Filesystem

Overview

The main entry point for the file system API is the HubzeroFilesystemManager. When working with file systems, this is the class that methods will be invoked on. The Manager makes use of the adapter pattern which helps eliminate the inconsistencies of the different file systems by providing a common interface. So, wether using a Local, Ftp, or Dropbox adapter, the method calls and returned data types will be the same.

Adapters

Local

This is the default file system adapter.

```
$adapter = new HubzeroFilesystemAdapterLocal();

$filesystem = new HubzeroFilesystemFilesystem($adapter);
```

FTP

```
$adapter = new HubzeroFilesystemAdapterFtp(array(
    'host' => 'ftp.example.com',
    'port' => 21,
    'username' => 'username',
    'password' => 'password',
    'root' => '/path/to/root',
));

$filesystem = new HubzeroFilesystemFilesystem($adapter);
```

None

A base adapter used primarily for testing.

```
$adapter = new HubzeroFilesystemAdapterNone();

$filesystem = new HubzeroFilesystemFilesystem($adapter);
```

Retrieving Files

The read method may be used to retrieve the raw string contents of a given file:

```
Filesystem::read('file.jpg');
```

The exists method may be used to determine if a given file exists:

```
if ( ! Filesystem::exists('file.jpg'))
{
    throw new Exception('File not found.');
}
```

Storing Files

The write method may be used to store a file on disk. This method accepts two arguments: the file path and the contents to write.

```
Filesystem::write('file.jpg', $contents);
```

Copy / Move

The copy method may be used to copy an existing file to a new location on the disk:

```
Filesystem::copy('old/file1.jpg', 'new/file1.jpg');
```

The move method may be used to move an existing file to a new location:

```
Filesystem::move('old/file1.jpg', 'new/file1.jpg');
```

Prepend / Append

The prepend method allows for easily prepending contents to the beginning of an existing file.

```
Filesystem::prepend('file.log', 'Prepended Text');
```

Similarly, the append method allows for easily appending contents to the end of a file.

```
Filesystem::append('file.log', 'Appended Text');
```

Removing Files

The delete method accepts a single filename to remove from the system:

```
Filesystem::delete('file.jpg');
```

Directories

soon

Macros

If a feature is not included in the Filesystem class, it can be extended through macros. A macro can extend basic, existing functionality to perform more complex tasks. One example of this would be a macro for creating a directory tree.

Macros must implement HubzeroFilesystemMacroInterface and generally consist of at least two methods: getMethod and handle.

```
class Example extends HubzeroFilesystemMacroBase
{
 public function getMethod()
 {
  return 'examplify';
 }

 public function handle($path)
```

```
 {
  $data = $this->filesystem->read($path);

  return $data . 'EXAMPLE';
 }
}
```

The getMethod method returns the name of the call to the filesystem that will invoke the macro.

The handle method does all the real work. When a macro is invoked, the filesystem object calls handle on the macro and passes it all arguments it received from invoking call.

```
// Add the macro to the filesystem object
$filesystem->addMacro(new Example);

// Append 'EXAMPLE' to a file's contents
$content = $filesystem->examplify('path/to/file');
```