Zend_Log_Writer_Firebug - Christoph Dorn

<ac:macro ac:name="unmigrated-inline-wiki-markup"><ac:plain-text-body><![CDATA[

<ac:macro ac:name="unmigrated-inline-wiki-markup"><ac:plain-text-body><![CDATA[

Zend Framework: Zend_Log_Writer_Firebug Component Proposal

Proposed Component Name	Zend_Log_Writer_Firebug
Developer Notes	http://framework.zend.com/wiki/display/ZFDEV/Zend_Log_Writer_Firebug
Proposers	Christoph Dorn
Zend Liaison	Matthew Weier O'Phinney
Revision	1.0 - 15 June 2008: Initial Draft. 1.1 - 27 June 2008: Revisions based on comments. 1.2 - 16 July 2008: Major revision based on feedback from Wil and Matthew (wiki revision: 37)

Table of Contents

1. Overview
Why Wildfire
2. References
3. Component Requirements, Constraints, and Acceptance Criteria
4. Dependencies on Other Framework Components
5. Theory of Operation
Overview
Future
6. Milestones / Tasks
7. Class Index
8. Use Cases
9. Class Skeletons

1. Overview

> **Status**
> This component is now in the standard incubator with complete documentation and tests.

Zend_Log_Writer_Firebug is used to inject logging messages into the Firebug Console from PHP. The component is a plugin for the Wildfire project. The goal of the Wildfire project is to develop standardized communication channels and a dynamic and scriptable remote plugin architecture. At this time the primary focus is to provide a system to allow PHP code to inject logging messages into the Firebug Console.

For the purpose of logging to Firebug a communication protocol has been developed that uses HTTP request and response headers to send data between the server and client components. It is great for logging intelligence data generated during script execution to the client browser without interfering with the page content. Debugging AJAX requests that require clean JSON and XML responses is possible with this approach.

Why Wildfire

Javascript is becoming a major force in client browsers and applications not only for enhancing web pages but also the browser and applications themselves. The Firefox browser is a great example of an application that can be extended by Javascript-based extensions. Up until now

extensions installed on the client were fixed in their functionality and ability to interact with the loaded web age. That is about to change.

Wildfire will provide a standard for allowing server-side code to dynamically extend and enhance client extensions in a streamlined and secure fashion. Users will not need to install custom extensions for specific sites, but rather simply authorize a specific site to load dynamic extensions into the browser. The dynamically loaded extensions will reside in a sandbox with the same security policies as any other web page. In addition, the Wildfire client extension will provide an API to interact with the browser itself. This API is also extensible by plugins which are installed as any other browser extension.

The Wildfire concept has been proven by the FirePHP project. The Wildfire communication protocols and plugin system will be developed over time based on user demand.

2. References

- Wildfire - work in progress - will have more information soon
- Firebug
- FirePHP
- symfony debug toolbar

3. Component Requirements, Constraints, and Acceptance Criteria

- This component **will not** be automatically initialized by default
- This component **will** send logging information to Firebug only if the developer requests it
- This component **will not** send any logging information in the response headers if the FirePHP extension is not installed on the client
- This component **will** provide a Zend_Log_Writer implementation
- This component **will** provide a facility to log Exceptions
- This component **will** provide a Zend_Controller_Plugin_Abstract implementation to flush debug data on request end but before page content is sent
- This component **will not** prevent cyclical exception handling should exceptions occur in the component. This **must** be handled by the developer.
- This component **will** make use of Zend_Json_Encoder to encode debug data
- This component **will** allow for mapping of all existing and custom Zend_Log priority levels to Firebug logging styles
- This component **will not** save any data using Zend_Cache or the filesystem.

4. Dependencies on Other Framework Components

- Zend_Loader
- Zend_Log_Writer_Abstract
- Zend_Controller_Request_Abstract
- Zend_Controller_Response_Abstract
- Zend_Controller_Plugin_Abstract
- Zend_Json_Encoder

5. Theory of Operation

Overview

The idea is that the Zend_Log_Writer_Firebug component will provide the first transport to send debug information to a client in a non-destructive way. In this case the Firebug Console (via FirePHP).

Zend_Log_Writer_Firebug as it is implemented now provides a great way to aid in debugging Zend Framework applications on an ad-hock basis.

This use-case will be expanded to a more comprehensive debugging framework that hooks into all Zend Framework components.

Future

The future debugging/intelligence/insight system, possibly implemented by an enhanced Log Writer should collect data at all critical points within all components and provide facilities to enable and disable debug information on a component basis. The collected debug information may be persisted to files, logged or otherwise recorded as well as sent to the client via Zend_Log_Writer_Firebug.

A future version of Zend_Wildfire (the Firefox extension as well as the Zend components) will provide a debug toolbar to be displayed in the browser similar to the symfony debug toolbar. To achieve this we must first collect the mentioned debug information on a component basis.

All functionality beyond the current implementation should be part of new proposals that if applicable require certain additions to the Zend_Wildfire components.

6. Milestones / Tasks

- Milestone 1: [DONE] Reference implementation
- Milestone 2: [DONE] Reference implementation refined
- Milestone 3: [DONE] Working prototype checked into the incubator
- Milestone 4: [DONE] Unit tests exist, work, and are checked into SVN.
- Milestone 5: [DONE] Initial documentation exists.

7. Class Index

- Zend_Log_Writer_Firebug
- Zend_Wildfire_Exception
- Zend_Wildfire_Plugin_FirePhp
- Zend_Wildfire_Plugin_Interface
- Zend_Wildfire_Channel_Interface
- Zend_Wildfire_Channel_HttpHeaders
- Zend_Wildfire_Protocol_JsonStream

8. Use Cases

Typical use of the Zend_Log_Writer_Firebug component will be in conjunction with Zend_Controller_Front. In these use-cases initialize the component in your bootstrap file and use the logging calls in your models, views and controllers.

```
$writer = new Zend_Log_Writer_Firebug();
$logger = new Zend_Log($writer);
```

You can also use Zend_Log_Writer_Firebug without Zend_Controller_Front.

```
$writer = new Zend_Log_Writer_Firebug();
$logger = new Zend_Log($writer);

$request = new Zend_Controller_Request_Http();
$response = new Zend_Controller_Response_Http();
$channel = Zend_Wildfire_Channel_HttpHeaders::getInstance();
$channel->setRequest($request);
$channel->setResponse($response);
/**
 * Now you can make calls to the logger
 */

$logger->log('This is a log message!', Zend_Log::INFO);

/**
 * Flush log data to browser
 */
$channel->flush();
$response->sendHeaders();
```

9. Class Skeletons

]]></ac:plain-text-body></ac:macro>
]]></ac:plain-text-body></ac:macro>