);
        FS_ByteString bsItemName = FSByteStringNew();
        FRMenuItemGetName(nItem, &bsItemName);
        //Now, you get the menuItem name, save it if need.
        ...
        ...
        //end save
        FSByteStringDestroy(bsItemName);
```

```
        // Retrieve all the submenu items of the menu item
         FR_Menu subMenu = FRMenuItemGetSubMenu(mItem);
         if(NULL != subMenu)
         {
         //continue looking up
         }


    }

}
```

Tool Bar

Tool Bar Objects

The table below lists the API objects and functions that are used to access the tool bar objects in the picture above.

Structure	Tool Bar Object
FR_ToolBar	Represents a Foxit Reader/PhantomPDF tool bar. Use `FRAppGetToolBarByName` or `FRAppGetToolBarByIndex` to access specific tool bars. Use `FRToolBarNew` to create a new tool bar.
FR_ToolButton	Represents the individual buttons that make up a tool bar. Buttons can also function as separators that partition other buttons. Buttons have many attributes that can be set through the API (e.g. name, sub-menu, icon, callback function, etc).

Tool Button Callback Functions

Similar to custom menu items, tool bar buttons require the developer to implement the application logic in the tool bar button callback function. A tool bar button can invoke three different callback functions through the Core API,

```
typedef FS_BOOL (*FRExecuteProc)(void *clientData);
typedef void (*FRComputeEnabledProc)(void *clientData);
typedef FS_BOOL (*FRComputeMarkedProc)(void *clientData);
```

When a user clicks on a tool button, the `FRExecuteProc` callback is invoked. It is the developer's responsibility to implement application logic for this callback function.

The developer can control whether a tool button is enabled by implementing the `FRComputeEnabledProc` callback. Set this callback function to return true to enable the menu item and false to disable it.

For tool buttons that have check marks associated with them, the developer can control whether it is checked by implementing the `FRComputeMarkedProc` callback. Set this callback function to return true to check the menu item and false to uncheck it.

Toolbar Sample Project

The "Toolbar" sample project is a working example of the steps covered in the following section. Please refer to it for additional information.

How to Create a New Tool Bar

This section covers how to create a new tool bar and add a new button to the tool bar.

1. Create a new tool bar.
```
FR_ToolBar toolBar = FRToolBarNew("new tool bar", L"new tool bar", FALSE);
```

2. Create a new tool button.
```
FR_ToolButton toolButton = FRToolButtonNew("new tool button", FALSE);
```

3. Get the tool button icon.
```
FS_DIBitmap bitmap = GetBmpFromRes(IDR_PDF_TOOLBUTTON);
```

4. Set the icon to the tool button.
```
FRToolButtonSetIcon(toolButton, bitmap, NULL);
```

5. Set callbacks to the tool button.

```
FRToolButtonSetExcuteProc(toolButton, &OnClick);
FRToolButtonSetEnableProc(toolButton, &IsEnable);
FRToolButtonSetCheckProc(toolButton, &IsCheck);
```

6. Add the tool button to the tool bar.

```
FRToolBarAddButton(toolBar, toolButton);
```

Documents

This section will cover the main document structures provided by the Core API for manipulating PDF documents.

PDF Document Model

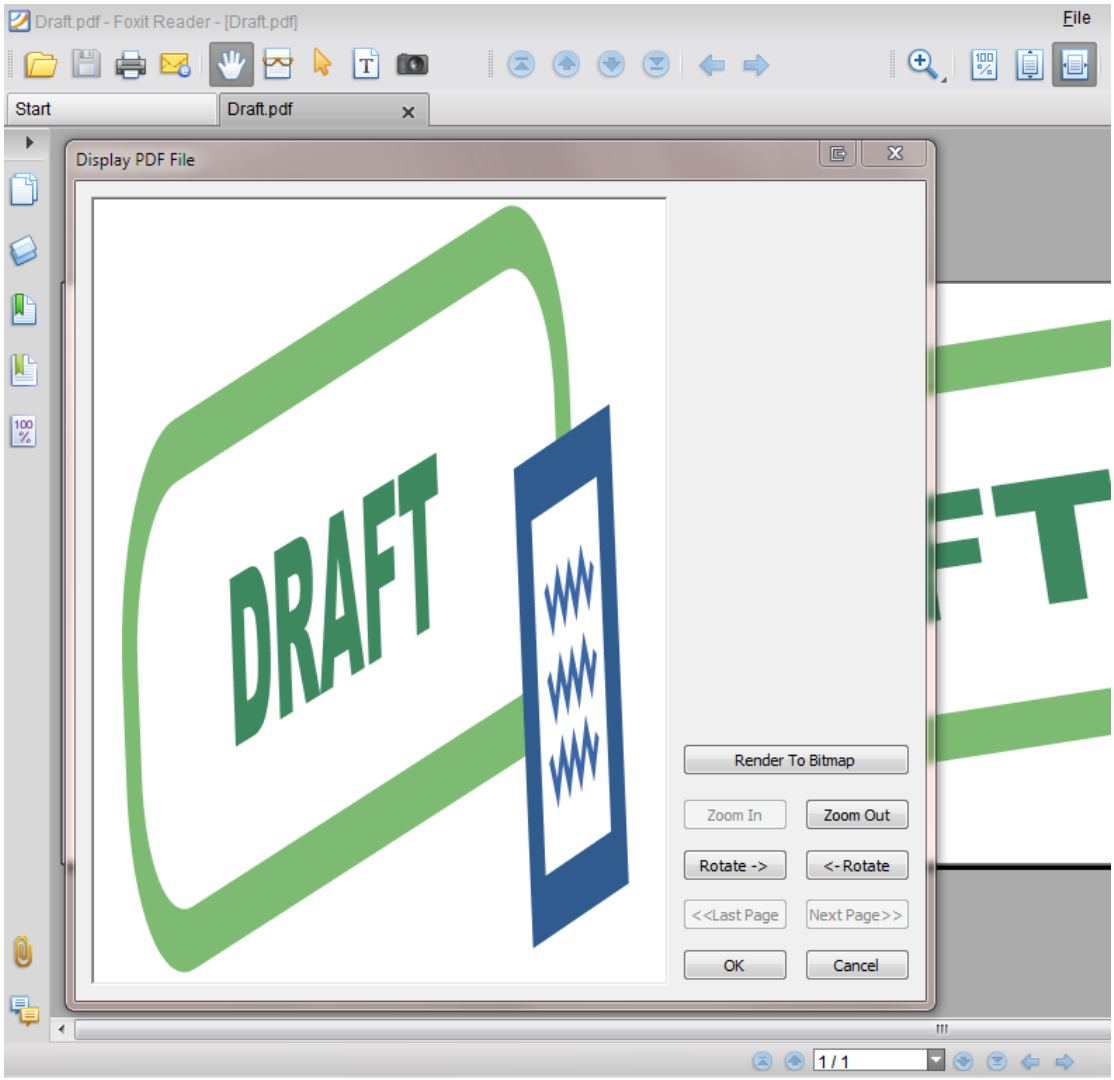
Structure	Description
`FPD_Document`	Primary document structure that represents the objects that make up a PDF document. Provides access to all of the objects contained within a PDF (e.g. tree of pages, trees of bookmarks, articles, information and security dictionaries, etc). These objects can be accessed through the FPD layer of the Core API.
`FPD_Page`	Provides access to a tree of pages.
`FPDDocGetPage`	Provides access to PDF pages within a document.
`FPDDocGetInfo`	Provides access to information dictionaries within a document.

PDF Document View

Structure	Description
`FR_Document`	Primary document structure that represents the view of a PDF document. The view is responsible for storing state information as changes are made to the document. Here are a few examples of state information that is handled by the PDF document view, • Current page that is on display. • Current zoom level setting. • Enable or disable the Save button depending on whether the contents of a document were modified.

Document Sample Project

The "Document" sample project shows how a PDF document can be manipulated (e.g. zoom in, zoom out, rotate clockwise, and rotate counterclockwise) using the `FR_Document` view structure. It also provides an example of using the `FPD_Document` structure to open a PDF document in an external window. The sample also includes a button for rendering a PDF document to a bitmap.



Display a PDF document in an external window using the "Document" sample plugin.

Event Notifications

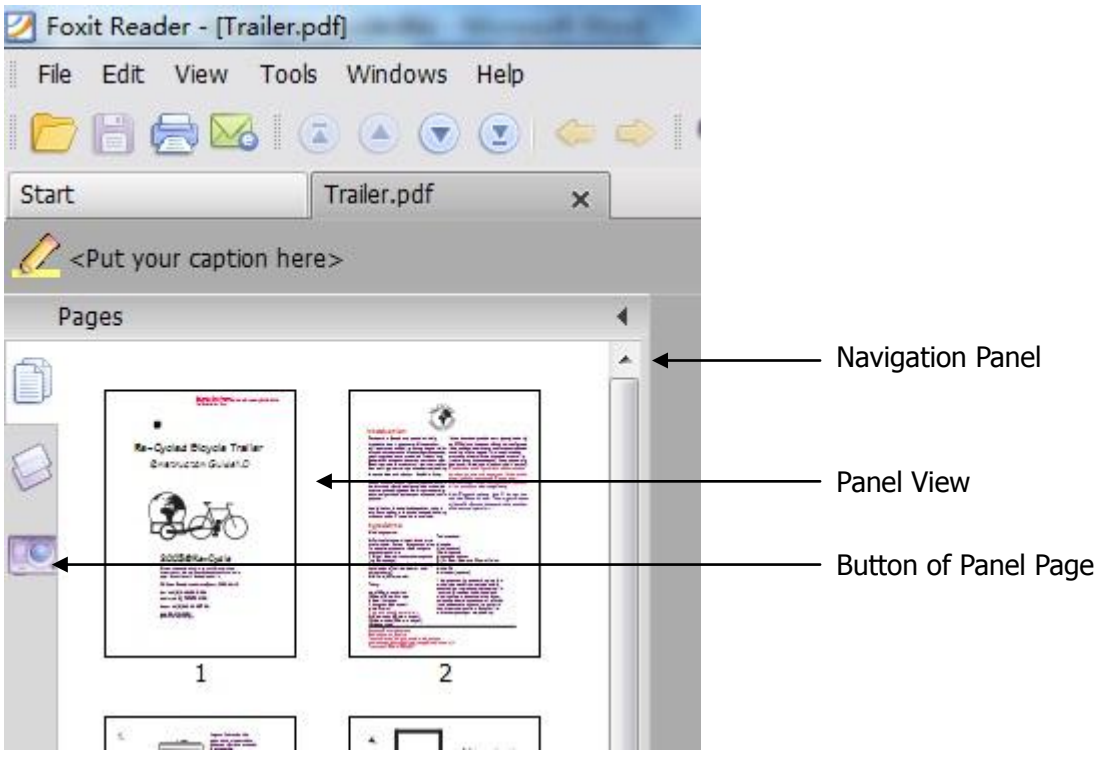
Plugins can register to receive notifications when events occur in Foxit Reader/PhantomPDF. The plug-in must provide a callback function for the notification. Callback functions are invoked by Foxit Reader/PhantomPDF when the corresponding event occurs. Developers can add custom application logic in the callback function to control how the event is handled. The Core API provides a number of functions for registering to receive event notifications. For example,

```
FRAppRegisterPreferencePageHandler
RegisterNavPanelView
RegisterAppEventHandler
RegisterDocHandlerOfPDDoc
```

The header file `fr_appCalls.h` contains a complete list of registration functions. Please refer to the "Event Notification" sample project for additional information.

Navigation Panel

Navigation Panel Objects



Navigation Panel View Sample Project

Plugins can receive notifications when panel events occur (e.g. the activation of panel view, the rotation of a page view, etc). Panel events invoke callback functions that execute application logic to perform a specific action. Developers can register for event notifications by adding custom application logic to these callback functions.

The "Navigation Panel View" sample project covers how to create a navigation panel view and shows how panel view event notifications should be handled.

Here is a high level outline of the application logic for the project,

1. Define the callback functions that will be invoked by Foxit Reader/PhantomPDF when a panel view events occur.

2. Define a record that contains the addresses of all panel view event callback functions.

3. Register to receive notifications of (navigation panel view) events when they occur. As long as there are plugins that are registered to receive notifications from callbacks stored in a particular record, that record cannot be de-allocated. Registration is done in the `PIInit` routine by invoking `FRAppRegisterNavPanelView`.

Enabling a Foxit Reader/PhantomPDF Plug-in

This section provides a detailed introduction to enabling a plug-in in Foxit Reader/PhantomPDF. The plug-in must be certified if it needs to invoke the restricted interfaces that perform editing functions.

Applying for Digital Certificate

To certify a plug-in, apply for a digital certificate.
To apply for a digital certificate, perform the steps as follows:

1. Finish development of the plug-in.

2. The plug-in must meet the criteria established by the Foxit Reader/PhantomPDF Integration Key Licensing Aggrement. Fill out agreement and send it to support@foxitsoftware.com. There is a fee involved. More information can be found at http://www.foxitsoftware.com.

   Once agreement is approved, you will receive a contract package that includes information that is unique to you.

3. Create the public/private key pair file using the KeyGen tool provided in the Foxit PhantomPDF Plug-in SDK.

4. Open the key pair file and extract the public key, then send the public key to support@foxitsoftware.com. The public key will be used to generate a certificate that will be sent back to you. Using the certificate, the plug-ins which meet the criteria established by the Foxit Reader/PhantomPDF Integration Key Licensing Agreement can be enabled.

5. The received digital certificate must be added to the plug-ins. This is covered in the following section.

   Note: If you want to release the plug-in for Foxit Reader, step 3 and 4 must be ignored. You will receive the digital certificate in step 2.

Certifying a Plug-in

Perform the following steps to certify a plug-in:

1. After receiving the digital certificate, open the dummy.rc file and the resource file of the plug-in project in a text editor. Copy the content of dummy.rc to the resource portion of your plug-in.

2. Place the digital certificate in the directory "Plug-in Project/res".

3. Rebuild your plug-in. The plug-in will be created with the digital certificate and dummy digest data, which will be replaced with real data in the next step.

4. Run PISignatureGen.exe. Input the path to the plug-in and the path to the key pair file generated above. Click on the "Generate Now" button to sign the plug-in.

Now, the plug-in can be loaded by Foxit Reader/PhantomPDF. Step 4 must be repeated each time the plug-in is rebuilt.

Note: If you want to release the plug-in for Foxit Reader, step 4 must be ignored. Instead, you must send the plug-in to support@foxitsoftware.com. Foxit Corporation will sign the plug-in for you and send it back to you.