CODING STANDARDIZATION

Coding standardization is not about what is right or wrong. It is simply a way to collaborate in a group setting and having a standard by which all agree to and follow. The standardization allows all code to be designed, written, and laid out the same to make it easier for developers to switch from one file to another.

These are some general conventions. Please refer to :
https://gist.github.com/indexzero/5368926
for javascript specifics.

Naming Conventions

Class Names

Class names should be nouns, in mixed case with the first letter of each internal word capitalized. Class names should be simple and descriptive. Use whole words-avoid acronyms and abbreviations. Examples:

```
class States;

class GameHandler;
```

Method Names

Methods should be verbs, in mixed case with the first letter lowercase, with the first letter of each internal word capitalized. Examples:

```
run();

getBackground();
```

Variable Names

Except for variables, all instance, class, and class constants are in mixed case with a lowercase first letter. Internal words start with capital letters. Variable names should not start with underscore _ or dollar sign $ characters, even though both are allowed.

Variable names should be short yet meaningful. The choice of a variable name should be mnemonic- that is, designed to indicate to the casual observer the intent of its use. One-character variable names should be avoided except for temporary "throwaway" variables. Common names for

temporary variables are i, j, k, m, and n for integers; c, d, and e for characters. Example:

```
String tempWord;
```

Constant Names

The names of variables declared class constants and of ANSI constants should be all uppercase with words separated by underscores ("_"). (ANSI constants should be avoided, for ease of debugging.) Example:

```
final int MAX_COUNTDOWN = 6;
```

Line Length

Avoid lines longer than 80 characters, since they're not handled well by many terminals and tools.

Wrapping Lines

When an expression will not fit on a single line, break it according to these general principles:

Break after a comma.

Break before an operator.

Prefer higher-level breaks to lower-level breaks.

Align the new line with the beginning of the expression at the same level on the previous line.

If the above rules lead to confusing code or to code that's squished up against the right margin, just indent 8 spaces instead.

Indenting

Indents should be a standard tab. (4 spaces)
Indents are required after each opening brace ({) and are terminated immediately before the closing brace.

Braces

The opening brace ({) should be placed at the end of the line immediately preceding the block of code it contains. The closing brace (}) should be the only character on the line following the block of code it contains, and should be placed in line with the preceding block of code; that is, its position should be reduced by one.

Exception Handling / Logging

***Consult Team

Commenting

Each file should have a block header that includes:

Filename: File.example
Creator: Cory Clow
Description: This is an example file that…
Change history:
15.01.2015 / Kyle Pineau

When a change is made to the file, a comment should precede it with the author, date, and description as well.

Commenting is very important for group collaboration, so when in doubt, add more comments. Adhere to standards that we have used throughout the CS program.

Updating Project Plan:

When members feel the need to update the project plan, they should notify the group using the group chat function.

Updating the Sprint Log:

When members change the sprint log, it should be reflected in the project plan as well.

Please refer to the following link for Javadoc guidelines:

Javadoc Guidelines

AngularJS Git Commit Message Conventions

Goals

- allow generating CHANGELOG.md by script
- allow ignoring commits by git bisect (not important commits like formatting)
- provide better information when browsing the history

Generating CHANGELOG.md

We use these three sections in changelog: **new features**, **bug fixes**, **breaking changes**.
This list could be generated by script when doing a release. Along with links to related commits.
Of course you can edit this change log before actual release, but it could generate the skeleton.

List of all subjects (first lines in commit message) since last release:
>> git log <last tag> HEAD --pretty=format:%s

New features in this release
>> git log <last release> HEAD --grep feature

Recognizing unimportant commits

These are formatting changes (adding/removing spaces/empty lines, indentation), missing semi colons, comments. So when you are looking for some change, you can ignore these commits - no logic change inside this commit.

When bisecting, you can ignore these by:
>> git bisect skip $(git rev-list --grep irrelevant <good place> HEAD)

Provide more information when browsing the history

This would add kinda “context” information.
Look at these messages (taken from last few angular’s commits):

- Fix small typo in docs widget (tutorial instructions)
- Fix test for scenario.Application - should remove old iframe
- docs - various doc fixes
- docs - stripping extra new lines
- Replaced double line break with single when text is fetched from Google
- Added support for properties in documentation

All of these messages try to specify where is the change. But they don't share any convention...

Look at these messages:

- fix comment stripping
- fixing broken links
- Bit of refactoring
- Check whether links do exist and throw exception
- Fix sitemap include (to work on case sensitive linux)

Are you able to guess what's inside ? These messages miss place specification...
So maybe something like parts of the code: **docs**, **docs-parser**, **compiler**, **scenario-runner**, …

I know, you can find this information by checking which files had been changed, but that's slow. And when looking in git history I can see all of us tries to specify the place, only missing the convention.

Format of the commit message

> **<type>(<scope>): <subject>**
> **<BLANK LINE>**
> **<body>**
> **<BLANK LINE>**
> **<footer>**

Any line of the commit message cannot be longer **100 characters**! This allows the message to be easier to read on github as well as in various git tools.

Subject line

Subject line contains succinct description of the change.

Allowed <type>

- feat (feature)
- fix (bug fix)
- docs (documentation)
- style (formatting, missing semi colons, …)
- refactor
- test (when adding missing tests)
- chore (maintain)

Allowed <scope>

Scope could be anything specifying place of the commit change. For example $location, $browser, $compile, $rootScope, ngHref, ngClick, ngView, etc...

<subject> text

- use imperative, present tense: “change” not “changed” nor “changes”
- don't capitalize first letter
- no dot (.) at the end

Message body

- just as in <subject> use imperative, present tense: “change” not “changed” nor “changes”
- includes motivation for the change and contrasts with previous behavior

http://365git.tumblr.com/post/3308646748/writing-git-commit-messages
http://tbaggery.com/2008/04/19/a-note-about-git-commit-messages.html

Message footer

Breaking changes

All breaking changes have to be mentioned in footer with the description of the change, justification and migration notes

BREAKING CHANGE: isolate scope bindings definition has changed and
the inject option for the directive controller injection was removed.

To migrate the code follow the example below:

Before:

```
scope: {
  myAttr: 'attribute',
  myBind: 'bind',
  myExpression: 'expression',
  myEval: 'evaluate',
  myAccessor: 'accessor'
}
```

After:

```
  scope: {
    myAttr: '@',
    myBind: '@',
    myExpression: '&',
    // myEval - usually not useful, but in cases where the expression is assignable,
you can use '='
    myAccessor: '=' // in directive's template change myAccessor() to myAccessor
  }
```

  The removed `inject` wasn't generaly useful for directives so there should be no code using it.

Referencing issues

Closed bugs should be listed on a separate line in the footer prefixed with "Closes" keyword like this:

Closes #234

or in case of multiple issues:

Closes #123, #245, #992

Examples

feat($browser): onUrlChange event (popstate/hashchange/polling)

Added new event to $browser:
- forward popstate event if available
- forward hashchange event if popstate not available
- do polling when neither popstate nor hashchange available

Breaks $browser.onHashChange, which was removed (use onUrlChange instead)

fix($compile): couple of unit tests for IE9

Older IEs serialize html uppercased, but IE9 does not...
Would be better to expect case insensitive, unfortunately jasmine does
not allow to user regexps for throw expectations.

Closes #392
Breaks foo.bar api, foo.baz should be used instead

feat(directive): ng:disabled, ng:checked, ng:multiple, ng:readonly, ng:selected

New directives for proper binding these attributes in older browsers (IE).
Added coresponding description, live examples and e2e tests.

Closes #351

style($location): add couple of missing semi colons

docs(guide): updated fixed docs from Google Docs

Couple of typos fixed:
- indentation
- batchLogbatchLog -> batchLog
- start periodic checking
- missing brace

feat($compile): simplify isolate scope bindings

Changed the isolate scope binding options to:
 - @attr - attribute binding (including interpolation)
 - =model - by-directional model binding
 - &expr - expression execution binding

This change simplifies the terminology as well as
number of choices available to the developer. It
also supports local name aliasing from the parent.

BREAKING CHANGE: isolate scope bindings definition has changed and
the inject option for the directive controller injection was removed.

To migrate the code follow the example below:

Before:

```
scope: {
 myAttr: 'attribute',
 myBind: 'bind',
 myExpression: 'expression',
 myEval: 'evaluate',
 myAccessor: 'accessor'
}
```

After:

```
scope: {
 myAttr: '@',
 myBind: '@',
 myExpression: '&',
 // myEval - usually not useful, but in cases where the expression is assignable, you can use '='
 myAccessor: '=' // in directive's template change myAccessor() to myAccessor
}
```

The removed `inject` wasn't generaly useful for directives so there should be no code using it.