



The Squidoo Module Development Kit (MDK)

Introduction

The MDK toolkit makes it easy to build a new module for Squidoo. Modules are created by making two files: one to control the edit form, and another for showing the saved results.

Requirements

In order to use the MDK, you'll need a web server with either PHP 4 or PHP 5 installed. PHP5 is required to run the example modules provided in the MDK. Our production servers use PHP 5.1, so feel free to use new features such as simplexml.

In production mode your module data will be stored in a database; however in the MDK all data is serialized and stored in a text file in the data/ directory. Depending on your server configuration, you may need to grant permissions for PHP to write to that directory.

Contents

1. index.php - framework for running and testing the module during the development process
2. form.php – controls the form for the module
3. view.php – controls what the saved module looks like
4. Two real modules (so you can see how they work)
5. Documentation

How It Works

- Download the MDK.
- Extract the kit and upload the entire directory to your web server.
- Open index.php in your web browser. Once there, you'll see a page similar to Squidoo's Workshop, which will help you view and debug your module during development.
- When you click the orange Edit button, an Ajax call is made to parse and display the contents of form.php.

- When you click the Save button the form contents are saved to a file in the data/ directory. Then view.php is parsed using the data saved from the form.

The Form

The first step to creating your module is to build an HTML form using form.php. We've included the basics (title, subtitle, and description) to get you started. When creating your form, name your form fields using the following format:

```
<input type="text" name="modules[id][details][fieldname]"
      value="<?php echo $module->details['fieldname']; ?>" />
```

where **fieldname** is the name of your field. Naming the fields in this format ensures that our javascript parser will be able to properly identify and store your fields in the database.

Here are a few more examples:

A multi-line text box

```
<textarea name="modules[id][details][fieldname]" rows="4" cols="40">
<?php echo $module->details['fieldname']; ?>
</textarea>
```

A set of checkboxes

When assigning multiple values to a single variable, the data will not be stored in an array, but in a comma-delimited serialized string. Therefore, a string function such as strpos() should be used to determine if a value exists in the serialized string. Split() and in_array() could also be used to convert the string into an array, but care should be taken to ensure that none of the values contain a comma, which is the delimiter.

```
<input type="checkbox" name="modules[id][details][fieldname][]"
      value="foo" <?php if (strpos($module->details['fieldname'], 'foo') > -1)
      echo 'checked="checked"'; ?> /> foo
```

```
<input type="checkbox" name="modules[id][details][fieldname][]"
      value="bar" <?php if (strpos($module->details['fieldname'], 'bar') > -1)
      echo 'checked="checked"'; ?> /> bar
```

Assigning default values to variables

You may want to consider adding some initialization code for the fields in your form. Generally, this code should be added to the very top of the form.php file. This will prevent any errors from variables not being initialized (which is the case until the module

is saved for the first time). In addition, doing so gives you the ability to define default values to your fields. Example:

```
if (!array_key_exists('fieldname', $module->details)) {
    $module->details['fieldname'] = "";
}
```

The View

The view file is essentially a template for how data from the form is presented on a lens. This is where you'll place your HTML template, CSS styles, and API calls.

In view.php your saved data from the form is stored in an array called `$this->attributes['details']`. Example:

```
<p><?php echo $this->attributes['details']['fieldname']; ?></p>
```

When using a third party API, please begin by using your own API key. We will apply for our own commercial API key before uploading your module to our production servers. Because API calls don't always connect, if you're using REST, please use our `$this->rest_connect()` function. The function attempts 3 connections over the course of about 10 seconds, and will return false if no connection was made. Here's an example:

```
<?php
if ($results = $this->rest_connect('url')) {
    echo 'Connection was successful.';
} else {
    echo 'Connection failed';
}
?>
```

Other stuff you might need to know

- If any PHP errors occur while testing your module, they *should* be displayed within the module frame.
- We've included a basic template and stylesheet to help make your module layout consistent with our UI. Should you need to add more styles, please add them directly to the form or view file, not the stylesheet in the styles/ directory.
- Should you need to write additional javascript code, just add it at the top of the form or view file.
- Place any libraries or other miscellaneous code files in lib/.
- If you need to create a helper app in PHP to retrieve something from an external API after responding to user input on the form, put your app in scripts/ajax/. If this doesn't make sense don't worry – if you need it, you'll know.
- We use the Prototype 1.4.0 and scriptaculous 1.5.1 libraries, so feel free to take advantage of what they have to offer.

Getting Help

If you need help or run into a problem while developing your module, please use our developer support forum at <http://www.squidu.com>. (You'll need to sign up for a Squidoo account and fill out [this short form](#) before visiting SquidU). After logging in, click on Lounge, then the Module Development forum. You can generally expect an answer from one of our engineers within a few hours. We're happy to help.