Contents

1 Installation 3

2 The Uploader 5
   2.1 Storage Adapters .............................................................. 5
   2.2 Version Descriptions .......................................................... 5
   2.3 File Naming ................................................................. 6

3 Integration Examples 7
   3.1 Laravel ................................................................. 7

4 Indices and tables 9
Hitch comes bundled as a package in the Packagist repository.

Ensure that you have setup composer, and add clarkf/hitch to your composer.json file:

```json
{
    "require": {
        "clarkf/hitch": "@dev-master"
    }
}
```

**Be careful.** Hitch has not yet been fully released, and as such, does not have any tagged versions within Packagist. By requiring @dev-master, you will get the most current version from the Github Repository. Be sure to verify the build status before including it in your project!
The Uploader is the core of Hitch. It directs your application how and where to store files. Simply define an uploader by extending Hitch\Uploader.

```php
class MyUploader extends Hitch\Uploader {}
```

The most important method provided by an uploader is its `store()` method. Simply provide a File object, and Hitch will take care of the rest:

```php
public function controllerAction()
{
    $file = $this->getRequest()->getFile('file');
    $uploader = new MyUploader();
    $uploader->store($file);
}
```

### 2.1 Storage Adapters

The uploader passes the files that it generates off to Storage Adapters. You can specify which storage adapters to use by overriding the uploader's `getStorageAdapters()` method:

```php
public function getStorageAdapters()
{
    $root_path = get_path_of_public_dir();
    return array
    (
        new Hitch\Storage\File($root_path)
    );
}
```

#### 2.1.1 File Storage Adapter

Hitch\Storage\File provides a means to store a file on the local filesystem. It’s also the default storage adapter. Simply pass the `root upload directory` as the constructor, and your files will be stored:

```php
new Hitch\Storage\File($root . "/public/images/");
```
2.2 Version Descriptions

Hitch will automatically generate different versions of each uploaded files for you. You must describe each version that you require:

```php
public function getVersionDescriptions()
{
    'thumb' => array(
        'resizeKeepAspect' => array(100, 100)
    ),
    'icon' => array(
        'resize' => array(16, 16)
    )
}
```

2.2.1 resize process

resize an image to exactly the supplied dimensions

```
"resize" => array(100, 100)
```

becomes

2.2.2 resizeKeepAspect process

Resize an image while maintaining its aspect ratio.

```
"resizeKeepAspect" => array(100, 100)
```
Available processes:

- **resizeKeepAspect($width, $height)** - Resize the image to the specified size, while keeping the aspect ratio

### 2.3 File Naming

You may specify how your files are named by overriding `getFilename()` and `getVersionPath()`:

```php
public function getFilename($original, $version = null)
{
    // Save files with timestamps
    $extension = $original->getExtension();
    return microtime(true) . $extension;
}

public function getVersionPath($original, $version = null)
{
    $filename = $this->getFilename($original, $version);
    // Save versions in their own subdirectory
    if (is_null($version)) {
        return "images/" . $filename;
    } else {
        return "images/" . $version . "/" . $filename;
    }
}
```
3.1 Laravel

```php
<?php
namespace Acme\Models;

use Illuminate\Database\Eloquent\Model;
use Acme\UpLoaders\ImageUploader;
use Symfony\Components\HttpFoundation\File\File;

class Image extends Model
{
    protected $fillable = array(*/ .. */
        "image",
        /* .. */
    );

    /**
     * @var Hitch\Uploader The uploader
     */
    protected $uploader;

    /**
     * @var File $file The file to upload
     */
    protected $file;

    /**
     * Initialize model events
     */
    public static function boot()
    {
        // Call the parent’s boot method
        parent::boot();

        // Attach to this model’s ‘saved’ event.
        self::saved(function ($image) {
            // Check to see if there was an image to upload
```

Integration Examples

CHAPTER 3
if (!isset($image->image)) {
    return;
}

// Pass the File to the Uploader’s store() method
$image->getUploader()->store($image->image);

/**
 * Get the uploader associated with this model
 *
 * @return Hitch\Uploader The uploader
 */
public function getUploader() {
    // Check to see if uploader has been initialized. If it hasn’t,
    // initialize it now
    if (!isset($this->uploader)) {
        $this->uploader = new ImageUploader;
    }

    return $this->uploader;
}

/**
 * Set the image attribute
 *
 * @param File $file The file to store
 *
 * @return void
 */
public function setImageAttribute(File $file) {
    // Note that we don’t want to store() the file now; the model
    // has not yet passed validation. For now, we’ll just store it
    // until the model has been saved.
    $this->file = $file;
}
Indices and tables

• genindex
• modindex
• search