PHPDevShell

Manual

1 — Last update: 2015/03/03

PHPDevShell

Table of Contents

Introduction

The current documentation is simple and to the point, please feel free to contribute to this documentation.

Please continue reading our Presentation. PHPDevShell uses the wonderful Launchpad System for team development collaboration. See the Launchpad Manual for instructions on using this tool.

For download details and other technical information, please visit our page on Launchpad.

- Read the FAQ (it's on Launchpad)
- Download PHPDevShell.
- Code Repository (on Launchpad)
- API Documentation

Join the development team by following this process

Joining the PHPDevShell Development team
If you want to get in contact with us for any matter, whether its to become a plugin developer, talk to us or just get support. Please use the forms supplied on the website.

You will need to join the development team if you are not already joined Joining PHPDevShel
Now that you have joined please read Bazaar Team Branches and understand the process clearly.
After you understand the process, please find out what branch you will be working on.

Presentation

What is PHPDevShell?

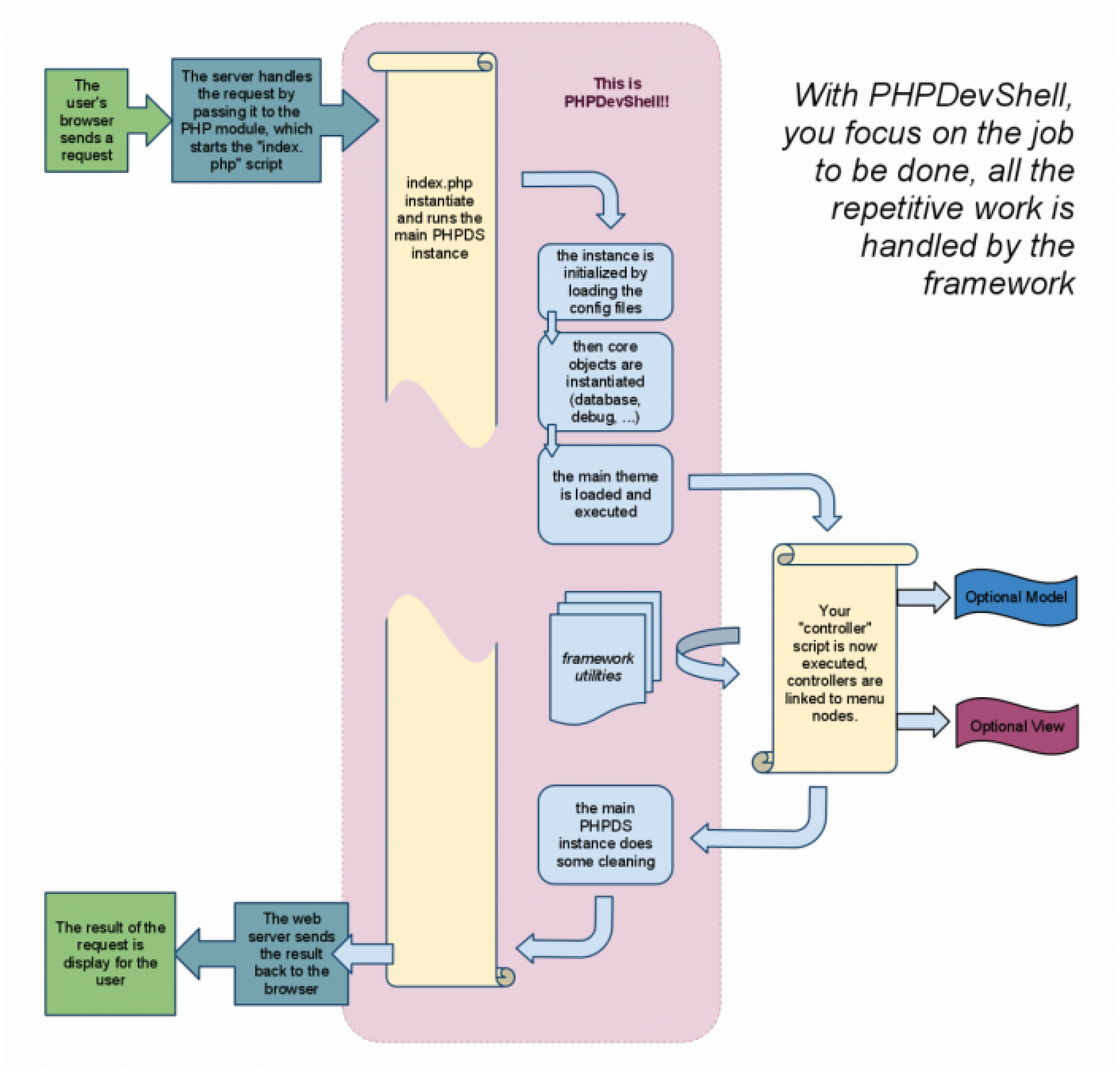
PHPDevShell is a PHP framework with a GUI. In short, that means it provides you with both PHP libraries to build on (like Zend Framework) and a working website to put your scripts in. Hence the name, it's in fact a "shell" to shelter and protect your work.

Where would I typically use PHPDevShell?

PHPDevShell can be used to develop web orientated system, from websites to business applications. PHPDevShell ships with most common used GUI functionality found in applications and websites. This head start in development allows you to immediately focus on the problem at hand. For security and usability it already provides you with user registration and management, role management, group management, access rights security, system settings, powerful theming system, control panel, cronjob management, menu management, plugin management, tagging management, CRUD, ORM, etc.

Why would I want to use PHPDevShell?

PHPDevShell provides the ability for you to get a php application/website up and running in an incredibly short amount of time. But what about resource usage and execution speed? PHPDevShell's default memory usage is half that of Wordpress which is also considered "lightweight" in the world of PHP. PHPDevShell never forces you in a particular design philosophy either. There is no doubt that there are incredible frameworks out there, we respect all of them and acknowledge them. However, PHPDevShell is designed to perform where little development time/resources is available.


The user's browser sends a request
The server handles the request by passing it to the PHP module, which starts the "index.php" script
This is PHPDevShell!!
With PHPDevShell, you focus on the job to be done, all the repetitive work is handled by the framework
index.php instantiate and runs the main PHPDS instance
the instance is initialized by loading the config files
then core objects are instantiated (database, debug, ...)
the main theme is loaded and executed
framework utilities
Your "controller" script is now executed, controllers are linked to menu nodes.
Optional Model
Optional View
the main PHPDS instance does some cleaning
The web server sends the result back to the browser
The result of the request is display for the user

Complete overview of a request handled by PHPDevShell

Final Word

Simply install PHPDevShell as per instructions, write a PHP script as you would normally using clean php, add it to the menu and you have a full blown system behind it handling everything for you. You will impress your clients having an available preview of the base within hours.

Unlike most other frameworks PHPDevShell does not generate any kind of code. No terminal commands have to be executed either. There are several reason why we opted out of code generation, mostly because a newcomer feels lost and things could get messy or complicated quickly. The cleanest way of coding is by doing things by hand.
PHPDevShell is an Open Source (GNU/LGPL) PHP Rapid Application Development framework, aimed at developing any web based application, where Speed, Security, Stability ,Modularity and Flexibility are all essential. It is designed keeping a very easy learning curve in mind. What makes PHPDevShell unique is the fact that if you know PHP, you can write your application inside PHPDevShell in minutes without learning a new “language” so to speak.

Wrapped in a complete HTML5 default theme, PHPDevShell provides:

- End-User system (Advance Registration, Login, Logs, Profile Management).
- Advanced Administration system.
- Cronjob system
- Templating system
- Unlimited level security system (Groups and Roles)
- Plugin System (To develop plugins for PHPDevShell)
- Logs System (Multiple Levels)
- Notice Objects
- Search Engine
- Help Engine
- Menu System
- Tagging System
- Class Registry System
- Easy Forms (CRUD)
- ORM
- And many more…

Version scheme

PHPDevShell release versioning follows a numerical convention comprised of three numbers: Major, Minor and Maintenance. The version is presented in the major.minor[.maintenance] format.

Major Release Number (X.1.1)

An increment of the major number generally indicates a major rework or rewrite of the code base.

May be completely incompatible with prior major releases.

Minor Release Number (1.X.1)

An increment of the minor number usually indicates a significant change to functionality or architecture.

Moderate to high level of backward compatibility with previous minor increments.

Maintenance Release Number (1.1.X)

An increment of the maintenance number usually indicates bug fixing within the minor release and possibly small enhancements and limited new features.

Thank you for trying out PHPDevShell, please ask question and help write documentation. This is all that is asked in return for saving you months of work.

Tools and languages

(updated 2014-06-02 by Greg)

As the name states, PHPDevShell is a framework written in PHP for PHP developers. And as it's mostly meant to create websites, it also has HTML and Javascript components.

You don't need any tool in particular to use it; however there are several tools we are using and that you may want to use too:

- PHPDoc: API documentation is embedded in the source files according to PHPDoc conventions
- PHPUnit: all PHP-related unit testing is done with PHPUnit
- bazaar: currently, the code is hosted on Launchpad, which uses bazaar the versionning system; it's needed to push/pull code from the repository (but not needed to just download a released version)
- PHP Code Sniffer: helps making the code quality better
- Manula: this very documentation is provided throught this service

Also, PHPDevShell itself uses several third-parties Open Source products:

- Smarty: a templating engine for PHP (you are not required to write your own pages with it)
- PNotify Pines): a Javascript libray to display visual notifications
- RedBeanPHP: ORM/CRUD library for PHP (as a plugin)
- TinyMCE: WYSIWYG editor

Available versions

PHPDevShell is at an important moment of its history. A major rewrite is under way, which will focus on its primary goals: simplicity and performance.

After almost 10 years of existence, many little compromises lead to the need of a major cleaning. That means some things have to be broken and some projects may need tuning. So we decided to have two versions :

1. the brand new PHPDevShell version 4.0
2. a transitional version 3.5

Available sources

Here is a list of source where you can find more information:

- the main site : http://www.phpdevshell.org
- Freecode (previously Freshmeat) : http://freecode.com/projects/phpdevshell
- Launchpad : https://launchpad.net/phpdevshell
  - Github : https://github.com/PHPDevShell

Version 3.x

This part describe the “legacy” version 3.x.

The current version is 3.5

System Requirements

In order to have PHPDevShell running, your system must meet some requirements.

You need the following on your Apache server for PHPDevShell to work correctly.

- PHP: Greater then PHP 5.2.x is minimum from PHPDevShell V 2.5.1.
- PHPDevShell was developed on MySQL 5.0.x, I believe older versions above MySQL 4.1 should work fine. (Multiple database support will be added soon).
- Apache: PHPDevShell was developed on Apache 2.x, 1.x should work fine though.

Be careful! the first file (PHPDS-defaults.config.php) is part of the distribution and therefore will be overwritten on each update. You should not modify this file, as your changes will be lost.
For a simple single site setup use the single-site.config.php.RENAME (remove .RENAME) file instead, or the site-specific config files (see below). single-site.config.php will always load when available.
Override default settings by copying settings from PHPDS-defaults.config.php to your own config file.

Installing

Now that you have checked that your system meet the requirements, you can proceed to the actual installation.

Installation of PHPDevShell is extremely easy;

1. Extract the downloaded compressed file of the PHPDevShell package.
2. Create a Mysql database (Character set : UTF-8 Unicode and Collation : utf8_general_ci) note the database name, username and password.
3. Rename /config/single-site.config.php.RENAME to /config/single-site.config.php and change the settings.
4. Upload the complete PHPDevShell directory to your web server including your modified setup files.
5. Make the whole “write” and sub folders writable (if default else make custom folders writable).
6. Browse to the http://localhost/other/service/service.php (directory where you uploaded PHPDevShell), a Database install script will appear, complete this as the final installation step.
7. Follow instructions on database install script that loads when you open PHPDevShell in your browser…
8. Optional Cronjobs

Create cronjob to run [http://[install location]/index.php?m=742061208 every 10 minutes or lower (if automated file execution is required).

Optional Cache, Gzip, Friendly URLs

If your server support mod_rewrite and mod_expires you can rename rename.htaccess to .htaccess to gain cache, gzip performance and friendly urls. PS: Remember to switch friendly urls on in the gui to make it work.

Optional Multiple Domains

For multiple domains working from the same directory with different databases for each package, modify _host.config.php and add configuration files as required.

Upgrading

Always do a database and file backup before upgrading.

Most older plugins will still work on PHPDevShell v3, however we have provided a backwards compatible theme for your old plugins called oldlegacy2, so make sure your old plugin files are switched to this theme if they are not using a custom theme.

Be careful! the first file (PHPDS-defaults.config.php) is part of the distribution and therefore will be overwritten on each update. You should not modify this file, as your changes will be lost. For a simple single site setup use the single-site.config.php.RENAME (remove .RENAME) file instead, or the site-specific config files (see below). single-site.config.php will always load when available.
Override default settings by copying settings from PHPDS-defaults.config.php to your own custom config file.

1. Extract and upload downloaded PHPDevShell package to your PHPDevShell install directory overwriting old content.
2. Rename /config/single-site.config.php.RENAME to /config/single-site.config.php and change the settings.
3. Execute /other/service/service.php from your installed PHPDEvShell directory.
4. Complete the instructions in your browser.

If you have PHP notices turned on, you might see a notice until you ran the upgrade, ignore these notices.

Errors, Exception, and Debug

Posted on Saturday, Oct 30, 2010 by Greg

PHPDevShell offers three errors-related mechanisms which can sometimes confuse the new user because they seems similar at first sight:

- **Errors** occurs when an unexpected problem cannot be solved by the program
- **Exceptions**, is a standard mechanism allowing the program to handle special cases
- **Debug** provides the user tools to interact with the program in order to fix bugs
  These three mechanisms interact with each other so we will study all three here.

These are three separate, however related, concepts.

Errors

Errors occur when something goes wrong at the language level: a typo, an impossible action (division by zero), a type error… They are generated by the PHP engine. Depending on the configuration, they can be left alone (not wise, it will break the script), or handled by PHPDevShell and turned into an special exception called `ErrorException`.

You can divide errors into three generic categories: *notices* (a situation which is probably harmless), *warning* (a situation which is likely problematic) and *critical* (abandon all hopes). The usual policy is to display notices and keep the cycle running, and handle everything else as a dead end. You can configuration everything to suit your needs (see below).

Exceptions

Present in most modern languages, the goal of the exception mechanism is to deal with “special cases” out of the main stream of code. The advantages of using it are:

- **protection**: any error occurring anywhere in the function or the functions it calls can be caught
- **code readability**: the error handling code is distinct and doesn’t mess with the main code
- **error versatility**: many types of errors can be handled, even if they’re not known from the developer
- **responsibility**: the function can handle some errors and leave the other to someone else

You can read about PHP exception in the online manual. PHPDevShell allows and uses Exceptions everywhere in the code. A top-level exception handler is provided in case a thrown exception is not caught anywhere else.

Example (from PHPDevShell code):

```
try {
        $this->db->get_essential_settings();
} catch (Exception $e) {
        session_destroy();
        header('Location: install.php');
}
```

That means: we try to fetch the essential settings from the database. In case we fail, we assume we need install is needed.

Debug

The debugging modules provided by PHPDevShell are basically extended `error_log()` functions. They are not designed to substitute for PHP debuggers such as Zend Debug or XDebug. They use the conduits configured by the `ExceptionHandler` to send data to the developer.

The ExceptionHandler

The ExceptionHandler's job is to react to such situations by reporting them according to the site's configuration. The reports can be sent through various conduits and should contain as much information as possible to help the site's owner/developer to give the appropriate response.

Currently the following conduits are available:

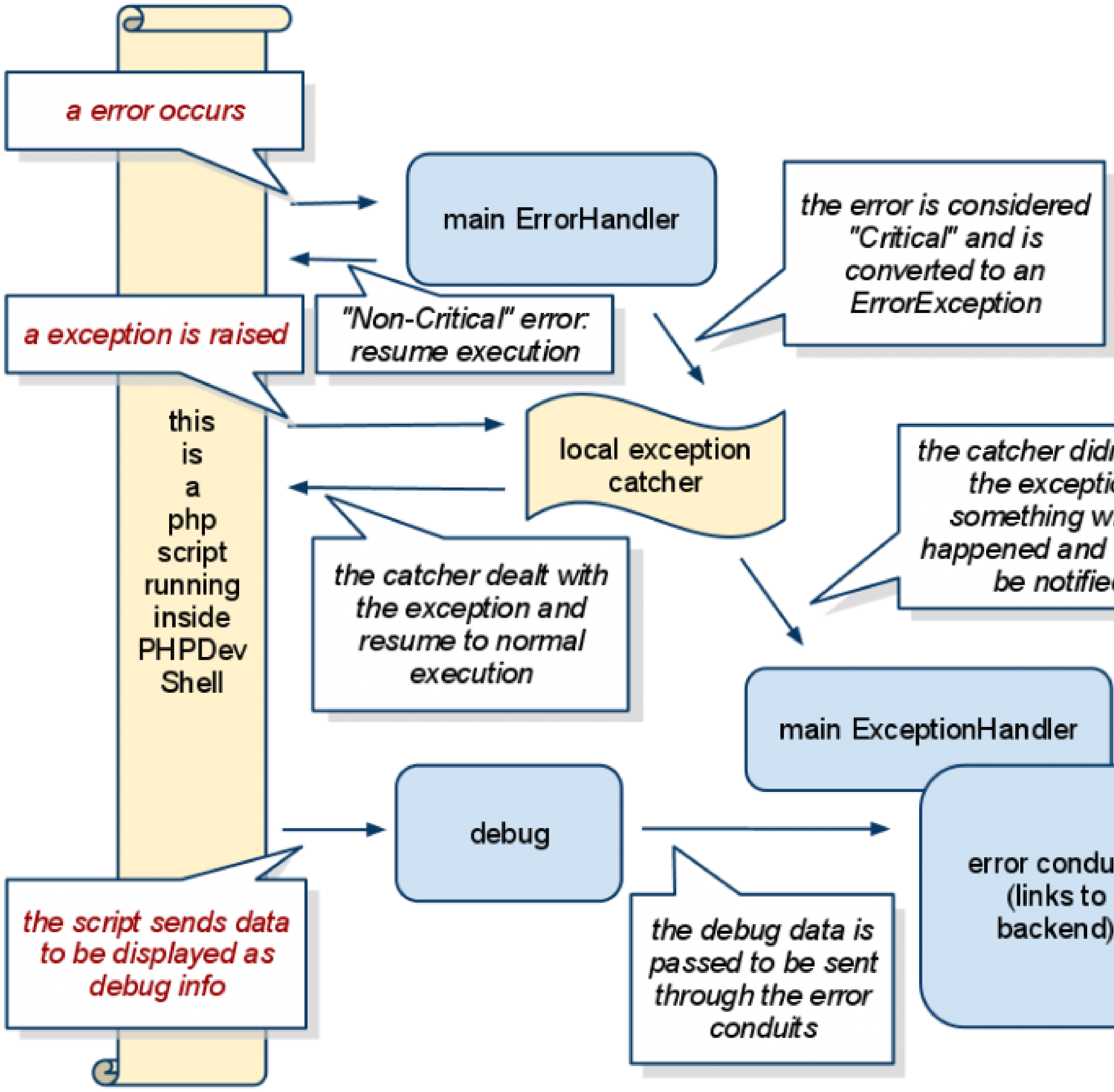
- *on screen* (on production site, it's better to let the user know an error occurred with no more information to avoid security leaks)
- through *FireBug/FirePHP* (only for development)
- by *mail* (to inform the site owner if he's not monitoring the site in real time)
- on *syslog*

Each can be activated and configured separately.

Depending on the conduit, the report contains information such as the location of the error, the timestamp when it occured, the backtrace of the function which triggered the error, etc.

Overview

You can see how these interact in this picture:


a error occurs
main ErrorHandler
the error is considered "Critical" and is converted to an ErrorException
"Non-Critical" error: resume execution
a exception is raised
this is a php script running inside PHPDev Shell
local exception catcher
the catcher didn the exceptio something w happened and be notifie
the catcher dealt with the exception and resume to normal execution
main ExceptionHandler
debug
error condu (links to backend)
the script sends data to be displayed as debug info
the debug data is passed to be sent through the error conduits

The relationships and interactions of Errors, Exceptions and Debug in PHPI

Enabling and disabling

- The exception mechanism is built in PHP so it cannot be disabled. If don't want to use it, just don't.
- The error handler is part of PHPDevShell and cannot be disabled; however you can enable/disable each conduit, so if you disabled all conduits, you won't receive any message.
- The debug can be disabled (useful for production sites) based on the concept of "domains" (refer to the documentation for debug).
  Remember: on a production site, debug MUST be disabled but errors MUST be handled (nicely).

Configuration

The configuration files allow you to:

- tell how you want error types to be handled (break on notices or not, on warnings, etc)
- filter what kind of error you want to to be handled (it's usually a good thing to catch all errors)
- set which conduits should sent the information where
  Note: if a warning or a notice occurs when set to non-fatal, they won't be converted to exception and therefore won't be catchable. They will, however, be displayed (unless configured to be ignored).

	fatal = false	**fatal = true**
ignore = true	not displayed, not catchable	not displayed, not catchable
ignore = false	displayed, not catchable	not displayed, catchable

Usage

Let's see how we can actually use our little debugger.

Sending data to the conduits

Basic usage is very simple: just send a string to the `log()` method:

```
$this->log("Hello, console");
```

This method is defined in PHPDS_dependant so pretty much any code can access it.

Domains

All data sent thought the conduits are tagged with a semantic domain, such as “db” or “user”, so only the data you want to study are actually sent. The default behavior is to use the class name (with the PHPDS_ prefix riped off). For you own classes, you can change it to anything you fancy, maybe having something adaptative. Take a look at what `PHPDS_query` does:

```
public function debugInstance($domain = null)
        {
                return parent::debugInstance(empty($domain) ?
'QUERY%'.get_class($this): $domaine);
        }
```

debug instances

Almost every object using the debug has a private debug instance ; this is mostly done to allow domain filtering, but you can override the `debugInstance()` method to provide your own customized debug instance or just customize the domain, like the Query system does.

Conduits

Data sent inside the debugger can be broadcast to various locations, depending on your configuration. Here are the default conduits available.

Display

This is the most detailed way of displaying errors, and it’s mostly used when the site is under development. It will display as much information as possible, including backtrace and code fragments.

Can be enabled (`true`) or disabled (`false`).

```
$configuration['error']['display_errors'] = true;
```

FirePHP

Used only when developing, FirePHP is very clever way to debug a webpage. It’s a plugin for Firebug which display some debug information sent along with the page (but not inside the page) by PHPDevShell. You can use it to study dynamic debug data without disturbing the page.

Can be enabled (`true`) or disabled (`false`).

```
$configuration['error']['firePHP'] = false;
```

Log file

This is the traditional conduit: writing textual data into a flat text file. Useful to keep hidden traces of a production site, but not interactive and very limited in the complexity of the data stored.

You can choose any file you have write access to.

```
$configuration['error']['file_log_dir'] = 'write/logs/';
```

Mail

A way to be alerted when something goes wrong. Should be used only for production site, and for important errors.

You can set the recipient of the emails.

```
$configuration['error']['email_critical'] = 'yourmail@phpdevshell.org';
```

Early debug

Even if PHPDevShell's error and exception handler is installed early in the cycle, everything executed prior to this point cannot, obviously, be handled that way. To help catching error in this first part of the cycle a very limited debug channel is available by setting the global variable `$early_debug` to true. In that case, anything set to `PHPDS->log()` will be output to the web server's error log (via `error_log()`), until the normal handlers are set.

See also:
Suggestions on error

Tidbits

The followings articles are in fact blog posts, often replying to a problem we were asked directly or on one of the sites.

Access to multiple database

I’ve been recently asked how you can connect to a database server different than the “master”, that is the server where the PHPDS installation resides.

Actually it’s very easy as long as the server is Mysql:

class test_connector extends PHPDS_legacyConnector
{ protected function applyConfig($db_config = ‘’) { $this->dbHost = “localhost”; $this->dbName = “test”; $this->dbUsername = “username”; $this->dbPassword = “userpassword”; }
}

class test_query extends PHPDS_query
{

protected $connector = “test_connector”; protected $sql = “SHOW TABLES”; } $tables = $this->db->invokeQuery(‘test_query’); As you see, a query will use a different database connector if it’s provided in its field. Just give the class name of a connector class and the framework will handle the rest.

In this case, we used the “legacy” connector, which is the Mysql connector used all around by the framework. If you want to connect to another database software, you’ll have to write a custom connector (it’s not a big deal actually, just remapping the library function to class methods).

In a future version, you’ll be able to set the actual parameters in the configuration file, like the “master” database.

See How can I interact with a remote database in my own plugins?

Return values for controllers

If you’re using the controller class, you’re writing the content of a method. Although you can just write your logic and not return anything, you can also return some values to instruct the framework of some special cases.

(this is experimental for 3.0.4 and currently only for the viaAJAX() method)
Note: from 3.2.1 the same applies to regular controllers

There are basically 4 regular cases and a special case :

- if you return nothing (i.e. null), the output is handled the usual way, in other words the whole page is built from the current theme template (it’s the common case for direct requests or ajax request who expect a complete page).

- if you return false (the exact boolean value), the framework will consider something went wrong and will display an error messages. By default this message warns the user that the controller couldn’t be found, but you can customize the message by setting an array in $this->core->haltController.

- if you return true (the exact boolean value), the framework will consider you have handled the output yourself, so won’t output anything on its own (just set an empty template and flush the buffer).

- in the last case, the data you return will be used as the controller content. Be sure to provide only a string, which will be displayed instead of the whole page. That means the theme template will not be used, an empty template will be used instead.

- the special case is when the request explicitly asked for JSON data (by setting the “HTTP_X_REQUESTED_TYPE” http header to “json”), whatever data you return will be json-encoded and returned to the browser.

Currently any other value will throw an exception.

Auto protecting queries

I’m working on the new Forms system and I realized something: by default, query parameters are NOT protected. I will not debate on SQL injection (consult the Oracle), but I want to stress out that you SHOULD ALWAYS protect the parameters, the easy way is to use the `autoProtect` field:

```
class HTVP_filmUpdate extends PHPDS_query
{
protected $sql = 'UPDATE `table` SET `field` = "%(parameter)s" WHERE `id` =
%(id)d';
protected $autoProtect = true;
}
```

(of course it’s not necessary if you deal only with %d parameters)

You can also use `$autoQuote = true` to automatically add quotes to your parameters.

Javascript plugins

Javascript plugins are an easy way to provide some packaged javascript features. It's especially helpful to keep a third party product separated and up to date.

Note that it's not a specific kind of plugin, it's just a feature a regular plugin can propose, so you can have your own "javascript plugins" and use them in your plugin (or site) or from another plugin or site.

We'll study Pines Notification, a library shipped with and used by the default PHPDevShell distribution.

Set up

As noted above, a JS plugin is just a regular plugin, so we have in the `plugins` folder a `PinesNotify` folder, containing these folders:

- `config`: for the `plugin.config.xml` file
- `images`: just the icon in the plugin manager
- `includes`: for the `GUI_notify.class.php` class file
- `public`: containing the files provided by the library

The library files

As the library is javascript code and it's related files, we'll place the downloaded files just inside our `public` folder. We could only copy the files we'll be actually using, but I like to keep the original distribution intact, we I usually put all their files together.

The config file

We will only provide a single class, so the declaration is simple:

```
<install version="1000">
        <classes>
                <class name="GUI_pnotify" alias="pnotify" plugin="PinesNotify" />
        </classes>
</install>
```

You can refer to TODO for a complete description of the config xml file.

The rest of the file is purely descriptive; you can read it here.

OK now the class is known to the system, let’s see what’s inside it.

The PHP class

The purpose of the PHP class is to give the framework the necessary entry points to add the JS code to our page. It must implement the `iPHPDS_activableGUI` interface and as you can see it’s really simple:

```
interface iPHPDS_activableGUI
{
    public function construct();
    public function activate();
}
```

The `construct()` method provides a path to the plugins files, and the `activate()` method will be called when the JS code will be required. Let’s take a look at them:

```
public function construct($path = null)
    {
        $this->path = $path;
    }
```

really simple we just keep the `$path` for future use.

Note that the `$path` is given without a trailing slash, you may want to add it depending on how you plan to use it later.

Then:

```
public function activate()
    {
        $file = $this->configuration['development']
            ? 'jquery.pnotify.js'
            : 'jquery.pnotify.min.js';

        $template = $this->template;

        $template->addJsFileToHead($this->path.'/public/'.$file);
        $template->addCssFileToHead($this->path.'/public/
jquery.pnotify.default.css');
        $template->addCssFileToHead($this->path . '/public/
jquery.pnotify.default.icons.css');
        $template->addCssFileToHead($this->path . '/public/oxygen/icons.css');

        $template->addJsToHead(
         '
            $(function(){
                $.pnotify.defaults.styling = "jqueryui";
            });
        '
        );
    }
```

the first part is a simple way to select a different library on the development and production systems; then we add the JS and CSS files. We are also given the opportunity to add some custom code, in this case defaults values.

You can read the whole file here.

Usage

As you already figured out, using such a plugin is simple:

```
$this->template->activatePlugin('GUI_pnotify');
```

The `activate()` function will be called, hence adding all the required links/CSS/JS to be used at the proper time. Of course it requires that you use a compatible theme, since the theme code is responsible for adding the necessary lines
to the ouput (“cloud” and “emptyCloud” are compatible).

You usually use this call from your controller `execute()` method, but you can use it anywhere between the `template` initialization and the actual output.

Note that you can also pass additional parameters to `activatePlugin()` and they will be passed as such to the `activate()` method.