clover Documentation

Release 0.1

nate skulic

January 18, 2014

Contents

1 Features **3**

2 Installation **5**

3 Contents **7**

3.1 Installation 7
3.2 Usage 7
3.3 Configuration 9
3.4 Javascript Testing 11
3.5 Internationalization 12
3.6 Troubleshooting 14
3.7 Changes 14
3.8 Indices and tables 14

Clover is a build tool that dynamically compiles JavaScript and Closure Template code.

CHAPTER 1

Features

- Simplifies development with the closure compiler, library and templates.
- Compiles javascript and soy templates on the fly
- Closure unit testing framework support
- Command line and web based test runners. * Selenium support to run tests in various drivers.
- Javascript internationalization support
- Sourcemap support
- Displays closure warnings and errors to the browsers console log.
- Allows a single configuration file defining the entire compilation; replacing build scripts/etc.

CHAPTER 2

Installation

Using pip:

```
pip install clover
```

For more ways of installation visit *Installation*

CHAPTER 3

Contents

3.1 Installation

3.1.1 Binary installation

Note: Clover uses java (and JPype). Make sure your JAVA_HOME environment variable is set before installation and during operation.

Installing from pip:

```
pip install clover
```

3.1.2 Installing from source

Checkout from source:

```
hg clone https://code.google.com/p/clover/
```

Build:

```
make
```

Clover can be run with the included shell script named `clover`.

3.1.3 Installing into the system

Follow the steps as outlined in *Installing from source*.

Change into the python directory and use `python setup.py install` to install the package.

3.2 Usage

Clover uses the YAML format for configuration files. At a minimum, the configuration file declares the list of files that should be compiled (*inputs option*), where their dependencies can be found (*paths option*) and *various other options*.

The following as an example of a minimal configuration file:

```
id: example
output: compiled.js
paths:
- .
inputs:
- main.js
```

Once clover starts, it will discover new JavaScript and Soy files added under the specified paths.

Tip: It is often easiest to declare a single file for inputs, such as main.js.

goog.require() and goog.provide() should be used as appropriate to transitively include the desired libraries for the application.

This avoids the need to restart clover and makes the insertion point for your application unambiguous.

3.2.1 For development

When clover is started in server mode, it takes the path to one or more config files as input:

```
clover serve clover.yaml
```

The primary way to access the output of the clover server is by making a GET request. By default, clover runs on port 2323.

Loading the following URL in a browser will return the compiled code for the configuration with the specified id:

```
http://localhost:2323/configs/example/output.js
```

This means that when developing your web application, you will need a <script> tag whose src attribute refers to the URL that is shown above:

```
<script src="http://localhost:2323/configs/example/output.js"></script>
```

Note: You will need to add logic to your web server to load JavaScript from clover in development mode, but to load JavaScript from wherever your other static content is hosted in production mode.

3.2.2 For testing

Refer to *javascript-testing* for how to write and run unit tests.

3.2.3 For Production

clover is designed to produce a static JavaScript file that can be served in production.

The *output-file option* allows one to specify an output destination.

If no output file is specified, the compilation will be printed to standard output.

The following command will compile to the output specified:

```
clover build clover.yaml <config id>
```

3.3 Configuration

3.3.1 Format

The configuration file is in yaml format.

Multiple configurations can be specified in one file. For example:

```
id: config1
---
id: config2
inherits: config1
---
id: config3
inherits: config1
```

3.3.2 Configuration options

alias-all-strings

Aliases all string literals to global instances, to avoid creating more objects than necessary.

checks

A map of Compiler check-levels. Values can be one of WARNING, ERROR or OFF.

For example:

```
checks:
    accessControls: ERROR
    visibility: ERROR
```

Available checks: accessControls visibility checkRegExp checkTypes checkVars deprecated fileoverviewTags invalidCasts missingProperties nonStandardJsDocs undefinedVars aggressiveVarCheck brokenClosureRequiresLevel checkGlobalNamesLevel checkGlobalThisLevel checkMissingGetCssNameLevel checkMissingReturn checkProvides checkRequires checkUnreachableCode reportMissingOverride

closure-library

True to enable closure library inclusion.

debug

Equivalent to the command-line –debug flag for the Closure Compiler. Defaults to false.

default-externs

Whether the default externs should be used by the compiler.

define

An object literal that contains a mapping of variables in the JavaScript code that are annotated with @define (indicating that they can be redefined at compile time) to the values that should be substituted. The following should be specified to set goog.DEBUG to false at compile time:

```
define:
  goog.DEBUG: false
```

Note that these compile-time defines will only take effect when the code is compiled in either SIMPLE or ADVANCED modes.

id

Every config must have an id.

The id must be unique among the configs being served by clover because the id is a parameter to every function in the clover REST API.

externs

Files that contain externs that should be included in the compilation.

By default, these will be used in addition to the default externs bundled with the Closure Compiler.

There are also externs for third party libraries, such as jQuery and Google Maps, that are bundled with the Closure Compiler but are not enabled by default.

These additional extern files can be seen in the Closure Compiler's contrib/externs directory. Such externs can be included with a // prefix as follows:

```
externs:
- //jquery-1.5.js
- //google_maps_api_v3.js
- //chrome_extensions.js
```

inputs

Input files to be compiled.

Each input file and its transitive dependencies will be included in the compiled output.

mode

Compilation level, which must be one of "DEPS", "RAW", "WHITESPACE", "SIMPLE", or "ADVANCED".

The default value is "DEPS".

DEPS mode simply includes a base.js (and the generated deps.js).

RAW mode includes all inputs (and transitive requirements) for the compilation.

WHITESPACE mode generates a concatenated whitespace.

SIMPLE and ADVANCED is equivalent to Closure compilers SIMPLE optimizations, respectively.

warnings

Sets how warnings are treated.

When set to true, warnings will be reported as errors.

output

If specified, when the build command is used, clover will write the compiled output to this file rather than standard out.

output-sourcemap

If specified, when the build command is used, clover will write the compiled output to this file rather than standard out.

paths

Files or directories where the transitive dependencies of the inputs can be found.

pretty-print

Equivalent to the command-line –formatting=PRETTY_PRINT flag for the Closure Compiler. Defaults to false.

strip-name-suffixes

Name suffixes that determine which variables and properties to strip.

This can be useful for removing loggers:

```
strip-name-suffixes:
- logger_
```

strip-name-prefixes

Name prefixes that determine which variables and properties to strip.

strip-type-prefixes

Qualified type name prefixes that determine which types to strip.

test-excludes

By default, all files that end in _test.js under the paths directories are included in the test runner that runs all of the JS unit tests. This option can be used to specify subpaths of paths that should be excluded from testing.

> For example, if the closure-library option is used to specify a custom Closure Library, then it is likely that third_party/closure will be specified as a path to include utilities such as goog .async.Deferred. Including such a directory will also include closure/goog/dojo/dom/query_test.js, which fails when run by clover because its corresponding query_test.html file includes a hardcoded path to

base.js that cannot be loaded by clover. If this is a problem, then third_party/closure should be included as an argument to test-excludes.

See the section on testing for more details.

translations

An xtb translations file.

3.4 Javascript Testing

The Closure Library contains a framework for jsunit-style unit tests.

Clover features web based and command line test runners.

3.4.1 Writing a Test case

The convention for unit testing a file of JavaScript code is to create the test case in a file of the same name, except with a _test.js suffix.

For example, to create a unit test for goog.string, which is defined in string.js, the test code should be written in string_test.js. Here is a code sample from such a string_test.js file:

```
goog.require('goog.string');
goog.require('goog.testing.jsunit');

var testStartsWith = function() {
  assertTrue('Should start with \'\'', goog.string.startsWith('abcd', ''));
  assertTrue('Should start with \'ab\'', goog.string.startsWith('abcd', 'ab'));
};
```

3.4.2 Running a individual Test (via the browser)

1. Visit the configuration listing page at http://localhost:2323/.
2. Follow the "List Test Files" link for the configuration you want to test.
3. There should be a link for each test file. Follow the link to the test file to run a test runner on that file.
4. Passing tests are displayed in green while failing tests are displayed in red.

3.4.3 TODO: Running a test via the command line

The command line test runner uses a webdriver instance to run and check your tests.

The server must be started before running tests:

```
clover serve <config file>
```

The following command should run tests using the server:

```
clover test <config id> <path to test file>
```

3.4.4 Running All Tests

To run all of your tests visit the configuration listing page and click "Run Tests" for a configuration.

This will bring up a goog.testing.MultiTestRunner configured to runs all of your tests.

3.4.5 Writing Asynchronous Tests

Globals named *AsyncTestCase* and *DeferredTestCase* are provided.

These are the instance of goog.testing.DeferredTestCase that are used to run your page.

To override this use the following after at the head of your test file:

```
var AsyncTestCase = new goog.testing.AsyncTestCase.createAndInstall(CLOVER_TEST_TITLE)
```

The global CLOVER_TEST_TITLE is the test title provided by the clover server (the file path of the test).

3.5 Internationalization

Clover has features to help developers produce translated versions of their applications.

Clover and the underlying closure tools use the XMB message format.

The XMB format is a key-value pair list. It has a mechanism for named placeholders, with descriptions and examples.

3.5.1 Extracting messages

Clover has a command to extract messages from the application:

```
clover messages <config> <id>
```

This command will extract all messages (defined using goog.getMsg) from the application to standard output.

3.5.2 General workflow

You should have a configuration file set up and a directory to store translation files (eg, "locale").

1. Extract all of the applications messages to an xtb:

   ```
   clover messages <config> <id> -o locale/en.xtb
   ```

   To reduce friction when translating applications, you may use tools to merge the extracted XTB into an existing XTB.
2. Copy and translate the generated xtb file to the target languages.
3. Use clover's *translations* to select the xtb file used during compilation.

3.5.3 Example

Create a directory for the following example.

1. Create a subdirectory *locale*.

2. Create a test.js:

```
/** @desc context menu - make a duplicate of the selected block */
MyProject.MSG_DUPLICATE_BLOCK = goog.getMsg("Duplicate");
```

3. Create a clover.yaml:

```
paths:
- .

inputs:
- test.js

output: build/app.js
translations: locale
```

4. Create a clover-fr.yaml:

```
inherits: clover.yaml
language: fr
output: build/app-fr.js
```

5. Extract the messages:

```
clover messages clover.yaml -o locale/en.xtb
```

6. Copy the generated locale/en.xtb to locale/fr.xtb.

7. Translate the <*translation*> element in locale/fr.xtb; replacing "Duplicate" with "Dupliquer". The file should look like this (the translation id will differ):

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE translationbundle SYSTEM "translationbundle.dtd">
<translationbundle lang="fr">
  <translation id="5213219513082471002">Dupliquer</translation>
</translationbundle>
```

8. Compile the application with the french configuration file:

```
clover build clover.yaml
```

The translated application will be in *build/app-fr.js*. All messages in the file will be replaced with the french versions specified in *locale/fr.xtb*.

3.5.4 Complex messages

Placeholder elements offered by the XMB are limited to simple value replacement. Messages must correspond 1:1 with those of English. In addition, goog.getMsg calls are compiled to strings, making dynamic replacement impossible.

For more complex patterns, consider using the goog.i18n.MessageFormat class. passing a message (using goog.getMsg) as the *pattern* argument.

The MessageFormat format is described in more detail at http://userguide.icu-project.org/formatparse/messages.

3.6 Troubleshooting

3.6.1 Difficulty installing Jpype

Please refer to https://github.com/originell/jpype

Later versions of clover will likely drop the dependency. Patches welcome :-)

3.6.2 RAW mode: Undefined GET errors

Adblock software is known to have issues with clover's raw mode. Disable it or whitelist the domain you are developing.

3.7 Indices and tables

- *genindex*
- *search*