WordpressPluginFrameworkUserManual
Implementation guide for the Wordpress Plugin Framework.

Introduction

The Wordpress Plugin Framework (WPF) is a PHP class that is used to provide a framework for the development of Wordpress plugins. The overall intention of the WPF is to generalize and simplify plugin design while also helping plugins adhere to a common administration and usage standard. This manual will help guide you in the development and implementation of a Wordpress plugin derived from the WPF base class.

> *NOTE:* This guide is written specifically for the **Wordpress Plugin Framework v0.04** release.

> *NOTE:* This latest version of this document can be found at http://code.google.com/p/wordpress-plugin-framework/w/edit/WordpressPluginFrameworkUserManual.

Getting Started

This section will provide a strategy for the creation of a plugin based from the WPF.

> *NOTE:* Throughout this document I will reference the design and development of a fictional **My Test Plugin** that resides in the **my-test-plugin** folder and is implemented as the **my-test-plugin.php** file.

Creating a Plugin Package

The first step to developing any Wordpress plugin is to create a plugin package. This package consists of a specially named folder to contain your plugin files and a specially named file that is your main plugin file.

So for our example plugin design we will need to do the following:

1. Create a **my-test-plugin** folder.
2. Create a **my-test-plugin.php** file within the **my-test-plugin** folder.

Don't worry about writing any code at this point. You will need to download and integrate the WPF before you can begin your true plugin development. Don't worry though... integrating the WPF into your plugin package will only take a few minutes.

> *NOTE:* You may want to integrate your plugin package with the Wordpress subversion repository in order to maintain proper revision control and allow for other Wordpress users to gain access to your plugin. Information on utilizing the Wordpress subversion repository is out of the scope of this document. However, more information on this topic can be found at the Wordpress Extend Add Your Plugin webpage.

Obtaining the Wordpress Plugin Framework

Now that you have created your basic plugin package you can begin integration of the WPF into your plugin package. But first, you will need to download the latest release of the WPF from the Wordpress subversion repository.

> Click Here To Download The Latest Release of the Wordpress Plugin Framework

Integrating the Wordpress Plugin Framework

Integration of the WPF into your plugin package is a fairly simple and painless process. First you will need to unzip the archive file that you retrieved from the Wordpress subversion repository. Next you will need to copy the **wordpress-plugin-framework.php** and **README.txt** files into your plugin package folder.

So for our example plugin design we will need to do the following:

1. Unzip the archive file to the **wordpress-plugin-framework** folder.
2. Copy the **wordpress-plugin-framework.php** file into the **my-test-plugin** folder.
3. Copy the **README.txt** file into the **my-test-plugin** folder.

See... I told you it would be painless... well, you aren't quite done yet. You will still need to customize a small section of the **wordpress-plugin-framework.php** file for your specific plugin.

Customizing the Wordpress Plugin Framework

The only customization required for integration of the WPF with the plugin is to modify the name of the WPF class so that it is specific to the plugin package. This is required due to the lack of namespace support in PHP 4/5.

This step is extremely important so to make sure that you get it right I will walk you through a demonstration of customizing for the **My Test Plugin** package.

So for our example plugin design we will need to do the following:

- Modify the name of the WPF base class to be specific to this plugin.

```
class MyTestPlugin_WordpressPluginFramework
```

You have now successfully customized the **wordpress-plugin-framework.php** file for your plugin. From this point forward, you will not need to modify this file unless you perform one of the following tasks.

- *Upgrade your plugin to a new release of the WPF* - You will need to follow the guidelines specified for the new release of the WPF.

Basic Plugin Implementation

You are now ready to get the ball rolling on your plugin development. However, before we can deep dive the usage of the WPF you will need to perform a few standard Wordpress plugin development steps first.

Standard Wordpress Plugin Requirements

Every Wordpress plugin is required to contain a valid Wordpress plugin header section and a valid **README.txt** file. Specifics about these two items are beyond the scope of this document. However, you may find more information about these two topics by clicking on the link below.

> Click Here To Read About Wordpress Plugin Development

Creating the Plugin Class

Before you can begin developing your plugin's class you will need to include the WPF in your plugin's file. This is easily accomplished by using the require_once() function.

So for our example plugin design we will need to do the following:

- Open the **my-test-plugin.php** file and add the following code just below the plugin header.

```
require_once( "wordpress-plugin-framework.php" );
```

Now that the WPF is included in the project you can safely begin development of the plugin class. The first step in the class development is to derive your plugin class from the WPF base class (i.e. the class that was previously customized for this plugin).

So for our example plugin design we will need to do the following:

- Add the following lines of code to the plugin file.

```
class MyTestPlugin extends MyTestPlugin_WordpressPluginFramework
{
  // Add class specific code here...
}
```

This step completes the creation of a plugin derived from the WPF base class.

Instantiating & Initializing the Plugin

Now that the derived plugin class has been created we need to do something with it. First the plugin class will need to be instantiated before it can be initialized. Once it has been instantiated it will need to be properly initialized so that the WPF will understand how to work with this specific plugin. The `Initialize()` function requires that the developer input the plugin's `$title`, `$version`, `$subfolderName`, and `$fileName` parameters.

So for our example plugin design we will need to do the following:

- Add the following lines of code to the plugin file.

```
if( !$myTestPlugin  )
{
  // Instantiate the plugin.
  $myTestPlugin = new MyTestPlugin();
  // Initialize the plugin.
  $myTestPlugin->Initialize( 'Test Plugin for the Wordpress Plugin Framework', '1.00', my-test-plugin', 'my-test-plugin' );
}
```

A full description of the `Initialize()` function is provided below.

```
/**
 * Initialize() - Initializes the standard parameter set associated with this plugin.
 *
 * This function initializes the standard parameter set associated with this plugin so that the plugin
 * may be safely integrated into the Wordpress core.
 *
 * @param string    $title              The title of the plugin.
 * @param string    $version            The version of the plugin.
 * @param string    $subfolderName      The name of the plugin subfolder installed under the root plugins directory.
 * @param string    $fileName           The name of the plugin's main file.
 *
 * @return void     None.
 *
 * @access public
 * @since {WP 2.3}
 * @author Keith Huster
 */
function Initialize( $title, $version, $subfolderName, $fileName )
{
  // Code to initialize the plugin...
}
```

Adding & Registering Plugin Options

The following sections discuss the methods used to add options (persitent data) to the plugin and properly register the plugin options with the Wordpress core.

Adding Plugin Options

Plugin options provide are simply persistent pieces of data that are stored within the Wordpress database. The option values are updates via HTML <input> objects such as textboxes, checkboxes, etc... The WPF currently only supports adding options to the main Wordpress Options database table so discussion of creating and managing database tables is beyond the scope of this document.

To add an option to the plugin you simply need to call the `AddOption()` function and provide it with the appropriate set of parameters for the specific option that is being added to the plugin.

So for this example a single HTML textbox option named `"myTextboxOption"` will be added to the `$myTestPlugin` class. This textbox will default to the text string `"Hello!"` and will be modifiable by the user. A description of the textbox option that displays `"'Simple textbox option for your plugin."` will also be provided to the user. The code to implement this option is described below.

```
$myTestPlugin->AddOption( $myTestPlugin->OPTION_TYPE_TEXTBOX, 'myTextboxOption', 'Hello!', 'Simple textbox option for your plugin.' );
```

Don't worry if the syntax for the `AddOption()` function looks confusing at this point. The following documentation for this function should help answer any questions that may be present at this time.

```
/**
 * _AddOption() - Adds an option to the plugin's options array.
 *
 * This function adds the specified option to the plugin's options array. This array can then be used to
 * manage the interface between the plugin options and the Wordpress options database.
 *
 * @param string    $optionType          Type of the option to add to the array.
 * @param string    $optionName          Name of the option to add to the array.
 * @param mixed     $optionValue         Value of the option to add to the array.
 * @param string    $optionDescription   Description of the option to add to the array.
 * @param string    $optionValuesArray   Array of selectable values for the option.
 *
 * @return void     None.
 *
 * @access public
 * @since {WP 2.3}
 * @author Keith Huster
 */
function AddOption( $optionType, $optionName, $optionValue, $optionDescription, $optionValuesArray = '' )
{
  // Code used to add the option...
}
```

The valid option types that are currently supported by the WPF are listed below.

- `OPTION_TYPE_TEXTBOX` - Single line textbox that can contain any text string.
- `OPTION_TYPE_TEXTAREA` - Multi-line textbox that can contain any text string.
- `OPTION_TYPE_CHECKBOX` - Checkbox containing the value `CHECKBOX_UNCHECKED` or `CHECKBOX_CHECKED` based on the state of the checkbox.
- `OPTION_TYPE_RADIOBUTTONS` - A group of radiobutton options (see usage note below).
- `OPTION_TYPE_PASSWORDBOX` - Single line password box that can contain any text string.
- `OPTION_TYPE_COMBOBOX` - A drop-down style box containing a set of options to select from (see usage note below).
- `OPTION_TYPE_FILEBROWSER` - Single line textbox with a "browse" button.
- `OPTION_TYPE_HIDDEN` - Hidden option type that is not displayed.

USAGE NOTE: The `OPTION_TYPE_RADIOBUTTONS` and `OPTION_TYPE_COMBOBOX` option types require a set of selectable values to be defined for the option. These options are specified as the `$optionValuesArray` input parameter. Also, the radiobuttons and combobox options will be displayed in the order that they are added to the `$optionValuesArray` parameter. The default value for these options types is specified in the standard '$optionValue' parameter of the option. An example radiobutton group containing three radiobuttons named "Radio1", "Radio2", and "Radio3" is shown below.

```
// Add a radiobuttons group containing three radiobuttons (default to "Radio2" being selected).
$myRadioButtonOptions = array( 'Radio1', 'Radio2', 'Radio3' );
$myTestPlugin->AddOption( $myTestPlugin->OPTION_TYPE_RADIOBUTTONS, 'myRadiobuttonsOption', $myRadioButtonOptions[1], 'Simple textbox option for your plugi
```

Registering Plugin Options

Once all of the plugin options have been added to the plugin it is safe to call the `RegisterOptions()` function to properly register the plugin options with the Wordpress core.

NOTE: The WPF requires that all of the plugin options must be added prior to attempting to register the plugin options with the Wordpress core.

So for our example plugin design we will need to do the following:

- Add the following lines of code to the plugin file.

```
// Register the plugin options with the Wordpress core.
$myTestPlugin->RegisterOptions( __FILE__ );
```

Once the plugin options have been successfully registered an administration interface can be added to allow the user to interact with the plugin.

Creating the Plugin Administration Interface

The following sections discuss the methods used to add an administration interface to the plugin utilizing the WPF that is accessible via the Wordpress administration dashboard.

Adding Plugin Administration Blocks

The WPF provides a standardized method of adding content to the plugin administration page. This standard method utilizes the concept of "blocks" of content to help define specialized sections of the plugin's administration page. Each of these "blocks" is displayed as a collapsible section within the plugin's administration page.

To add a block to the plugin's administration page the `AddAdministrationPageBlock()` function must be called. This function utilizes a defined set of parameters to determine the name, content, and location of the specified page block within the plugin administration page.

The following documentation is provided for the `AddAdministrationPageBlock()` function to help answer any questions that may be present at this time.

```
/**
 * AddAdministrationPageBlock() - Adds a block of content to be displayed in the plugin's administration page.
 *
 * This function adds a block of content (i.e. an instance of a dbx-box class) to the plugin's administration
 * page. The placement and size of the block is controlled by the $blockType parameter.
 *
 * @param string    $blockId            ID of the content block used in HTML formatting (no spaces allowed).
 * @param string    $blockTitle         Title of the content block.
 * @param string    $blockType          Type of content block (one of CONTENT_BLOCK_TYPE_xxx).
 * @param string    $blockFunctionPtr   Function containing the content to be displayed.
 *
 * @return void     None.
 *
 * @access public
 * @since {WP 2.3}
 * @author Keith Huster
 */
function AddAdministrationPageBlock( $blockId, $blockTitle, $blockType, $blockFunctionPtr )
```

```
{
  // Code used to add the page block...
}
```

The `$blockType` parameter is specifed to be one of the pre-defined block types listed below.

- `CONTENT_BLOCK_TYPE_MAIN` - Larger block type that displays main content information.
- `CONTENT_BLOCK_TYPE_SIDEBAR` - Sidebar information displayed to the right of the main content blocks.

The final parameter to the `AddAdministrationPageBlock()` function is a pointer to the function containing the content to be displayed within the page block. This implementation method allows the plugin developer to maintain separation between the plugin being developed and the core WPF functionality. Hence all HTML display functions within the plugin class can be isolated from the inner core functionality of the WPF.

So for this example a single page block that displays "Hello World!" will be added to the plugin administration page. The plugin class function used to contain the raw HTML code for displaying the "Hello World!" text string is defined to be `HTML_DisplayPluginHelloWorldBlock()`.

```
// Plugin class function containing the raw HTML content code.
function HTML_DisplayPluginHelloWorldBlock()
{
  ?>
  <p>Hello World!</p>
  <?php
}


// Function used to add the content block to the plugin administration page.
$myTestPlugin->AddAdministrationPageBlock( 'block-hello-world', 'Hello World', $myTestPlugin->CONTENT_BLOCK_TYPE_SIDEBAR, array($myTestPlugin, 'HTML_Displa
```

Registering the Plugin Administration Page

Once all of the content blocks have been added to the plugin administration page it is safe to call the `RegisterAdministrationPage()` function to properly register and display the plugin administration page within the Wordpress dashboard.

> *NOTE:* The WPF requires that all of the plugin page blocks must be added prior to registering the plugin administration page. Also, the blocks will be displayed in the order that they are added to the page. For proper block positioning, all "Main" blocks must be added prior to adding the "Sidebar" blocks (allows the CSS "float" property to work correctly).

The following documentation is provided for the `RegisterAdministrationPage()` function to help answer any questions that may be present at this time.

```
/**
 * RegisterAdministrationPage() - Registers the plugin's administration page.
 *
 * This function registers the plugin's administration page with the Wordpress core via the add_action()
 * hook. This hook allows the plugin's administration paege to be processed as any standard Wordpress
 * administration page (such as the dashboard).
 *
 * @param string    $parentMenu          Parent menu of the plugin's administration menu.
 * @param string    $minimumAccessLevel  Minimum user access rights required to access the plugin's administration page.
 * @param string    $adminMenuTitle      Name of the plugin's administration menu.
 * @param string    $adminMenuPageTitle  Browser title of the plugin's administration page.
 * @param string    $adminMenuPageSlug   URI slug displayed for the plugin's administration webpage.
 *
 * @return void     None.
 *
 * @access public
 * @since {WP 2.3}
 * @author Keith Huster
 */
function RegisterAdministrationPage( $parentMenu, $minimumAccessLevel, $adminMenuTitle, $adminMenuPageTitle, $adminMenuPageSlug )
{
  // Code used to register the plugin administration page...
}
```

The `$parentMenu` parameter is specifed to be one of the pre-defined menu items listed below.

- `PARENT_MENU_DASHBOARD` - The "Dashboard" parent menu item.
- `PARENT_MENU_WRITE` - The "Write" parent menu item.
- `PARENT_MENU_MANAGE` - The "Manage" parent menu item.
- `PARENT_MENU_COMMENTS` - The "Comments" parent menu item.
- `PARENT_MENU_BLOGROLL` - The "Blogroll" parent menu item.
- `PARENT_MENU_PRESENTATION` - The "Presentation" parent menu item.
- `PARENT_MENU_PLUGINS` - The "Plugins" parent menu item.
- `PARENT_MENU_USERS` - The "Users" parent menu item.
- `PARENT_MENU_OPTIONS` - The "Options" parent menu item.

The `$minimumAccessLevel` parameter is specifed to be one of the pre-defined access level items listed below.

- `RIGHTS_REQUIRED_ADMIN` - The "Administrator" access level.

So for our example plugin design we will need to add the plugin administration page to the "Plugins" parent menu and we will require "Administrator" level access to this page.

```
// Register the plugin administration page with the Wordpress dashboard core.
$myTestPlugin->RegisterAdministrationPage( $myTestPlugin->PARENT_MENU_PLUGINS, $myTestPlugin->RIGHTS_REQUIRED_ADMIN, 'My Test Plugin', 'My Test Plugin Opt
```

Once the plugin has been successfully initialized an administration interface has been added and registered the plugin can be safely integrated into a Wordpress installation.

Appendix A: Example "my-test-plugin.php" File

The following code fully implements the **My Test Plugin** plugin described throughout this document.

```php
<?php
/**
 *     ---------------       DO NOT DELETE!!!     ---------------
 *
 *     Plugin Name:  My Test Plugin
 *     Plugin URI:   http://code.google.com/p/wordpress-plugin-framework/w/edit/WordpressPluginFrameworkUserManual
 *     Description:  A simple test plugin used to demonstrate the Wordpress Plugin Framework.
 *     Version:      0.01
 *     Author:       Double Black Design
 *     Author URI:   http://www.doubleblackdesign.com
 *
 *     ---------------       DO NOT DELETE!!!     ---------------
 *
 *    This is the required license information for a Wordpress plugin.
 *
 *    Copyright 2007  Keith Huster  (email : husterk@doubleblackdesign.com)
 *
 *    This program is free software; you can redistribute it and/or modify
 *    it under the terms of the GNU General Public License as published by
 *    the Free Software Foundation; either version 2 of the License, or
 *    (at your option) any later version.
 *
 *    This program is distributed in the hope that it will be useful,
 *    but WITHOUT ANY WARRANTY; without even the implied warranty of
 *    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 *    GNU General Public License for more details.
 *
 *    You should have received a copy of the GNU General Public License
 *    along with this program; if not, write to the Free Software
 *    Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA  02110-1301  USA
 *
 *     ---------------       DO NOT DELETE!!!     ---------------
 */
/**
 * Include the WordpressPluginFramework.
 */
require_once( "wordpress-plugin-framework.php" );
/**
 * MyTestPlugin - Simple plugin class used to demonstrate the WordpressPluginFramework.
 *
 * @package my-test-plugin
 * @since {WP 2.3}
 * @author Keith Huster
 */
class MyTestPlugin extends MyTestPlugin_WordpressPluginFramework
{
  /**
   * HTML_DisplayPluginHelloWorldBlock() - Displays the "Hello World!" content block.
   *
   * This function generates the markup required to display the specified content block.
   *
   * @param void      None.
   *
   * @return void     None.
   *
   * @access private  Access via internal callback only.
   * @since {WP 2.3}
   * @author Keith Huster
   */
   function HTML_DisplayPluginHelloWorldBlock()
   {
      ?>
      <p>Hello World!</p>
      <?php
   }
}
/**
 * Demonstration of creating a plugin utilizing the WordpressPlugin Framework.
 */
if( !$myTestPlugin )
{
  // Create a new instance of My Test Plugin class and initialize the plugin.
  $myTestPlugin = new MyTestPlugin();
  $myTestPlugin->Initialize( 'Test Plugin for the Wordpress Plugin Framework', '1.00', my-test-plugin', 'my-test-plugin' );
  // Add the plugin options and initialize the plugin.
  $myTestPlugin->AddOption( $myTestPlugin->OPTION_TYPE_TEXTBOX, 'myTextboxOption', 'Hello!', 'Simple textbox option for your plugin.' );
  $myTestPlugin->RegisterOptions( __FILE__ );
  // Add the administration page content blocks and register the page.
  $myTestPlugin->AddAdministrationPageBlock( 'block-hello-world', 'Hello World', $myTestPlugin->CONTENT_BLOCK_TYPE_SIDEBAR, array($myTestPlugin, 'HTML_Dis
  $myTestPlugin->RegisterAdministrationPage( $myTestPlugin->PARENT_MENU_PLUGINS, $myTestPlugin->RIGHTS_REQUIRED_ADMIN, 'My Test Plugin', 'My Test Plugin O
}
```

Shortcut Text	Internet Address
http://code.google.com/p/wordpress-plugin-framework/w/edit/WordpressPluginFrameworkUserManual	http://code.google.com/p/wordpress-plugin-framework/w/edit/WordpressPluginFrameworkUserManual
Add Your Plugin	http://wordpress.org/extend/plugins/add/
Click Here To Download The Latest Release of the Wordpress Plugin Framework	http://wordpress.org/extend/plugins/wordpress-plugin-framework/
Click Here To Read About Wordpress Plugin Development	http://codex.wordpress.org/Writing_a_Plugin