Zend_Service_Nirvanix

<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_Service_Nirvanix Component Proposal

Proposed Component Name	Zend_Service_Nirvanix
Developer Notes	http://framework.zend.com/wiki/display/ZFDEV/Zend_Service_Nirvanix
Proposers	Mike Naberezny, with Darby Felton and Matthew Weier O'Phinney
Revision	1.0 - 29 January 2008: First release. 1.1 - 30 January 2008: Added examples UC-05 and UC-06. (wiki revision: 15)

Table of Contents

1. Overview

This component will provide lightweight wrappers for the Nirvanix IMFS (Internet Media File System) services, in the same spirit of other Zend Framework web service components like Zend_Service_StrikeIron.

These wrappers will abstract away details of the XML-based REST interface and provide a familiar and easy-to-use PHP interface for IMFS that will be accessible to more PHP developers.

2. References

- Nirvanix API documentation

3. Component Requirements, Constraints, and Acceptance Criteria

- The component will wrap the Authentication namespace to authenticate with Nirvanix for access to the IMFS services.
- The component will provide support for the IMFS namespace, allowing files on the remote file system to be manipulated and retrieved.
- The component will provide support for the Metadata namespace, allowing tags and other metadata attached to file on the remote file system to be searched and manipulated.

- The component will provide support for the Upload namespace, to facilitate uploading data to the remote file system.
- The component will not initially provide support for other Nirvanix namespaces but will be built in a way that they can be easily added at a future time.

4. Dependencies on Other Framework Components

- Zend_Http_Client
- Zend_Exception

5. Theory of Operation

Nirvanix IMFS is a utility computing service that provides a remote file system accessed over the web using either SOAP or vanilla XML over HTTP (REST). This wrapper will implement the faster REST-based interface.

This wrapper provide PHP applications with a convenient interface to use the IMFS through familiar easy-to-use PHP objects representing remote folders, the files they contain, and associated metadata.

6. Milestones / Tasks

- Iteration 1 - Nirvanix Base Class, Exceptions
- Iteration 2 - Access to Folders and Files
- Iteration 3 - File Uploads
- Iteration 4 - Metadata Support
- Iteration 5 - Documentation (DocBook)

7. Class Index

- Base (Zend_Service_Nirvanix)
- Filesystem (Zend_Service_Nirvanix_Imfs_*)
- Error (Zend_Service_Nirvanix_*Exception)

8. Use Cases

UC-01	Authentication

The Nirvanix service requires a username, password, and API key to be supplied in order to get a session token that allows operations against the service.

```
$nirvanix = new Zend_Service_Nirvanix(array('username' => ''
                                            'password' => '',
                                            'apiKey'   => ''));
```

The credentials are supplied in an associative array for clarity and to allow any future options that might be needed.

UC-02	Store a New File

Quickly create a new file on the remote file system:

```
$file = $nirvanx->putContents('/path/to/foo.txt', $data);
```

The Nirvanix file system is a hierarchy of folders and files where the path separator is always "/". The library will attempt to validate paths per the Nirvanix specification before sending them to the service.

Some applications may not need a folder structure. If no path is specified, the root folder will be used.

The $data may be either a string or a stream resource. The return value will be an object representing the file that can then be manipulated.

UC-03 Retrieve File Contents

Quickly retrieve the contents of a file:

```
$data = $nirvanix->getContents('/path/to/foo.txt');
```

The getContents() method will return the data as a string, which may be the most common usage. A corresponding getStream() method will return the file's data as a stream resource for working with large files. Finally, a getObject() method will return an object representing the file and its metadata on the Nirvanix filesystem to allow it to be manipulated.

UC-04 Working with a Folder

Folder contents can be listed. Folders may contain both files and child folders.

```
$folder = $nirvanix->getFolder('/foo');
foreach ($folder->getEntries() as $entry) {
  echo $entry;
}
```

Each entry in the folder is either a file object or a folder object that can then be manipulated or retrieved.

Relative operations may also be done on the folder and the absolute path will be computed automatically.

```
$file = $folder->putContents('foo.txt', $data);
```

UC-05 Working with a File Object

For more sophisticated uses of the files on the Nirvanix filesystem, a getObject() method is provided to return an object representing the file and its metadata. This is the same object that is returned when traversing a Folder (see UC-04).

```
// get a file object
$file = $folder->getObject('foo.txt');

// built-in attributes
$size = $file->getSize();
$data = $file->getContents();
```

UC-06 Working with File Metadata

Nirvanix also supports attaching metadata to a file. These can be arbitrary key/value pairs and are exposed through the File object.

```
// display arbitrary metadata
foreach ($file->metadata as $k => $v) { echo "$k = $v\n"; }

// add arbitrary metadata to the file
$file->metadata['foo'] = 'bar';
$file->update();
```

9. Class Skeletons

Class skeletons will not be proposed at this stage. When development begins, behavioral requirements of the objects will be expressed as tests and the classes will then be developed based on these requirements using TDD.

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