CoffeeScript code guidelines

0. Main techniques and grammar

http://coffeescript.org/

1. Naming Conventions

1.1 All names should be written in English.

```
fileName
NOT: fr2VizCartImpl
```

1.2 Names representing types (classes, namespaces) must be in mixed case starting with upper case.

```
ClassName, StructureName, NamespaceName
NOT: classname, className, Class_Name
```

1.3 Variable names must be in mixed case starting with prefix consists of 2 or 3 letters
First letter defines variable's area. c - class member, `i` - instance member, `a` - method argument (or item in `for-in and `for-of` loops), v - local variable
Second letter is used if variable is pointer [p]
Third letter defines variable's type [t]
t - type identifier

- m : for method (function) type
- n : for integer type
- f : for float type
- b : for boolean type
- s : for string value
- d : for date value
- l : array or array like object and **name should be in the plural**
- h : hash (associative array / simple object) or map or set
- o : instance of some class

1.3.1 Instance attribute accessor methods naming
i[p][t]InstanceAttributeName = Symbol '<InstanceAttributeName>'

```
class Person extends CoreObject
  ipsFirstName      = Symbol 'firstName'
  ipsLastName       = Symbol 'lastName'
  ipmCalculateSize  = Symbol 'calculateSize'
```

```
@defineAccessor   String, ipsFirstName
@defineSetter     String, ipsLastName

# example of private method definition
@instanceMethod ipmCalculateSize, ->

constructor: (aOptions={})->
  @[ipsFirstName]    = aOptions.firstName if aOptions.firstName?
  @[ipsLastName]     = aOptions.lastName if aOptions.lastName?
```

1.3.2 Method arguments
a[t]Argument
`adFirstParam`

1.3.3 Local variables
v[t]LocalVariable
`vnMaxHeight`

1.4 The prefix *`is`* should be used for boolean methods.
isSet, isVisible, isFinished, isFound, isOpen

There are a few alternatives to the *`is`* prefix that fit better in some situations. These are the *`has`*, *`can`* and *`should`* prefixes:
hasLicense ->
canEvaluate ->
shouldSort ->

1.5 Named constants (including enumeration values) must be all uppercase using underscore to separate words.
`MAX_ITERATIONS, COLOR_RED, PI`
`NOT: max_Iterator, min_value, COLORRED`

1.6 Names representing methods or functions must be verbs and written in mixed case starting with lower case.
`defineName(), computeTotalWidth()`
`NOT: Name(), VALUE(), ComputeTotalWidth()`

1.7 `defineGetter`, `defineSetter` and `defineAccessor` methods must be in CoreObject class

```
class CoreObject
  @defineGetter: (Class, aName, aGetter)->
    if aName.constructor.name is 'Symbol'
```

```
    vSymbol = aName
    [vSource, v1, aName, v2] = String(vSymbol).match /(^.*\()(.*)(\)$)/
  else
    vSymbol = Symbol.for aName
  aGetter ?= -> @[vSymbol]
  @::__defineGetter__ aName, aGetter
  return

@defineSetter: (Class, aName, aSetter)->
  if aName.constructor.name is 'Symbol'
    vSymbol = aName
    [vSource, v1, aName, v2] = String(vSymbol).match /(^.*\()(.*)(\)$)/
  else
    vSymbol = Symbol.for aName
  aSetter ?= (aValue)->
    if aValue.constructor isnt Class
      throw new Error 'not acceptable type'
      return
    @[vSymbol] = aValue
  @::__defineSetter__ aName, aSetter
  return

@defineAccessor: (Class, aName, aGetter, aSetter)->
  @defineGetter Class, aName, aGetter
  @defineSetter Class, aName, aSetter
  return
```

2. Files and Class example

2.1 File name rules must be based on a framework rules.

2.2 All definitions should be placed in some standard segments in source files.

upload.coffee :

```
#-- dependencies segment --#
_               = require 'lodash'
joi             = require 'joi'
#-------------------------#

#-- class definition segment --#
class Basis::Upload extends FoxxMC::Model  # with using namespaces
  ipsFirstName      = Symbol 'firstName'  # pointer definition
  ipsLastName       = Symbol 'lastName'  # pointer definition
```

```
  ipmCalculateSize  = Symbol 'calculateSize'  # pointer definition

#-----------------------------#

#-- class attributes segment --#
@_sRootPath: "#{__dirname}/../../"
#-----------------------------#

#-- call major class methods (including mixins f.e.) segment --#
@include Basis::CheckSessionsMixin
@include Basis::CheckPermissionsMixin
#-------------------------------------------------------------#

#-- class methods segment --#
@defineAttachmentSchema: ->
#--------------------------#

#-- instance attributes segment --#
Model: Basis::Upload
#--------------------------------#


#-- defining instance attributes by using class methods segment --#
  @attr 'description',  joi.string().empty(null).default(null)
  @attr 'attachments',  joi.any().empty(null).default(null)
  @attr 'metadata',     joi.any().empty(null).default(null)
  @attr 'kind',         joi.string().required()
  @attr 'aspectRatio',  joi.string().empty(null).default(null)

  @belongsTo 'owner',   joi.string().empty(null).default(null),
    model: 'user'
    attr: '_owner'
  @belongsTo 'space',
joi.string().empty(null).empty('').default('_default'),
    attr: '_space'
#-------------------------------------------------------------------#

#-- call other class methods (define chains, hooks f.e.) segment --#
@chains ['add_attachment']

@initialHook 'checkSession'
@initialHook 'checkPermission'
```

```
@beforeHook 'beforeAddAttachment', only: ['add_attachment']
@beforeHook 'beforeUploadUpdate', only: ['update']

@finallyHook 'itemDecorator', only: ['add_attachment']

#-----------------------------------------------------------------------#

#-- instance methods segment --#
serializeForClient: ({withAttachmentData} = {}) ->
createImageFormats: (preset = '16:9') ->
updateAttachments: ->
#------------------------------#

#-- defining instance methods by using class methods segment --#
@action 'add_attachment', ->
@swaggerDefinition 'add_attachment', (endpoint) ->
@instanceMethod 'someMethod', ->
#------------------------------#

#-- exports segment (must be last definition in the file) --#
module.exports = Basis::Upload.initialize()
#--------------------#
```

3. Statements

3.0 Main rules

3.0.1 Main line rule.
One line with some code must be less than or equal 80 literals

3.0.2 Main code rule.
All expressions must be formatted for lite reading

3.0.3 Main brackets rule.
Use brackets if it really need

3.0.4 Main `return` rule.
Use `return` if it really need

3.0.5 Destructuring Assignment
use destructuring everything

```
{poet: {name, address: [street, city]}} = futurists

[city, temp, forecast] = weatherReport "Berkeley, CA"

class Animal
  constructor: (@name) ->

  someMethod: (aName, aAge)-> {aName, aAge}

[open, contents..., close] = tag.split("")
```

3.0.6 Main String Interpolation rule.
use string interpolation everything

```
quote  = "A picture is a fact. -- #{ author }"

sentence = "#{ 22 / 7 } is a decent approximation of π"

upperCaseExpr = (textParts, expressions...) ->
  textParts.reduce (text, textPart, i) ->
    text + expressions[i - 1].toUpperCase() + textPart

greet = (name, adjective) ->
  upperCaseExpr"""
               Hi #{name}. You look #{adjective}!
               """
```

3.0.7 Type conversions
Type conversions must always be done explicitly. Never rely on implicit type conversion

```
vnValue = Number vsValue
vbValue = Boolean vsValue
vsValue = String vnValue

// NOT: vnValue = +vsValue
// NOT: vbValue = ~!!vsValue
```

3.0.8 Use standard coffeescript expressions

```
if a in b
 doSomething()
// NOT:
if b.includes a
  doSomething()
```

3.0.9 `:` must be near key (not near value) in object definitions

```
hSomeObject =
  key1:      'value1'
  key2:      'value2'
// NOT:
hSomeObject =
  key1     : 'value1'
  key2     : 'value2'
```

3.0.10 function definition without arguments must not use brackets

```
computeAvg = ->

// NOT:  computeAvg = ()->
```

3.0.11 Use the vertical format always when possible

```
_                = require 'lodash'
joi              = require 'joi'
inflect          = require('i')()
status           = require 'statuses'
```

3.0.12 Do simple, be happy

3.1 Structures

Можно использовать элементарные структуры без логики и методов просто для передачи данных между методами или при сохранении в базу (отправке в браузер)
Это экземпляры классов Object и Array создаваемые с помощью операторов `{}` и `[]`. В именах ключей спец. символы не используем (т.к. ключи нужны только для получения значения)
В начале любого метода (функции) принимающей в качестве аргументов такие структуры необходимо выполнять глубокое копирование структуры во внутреннюю переменную - чтобы не произошло случайного изменения в передаваемых данных.

3.2 Variables

3.2.1 Use of global variables should be minimized.

3.2.2 Class variables (instance attributes) should never be declared public.
The concept of C++ information hiding and encapsulation has been inherited in this methodology. Use private variables and access functions instead. One exception to this rule is when need use a data structure, with no behavior (equivalent to a C struct). In this case it is appropriate to make instances of simple Object or Array by using `{}` and `[]` operators.

3.3 Functions

If function body is simple expression body may be in one line

```
square = (x) -> x * x
```

But if function body large or complex it must be start from new line

```
fill = (container, liquid = "coffee")->
  "Filling the #{container} with #{liquid}..."
```

3.4 Objects and arrays

If object or array definition is a little it may be in one line too

```
song = ["do", "re", "mi", "fa", "so"]
singers = {Jagger: "Rock", Elvis: "Roll"}
```

But if its definition is complex

```
bitlist = [
  1, 0, 1
  0, 0, 1
  1, 1, 0
]


kids =
  brother:
    name: "Max"
    age:  11
  sister:
    name: "Ida"
    age:  9
```

3.5 Loops

3.5.1 Iterator variables should be called i, j, k etc if its simple or not use.

```
for i in [1..10]
  doSomething()
```

3.5.2 When use `for-in` or `for-of` iterators with using item you should be use do() wrapper

```
for aFilename in list
```

```
do (aFilename) ->
  fs.readFile aFilename, (err, contents) ->
    compile aFilename, contents.toString()
```

3.6 Conditionals

3.6.1 Post `if` rule.
you simply use a regular **if** statement on a single line if expression is simple

```
mood = greatlyImproved if singing
```

3.6.2 `unless` rule.
you may use `unless` statement if condition is simple

```
unless bIsHappy
  clapsHands()
  chaChaCha()
else
  showIt()
```

3.6.3 simple condition may be in one line.

```
date = if friday then sue else jill
```

3.6.4 for all complex conditions must be used only `**if**` operator.

```
if a and b or c and d and (e or g)
 doSomething()
// NOT:
unless a and b or c and d and (e or g)
 doSomething()
```

3.7 Operators and aliases

Never use `==` or `===`! You must use **is** or **isnt** operators
Never use `!` You must use **not** operator
Never use `||` or `&&`! You must use **or** or **and** operators

Never use `false` or `true`! You must use **yes** or **no** boolean values
Never use `this`! You must use `**@**` symbol, fat arrow `**=>**` or renaming `**self = @**`

```
a ** b          Math.pow(a, b)

a // b        Math.floor(a / b)

a %% b         (a % b + b) % b
```

3.8 The Existential Operator

If some function may return null/undefined or if some object key may have null/undefined value in chaining the existential operator must be used

```
zip = lottery.drawWinner?().address?.zipcode
```

4. Layout and Comments

4.1 Layout

Code layout fundamental and major feature of CoffeeScript compiler.
First, the basics: CoffeeScript uses significant whitespace to delimit blocks of code. You don't need to use semicolons `;` to terminate expressions, ending the line will do just as well (although semicolons can still be used to fit multiple expressions onto a single line). Instead of using curly braces `{ }` to surround blocks of code in functions, if-statements, switch, and try/catch, use indentation.
You don't need to use parentheses to invoke a function if you're passing arguments. The implicit call wraps forward to the end of the line or block expression.
`console.log sys.inspect object` → `console.log(sys.inspect(object));`
(reed full documentation on http://coffeescript.org/)

4.2 White Space

4.2.1

- Commas should be followed by a white space.
- Semicolons in for statements should be followed by a space character.

```
doSomething(a, b, c, d)
// NOT: doSomething(a,b,c,d)

doSomething = (a, b, c, d)->
// NOT: doSomething = (a,b,c,d) ->
```

```
doSomething a, b, c, d
  .doSomething1()
  .doSomething2()
  .doSomething3()
// NOT:
doSomething a, b, c, d
.doSomething1()
.doSomething2()
.doSomething3()
// BUT:
doSomething
  name: a
  age: b
.doSomething1()
.doSomething2()
```

4.3 Comments

4.3.1 Tricky code should not be commented but rewritten!
In general, the use of comments should be minimized by making the code self-documenting by appropriate name choices and an explicit logical structure.

4.3.2 All comments should be written in English

4.3.3 Comments should be included relative to their position in the code.

```
  # Protect attachments and metadata from direct edit
  beforeUploadUpdate: (key, body = {}, opts) ->
    itemName = "#{inflect.underscore @Model.name}"
    body = _.omit body[itemName] ? {}, ['attachments', 'metadata']
    [key, body, opts]
```

4.3.4 Class and method header comments should follow the `Codo` conventions.

TODO: need examples