Ambition. Enthusiasm. Lifeblood. Flexibility.
Network. Passion for technology. Munich.

ExporterBundle

Documentation v0.2a

January 5, 2013

Bruno Lorenz - Namics (Deutschland) GmbH
Watzmannstraße 1a - 81541 Munich
Germany
bruno.lorenz@namics.com

Versionhistory

Version	Date	Author	Description of changes
v0.1a	2012-01-04	Bruno Lorenz	Initial document
v0.2a	2012-01-05	Bruno Lorenz	Updated composer configuration settings

Contents

1 Installation

1.1 Requirements

During some functionality is only available since symfony 2.1 it is **necessary to have your project on a symfony 2.1 basis**. Some actions requires additional tools to do their work. **As long as this actions are part of the actionstack they will force you to have this tools installed and callable!** If you don't need this action and don't want to install the tools needed for it you have to configure a action stack without this specific action. See chapter 'Configuration' for more information about defining you own action stack.

1.2 Dependency management

Update your composer.json and add a new repository.

```
{
"type" : "vcs",
"url"  : "http://github.com/senuphtyz/TerrificExporterBundle"
}
```

After that add a new requirement to your project.

```
"senuphtyz/terrific-exporter-bundle" : "v2.*"
```

Now update your project using the composer.

```
php composer.phar update
```

This should now install alle necessary requirements for your project.

1.3 Tooling

There are a number of tools needed depending on your configuration and tasks.

- YUIDoc
- csshint
- jslint
- jpegoptim
- optipng
- advpng
- montage

- diff

It is necessary to have all tools within path, the exporter won't search for tools on your hardrive. So you have to setup your path variable depending on your os system correctly to have all tools within path's.

On Windows: http://www.computerhope.com/issues/ch000549.htm

On *nix/MacOSX: http://www.troubleshooters.com/linux/prepostpath.htm
To do a permanent change it is necessary to change /.bashrc or /.bash_profile depending on your os.

1.3.1 YUIDoc, jslint, csshint

YUIDoc, jslint and csshint are installed using Node.js. Just go to nodejs.org download the package fits for you operating system and install it.
After the installation is done open up a new commandline and install Node.js.

```
npm -g install yuidocjs jslint csshint
```

For further help and syntax for YUIDoc visit http://yui.github.com/yuidoc/.

1.3.2 jpegoptim, optipng, advpng, montage

Windows
Jpegoptim is currently not available on Windows systems.

Optipng can be retrieved from http://optipng.sourceforge.net/. Just download the Windows package unzip it no installation required.

Advpng or advancecomp can be fetched from http://advancemame.sourceforge.net/comp-download.html. The same just download and unzip.

Montage is part of the ImageMagick toolset. To install ImageMagick visit:
http://www.imagemagick.org/script/binary-releases.php

Unix/Linux
On Ubuntu/Debian based Linux it is possible to install jpegoptim directly using your package manager.

```
sudo apt-get install jpegoptim advancecomp optipng imagemagick
```

On RHEL/Fedora/Centos Linux you have to install jpegoptim from source, rest of the tools could be installed using yum. Download the current version from http://www.kokkonen.net/tjko/projects.html.

```
sudo yum install advancecomp optipng ImageMagick
```

jpegoptim

```
tar zxf jpegoptim-1.2.4.tar.gz
cd jpegoptim-1.2.4
./configure && make && make install
```

MacOSX
On MacOSX the easiest way to get the whole toolset is to install ImageOptim. This application contains all necessary image optimizing tools needed by the exporter.

http://imageoptim.com/

Montage is part of the ImageMagick toolset. To install ImageMagick visit:
http://www.imagemagick.org/script/binary-releases.php

1.3.3 diff

Windows
On windows there are a number of tools doing the same job as diff on *nixes. You can install a commandline version from diff with cygwin.
Unix/Linux
Normally diff should be installed on all *nixes. If not just install it using your packagemanager.

```
# Debian/Ubuntu:
sudo apt-get install diff

# RHEL/CentOS/Fedora:
sudo yum install diff
```

MacOSX
On MacOSX diff is already installed.

1.4 Setup an export environment

It is necessary to setup a new environment for you export. To setup an export environment just copy your app/config.yml to app/config_export.yml. Now you created a new environment called "export".

You can now configure the environment to your project needs. For further information visit http://symfony.com/doc/current/cookbook/configuration/environments.html.

2 Configuration

All configuration settings goes beyond a terrific_exporter node within your environment configuration file.

build_local_paths: (true/false)

If build_local_paths is enabled the exporter will change all urls within html and css files to match within the exported package.
See actions 'ExportAssets' and 'ExportViews'.

build_js_doc: (true/false)

Enables the export of a javascript documentation. The documentation is generated using YUIDoc.
See action 'BuildJSDoc'.

build_settings: (path)

This setting should target to a build.ini file. Within this file there are only settings for the projektname an versioning data.
See section 'build.ini'.

build_path: (path)

Has to target to a path which is the export target path.

export_with_version: (true/false)

Set to true if the exporter should build zips/folders with version numbers within its name.
See section 'build.ini'.

autoincrement_build: (true/false)

True if the exporter should increase the revision after each build.
See section 'build.ini'.

validate_js: (true/false)

Activates the validation of javascript. Validation is done using jshint.
See action 'ValidateJS'.

validate_css: (true/false)

Activate the validation of css. Validation is done using csshint.
See action 'ValidateCSS'.

optimize_image: (true/false)

Set to true to optimize images in the output directory. Optimization is done using jpegoptim, optipng and advpng.
See action 'OptimizeImages'.

export_views: (true/false)

Activates the export of the views marked with a @Export annotation.
See action 'ExportViews'.

export_modules: (true/false)

Activates the export of plain module html. The url within this modules are not rewriten even if the build_local_paths option is activated.
See action 'ExportModules'.

export_type: (string: folder/zip)

Set the export type if the export should be done as folder or as a zip.

build_actions: (list of objects)

These option allows to setup a build chain. Here you can append project related exporting tasks or change the builtin order.

- Terrific\ExporterBundle\Actions\ClearAction
- Terrific\ExporterBundle\Actions\BuildJSDoc
- Terrific\ExporterBundle\Actions\ValidateJS
- Terrific\ExporterBundle\Actions\ValidateCSS
- Terrific\ExporterBundle\Actions\ValidateModules
- Terrific\ExporterBundle\Actions\ValidateViews
- Terrific\ExporterBundle\Actions\GenerateSprites
- Terrific\ExporterBundle\Actions\ExportImages
- Terrific\ExporterBundle\Actions\ExportAssets
- Terrific\ExporterBundle\Actions\OptimizeImages
- Terrific\ExporterBundle\Actions\ExportModules
- Terrific\ExporterBundle\Actions\ExportViews
- Terrific\ExporterBundle\Actions\ExportChangelogs

See section 'Actions' or 'Extending' for further information.

pathtemplates: (set of string values)

The pathtemplates option allows you to customize you paths within your export package. All paths begin with a starting '/' each given directory will begin relative to the given export_path. So a value like '/img/common' will end up in '/exportpath/img/common'. The optional %module% variable within will be resolved by the PathResolver into a modulename. This variable only get matched in 'module_*' options.
It is possible to set paths for the following types of files:

- image: (default: '/img/common')
- font: (default: '/fonts')
- css: (default: '/css')
- js: (default: '/js')
- view: (default: '/views')
- changelog: (default: '/changelogs')
- diff: (default: '/changelogs/diff')
- module_image: (default: '/img/%module%')
- module_font: (default: '/fonts/%module%')
- module_css: (default: '/css/%module%')
- module_js: (default: '/js/%module%')
- module_view: (default: '/views/%module%')

See actions 'ExportModules', 'ExportAssets' and 'ExportViews'.

sprites: (list of objects)

Here you can setup sprite information. The exporter will build the sprites with the given data. See section 'GenerateSprites'.

changelog_path: (directory)

Set this value to a valid path which contains the changelogs for your project. If this folder doesn't exist no changelogs are appended. The value is also relative from the projectfolder like 'build_path' or 'build_settings'. Defaultvalue 'build/changelogs'

2.1 Example configuration

```
terrific_exporter:
build_local_paths:        true
build_js_doc:             true
build_settings:           "build/build.ini"
build_path:               "build/"
export_with_version:      false
autoincrement_build:      true
validate_js:              false
validate_css:             false
validate_html:            false
optimize_images:          true
export_views:             true
export_modules:           true
export_type:              folder

pathtemplates:
        image:     "/bilder/common23"
        font:      "/schriften"
        css:       "/styles"
        js:        "/scripts"
        view:      "/html"
        changelog: "/changelogs"
        diff:    "/changelogs/diff"

        module_image: "/module/%%module%%/bilder"
        module_font:  "/module/%%module%%/schriften"
        module_css:   "/module/%%module%%/styles"
        module_js:    "/module/%%module%%/scripts"
        module_view:  "/module/%%module%%/html"

sprites:
              - {
                directory: "PROD/internet_sprite_icons",
                target: "web/img/sprite_icons.png", item: { height: 50, width: 100 }
              }
```

2.2 YUIDoc

The exporter will use the yuidoc configuration file named yuidoc.json within the app/config directory. The syntax of the file could be read on the YUIDoc page http://yui.github.com/yuidoc/args/index.html.

2.3 csslint

Csslint will use a configuration if one is found under app/config, if there is no configuration named csslint.cfg the exporter will use its default configuration found within %kernel.root_dir%/vendor/senuphtyz/terrific-exporter-bundle/Terrific/ExporterBundle/Resources/config.

If you want to specify a different csslint.cfg you should take a copy from the default and change the settings within. Show a list of available options just enter the following command in your console.

```
csslint --list-rules
```

2.4 jshint

JShint will also use a configuration file named jshint.json, normally found within app/config. If no configuration file is found there the default within ....ExporterBundle/Resources/config is used.

See http://www.jshint.com/docs/ configuration settings.

3 build.ini

```
[version]
name=Terrific
major=0
minor=0
build=0
```

Normally the default build.ini will look like this. If you specify a build.ini with option 'build_settings' that is not available, the exporter will create it from the default. Depending on option 'autoincrement_build' the build number is incremented each export.

4 Usage

To startup an export use the following command.

```
app/console build:export --env=export --no-debug
```

4.1 Commandline options

–no-image-optimization

Overrides configuration setting optimize_images and starts the chain without optimizing Images. Skip actions:

- Terrific\ExporterBundle\Actions\OptimizeImages

–no-js-doc

Do not build a javascript documentation for this export run. This does skip the following actions:

- Terrific\ExporterBundle\Actions\BuildJSDoc

–no-validation

Ignores validation configuration and skip the following actions:

- Terrific\ExporterBundle\Actions\ValidateJS
- Terrific\ExporterBundle\Actions\ValidateCSS
- Terrific\ExporterBundle\Actions\ValidateModules
- Terrific\ExporterBundle\Actions\ValidateViews

–last-export=[directory]

To generate diffs between the last export and the current one, it is needed to give additional information to build this files. This parameter have to be set to the folder of the last export. If you don't add this information on your console call the following actions are skipped:

- Terrific\ExporterBundle\Actions\ExportDiffs

4.2 Annotations

@Export

To export a view it is necessary to annotate a controller method with this annotation. It is possible to control a set of options for each view directly within this annotation. This example is only valid if you don't have to export a localized version of a view. If no environment is given this view will exported in all environments.

```
@Export(
    name="viewname.html",
    environment="env1,env2,..."
)
```

@LocaleExport

Each view must have their own for each locale which should exported. To set different settings for each locale you have to use the @LocaleExport annotation. This annotation is used in combination with the @Export annotation. Setting up locale exportation will disable exporting the default language which means you have to annotate **all** locales that should exported. If no environment is given this view will exported in all environments.

Usage with @Export:

```
@Export(
    @LocaleExport(name="viewname_de.html", locale="de"),
    @LocaleExport(name="viewname_en.html", locale="en", environment="env1"),
    @LocaleExport(name="viewname_lv.html", locale="lv", environment="env2"),
    ....
)
```

5 Debugging

5.1 Configuration

```
services:
    terrific.formatter.line:
        class: Monolog\Formatter\LineFormatter
        arguments:
            - "[%%datetime%%] [%%level_name%%]: %%message%%\n"

monolog:
    handlers:
        file:
            type:  stream
            path:  %kernel.logs_dir%/%kernel.environment%.log
            level: debug
            formatter: terrific.formatter.line
```

Add this snippet to your export environment configuration. This snippet will activate logging to a file called like your environment beyond the app/logs directory. The loglevel debug will output a massive amount of logging from the exporter. Currently only the log file contains a list of validation errors for your css / js files. For more information about how to configure your monolog see http://symfony.com/doc/current/cookbook/logging/monolog.html.

6 Actions

All buildin actions are documented within these chapter. This maybe have some helpful information when you have to build your own action chain.

6.1 Terrific\ExporterBundle\Actions\ClearAction

This action simple just clear all data from the target directory. If configuration option 'export_with_version' is activated this action does simply nothing cause normally the export folder should not exists during startup.

6.2 Terrific\ExporterBundle\Actions\BuildJSDoc

BuildJSDoc will first test if yuidoc is callable. After that test it look for a suitable yuidoc.json file, if a file has been found the action starts the yuidoc command with the found configuration.

An example call could look like this, and is going to be called withing the current project folder.

```
yuidoc -c /data/symfony2-project/app/config/yuidoc.json
```

6.3 Terrific\ExporterBundle\Actions\ValidateJS

Javascript will be validated within this action. This action will get a list of all necessary javascript assets from the PageManager class. After retrieving this list it temporary removes the min filters and place the content for **each part** of the asset within a temporary file. After saving that file the jslint command will looking for a suitable configuration file an starts to verify the contents of the temp file.

An example call could look like this:

```
jshint --jslint-reporter --config /sf2-prj/app/config/jshint.json /tmp/ZIEASDZER
```

6.4 Terrific\ExporterBundle\Actions\ValidateCSS

Stylesheets will be validated using this actions. This action also retrieves a list of all potential used style assets from the PageManager class. After retrieving this list the minfilter will be removed for dumping the content **each part** the asset to a temp files.

Example for the csslint validation command:

```
csslint
    --format=lint-xml
    --errors=[from cfg]
    --warning=[from cfg]
    --ignore=[from cfg]
    /tmpdir/ASDFETZ
```

6.5 Terrific\ExporterBundle\Actions\ValidateModules

First of all this action has a additional requirement on the configuration. This configuration must have enabled the options 'validate_html' and 'export_modules'. If one of them is not activated this action will be skipped. After starting this action it retrieves a list of all module combinations (module <-> skin <-> view) from the PageManager class. The modules are exported without any HTML from the view, just the plain modules. After exporting this HTML to a temp file the action try's to send this file to the W3CValidator (http://validator.w3.org/). Its required to have a internet connection to start this action.

To generate a valid HTML ValidateModule uses a template file. This template file (module-template.tpl.html) can be replaced by placing a file with the same name in your app/config folder. Within this file just two variables are special '%MODULE_NAME%' and '%MODULE_CONTENT%'. Both variables are replaced with the corresponding content before sending to the W3CValidator.

6.6 Terrific\ExporterBundle\Actions\ValidateViews

This action handles the validation for the views just like the ValidateModules action. The difference between this both action are that ValidateViews requires other configuration options to start ('export_views' and 'validate_html'). The views also have all source code from the used modules within this view. This ValdationAction also uses the W3CValidator.

6.7 Terrific\ExporterBundle\Actions\GenerateSprites

GenerateSprites needs some more configuration than the other actions. Each sprite which should be generated must have an entry within the config file. If there is no configured sprite(s) the action will just be skipped. To generate a sprite this action uses the montage tool from the ImageMagick toolset. After retrieving a filelist for merging as a sprite the action will sort this files by name. This offers you the possibility to order you images within your sprite. Filenames like 0000_arrow.png are best practice. If this file is part of the export, which should be the normal usecase, this action has to **run before ExportImages**.

Example for a sprite generation call:

```
montage
    -mode Concatenate
    -tile x${height * count images}
    -geometry ${width}x${height}+0+0
    -bordercolor none
    -background none
    ${list of files} ${target}
```

6.8 Terrific\ExporterBundle\Actions\ExportImages

All images which are used within the exported views should be exported by ExportImages. To control which image is part of the export you have to encapsulate each image within the **Twig's asset()** function. **Imagepath's which are not generated using this function won't be part the export!** The images used within the css file(s) will also be exported but there is it not possible to control which should be part of the export. **This action have to run before OptimizeImages**.

6.9 Terrific\ExporterBundle\Actions\ExportAssets

Just simply do dumps of all used assets (css and javascript). It also retrieve a list of used assets from the PageManager and dump the assets to the exporting files. This time the exported assets are also get minified if its configured! If the 'build_local_paths' options is enabled the paths within the css files are change to match the exporting paths. **Paths within javascript files currently won't get changed!**

6.10 Terrific\ExporterBundle\Actions\OptimizeImages

All images within the export path are optimized. So the startup of the OptimizeImages action depends on the ExportImages action. After retrieving a list of all image files in the export the action will optimize file by file depending on the file extension. After running this command it will print the amount of bytes saved for each file and as total amount.

Example commands for optimizing images:

```
optipng -o7 /exportpath/img/picture.png

advpng -q -4 /exportpath/img/picture.png

jpegopim -q /exportpath/img/picture.jpg
```

6.11 Terrific\ExporterBundle\Actions\ExportModules

Retrieves a list of all module combinations (module <-> skin <-> view) and exports the HTML for all modules in a seperate folder within the exporting path.

6.12 Terrific\ExporterBundle\Actions\ExportViews

Retrieves a list of all controller methods which are annotated with @Export. If there is no such a method the exporter simply won't just export anything.

6.13 Terrific\ExporterBundle\Actions\ExportChangelogs

This action just copies your changelog files (file_extensions: log, txt, md) to the exporting path. To append all changelogs create a folder under build/changelogs or setup your 'changelog_path' option. If this folder is not available no changelogs will appended.

6.14 Terrific\ExporterBundle\Actions\ExportDiffs

ExportDiffs will generate diff files for each view and append it to your export. The exporter needs an additional information given as console parameter to find the last export. This argument is called '–last-export=[directory]' must be set to the last export package. To customize your diff path see chapter 'Configuration'.

7 Extending

This chapter describes how to build your own action and how to use the available infrastructure get retrieve your results.

7.1 Components & Services

7.1.1 BuildOptions

ConfigFinder is just a simple class that holds data hierarchical data. After startup the BuildOptions class will get initialized with a file given in the project configuration. When initalizing is complete you can access all data from the *.ini file like a array. A dot is used to do step downwards in the hierarchy. Saving is normally done in the destructor of this service. If you're having trouble to wait on the destructor call you can manually run the save() function. This service resides within the servicecontainer id 'TODO: FILL IN ID'.

```
$majorVersion = $buildOptions["version.major"];
$buildOptions["version.major"] = 2;
```

7.1.2 ConfigFinder

This service helps you to find configuration files. It will first look for a configuration file within the app/config directory and after that in the bundle's resources/config directory. You will just receive a path for the file which is first found. This service resides within the servicecontainer id 'TODO: FILL IN ID'.

```
$configFile = $configFinder->find('jshint.json');
```

7.1.3 Log

Just a simple logging wrapper for console output. During the fact it is a static class you can simply use it without initializing it or retrieve it from the servicecontainer. After the ExportCommand the Log class is feeded with all data needed to do outputs on the console. This wrapper is also helpful to do hierarchically output using the blkstart() and blkend() functions.

```
Log::info("Console info");
Log::blkstart(); // intends the output for info/err/warns until blkend() called
    Log::err("Console error");
    Log::warning("Console warning")
Log::blkend(); // safety always reset your hierarchy
```

7.1.4 PathResolver

This service just have two tasks first look for a specific asset in your path's, second resolve a new path (export path) for the given asset url. To build an export path this class takes all configuration options from the pathtemplates option from the configuration. This service resides within the servicecontainer id 'TODO: FILL IN ID'.

```
$asset = $pathResolver->find('filename', 'path_with_file');
$exportPath = $pathResolver->resolve('/terrificmoduleTest/img/testimage.png');
```

7.1.5 TempFileManager

It's a manger that handles temporary files. Normally you receive some output from the project and just safe a temp file for it than do further operations until the final results are received. This service helps you to always get a clean file with no content and removes all files during service destruction which were created during the whole export run. To prevent the Manager from removing the temporary content you can set the keepTemp property to true. This service resides within the servicecontainer id 'TODO: FILL IN ID'.

```
$content = "some content here";
$tempFileMgr->putContent($content); // returns a complete path to a file
```

7.1.6 TimerService

This service just helps get the span between two specific points of time. You can receive the difference between these points. The service resides within the servicecontainer id 'TODO: FILL IN ID'.

```
$timer->lap("START-doSomething");
// do something
$timer->lap("END-doSomething");

$time = $timer->getTime("START-doSomething", "END-doSomething");
```

7.1.7 W3CValidator

A service which uses the online w3c validator to validate content or complete files against the online validator. Both validation commands will return a ValidationResult object. This service resides within the servicecontainer id 'TODO: FILL IN ID'.

```
$w3val->validateFile('/path/to/file');
$w3val->validate('html content here');
```

7.1.8 PageManager

PageManager is the complexest service within the exporter. It manages to retrieve get all routes from the router and all assets which are used in the twig templates. During the huge amount of time PageManager needs to initialize it caches most of the data and do a lazy initalization after the first method is called, so the first answer may need some time depending on the amount of views/routes which are configured within your frontend project. This service resides within the servicecontainer id 'TODO: FILL IN ID'.

```
$pageManager->findRoutes(true);
```

7.2 Helpers

The exporter sometimes do specific things more than one time. This helpers are used to concentrate specific things to a concrete class. **All helper classes are static so you can just call the methods within.**

7.2.1 AsseticHelper

Helper contains functions to work with assetic and asset specific tasks. Currently the AsseticHelper can export fonts and images from css content and removes minification filter from assetic.

7.2.2 FileHelper

This helper helps you with file identification like it is a stylesheet / javascript. Other functionalities are removing query params from urls or create whole paths without having to create the parent folders.

7.2.3 NumberHelper

Currently this helper just can format byte information to the highest possibly unit.

7.2.4 OSHelper

At the moment OSHelper just can give an answer if the current running OS isWin().

7.2.5 ProcessHelper

The ProcessHelper has two specific tasks. First check a command if its available. Second start the command. The command string will be retrieved using the ProcessBuilder class from symfony 2.1.

7.2.6 StringHelper

StringHelper currently just helps you to build working filenames and strip chars like & or / out from the filename.

7.2.7 XmlLintHelper

This helper is a verify specific helper. It helps to convert a linter xml format to ValidationResult objects.

7.2.8 GitHelper

GitHelper help you to retrieve informations from git archives using the git commandline executeable. This helper is currently not in use and maybe used in future builtin actions or project based actions.

7.3 Build own actions

To write you own action you have at least to implement the IAction interface. This interface will guarantee that all methods that are called by the ExportCommand are available.

```
public function setWorkingDir($directory);
public function run(OutputInterface $output, $params = array());
public function isRunnable(array $params);
public static function getRequirements();
```

First the ExportCommand will retrieve all requirements from the action stack. After checking all requirements it looks for the necessary to run this action with calling the isRunnable() method. When all requirements are fit and this action should be run the ExportCommand will run your action.

To make it more easy to implement your own action you can just use a predefined abstract class which already have some useful functions. There are two of them AbstractAction and AbstractExportAction. AbstractAction contains helpers like a reference to the servicecontainer and initializes a filesystem object. AbstractExportAction contains all function the AbstractAction class contains and adds a helper function for saving files to a directory.

To get further information on implementing own actions just take a look on the api documentation.

8 Known Problems

During the complexity of a project and this exporter there will always some constructs which are problematic and the exporter won't export. Some of the currently known limitations are written down here.

8.1 Module command with parameter set by the controller

Constructs like *tc.module(module,skins,views)* which needs data from the controller will currently not work. This is due to the fact that PageManager generates a list of assets from the templates directly and won't call the controller method for this.

8.2 Methods that returns a response rather than connect a view with @Template()

8.3 Assets with the path and name but different content within the same export

If two or more views use a asset named default.css but with different content, the export won't understand that this files named the same but different content so it will tell you that there is already a file with this name and stop exporting.

8.4 Asset usage commands within parent templates

At the moment the PageManager generates a list of all views with all used assets. Currently the PageManager ignores twig's block overrides. This means the exporter will also export assets within overriden unused blocks.

8.5 Templates engines other than Twig

Currently not supported during the usage of the twig template lexer to find used assets within the templates.

References

[1] How to set the path and environment variables in Windows
http://www.computerhope.com/issues/ch000549.htm

[2] Adding a Directory to the Path
http://www.troubleshooters.com/linux/prepostpath.htm

[3] YUIDoc
http://yui.github.com/yuidoc/

[4] YUIDoc config syntax
http://yui.github.com/yuidoc/args/index.html

[5] optiPNG
http://optipng.sourceforge.net/

[6] advPNG
http://advancemame.sourceforge.net/comp-download.html

[7] ImageMagick
http://www.imagemagick.org/script/binary-releases.php

[8] jpegOptim
Installing it under RHEL / CentOS / Fedora
http://www.kokkonen.net/tjko/projects.html

[9] ImageOptim
http://imageoptim.com/

[10] Symfony environment
http://symfony.com/doc/current/cookbook/configuration/environments.html

[11] JSHint
http://www.jshint.com/docs/

[12] Monolog
http://symfony.com/doc/current/cookbook/logging/monolog.html

[13] W3CValidator
http://validator.w3.org/

[14] Diff
http://en.wikipedia.org/wiki/Diff

[15] Cygwin
http://www.cygwin.com/

[16] NodeJS
http://nodejs.org/

[17] CSSLint
http://csslint.net/