Design Guidelines for JavaScript Developers

Technical Documentation

© Copyright 2014, mdObject, Inc.

Contents

[Copyright 1](#_Toc405276136)

[1 Introduction 2](#_Toc405276137)

[1.1 Objective 2](#_Toc405276138)

[2 General rules 2](#_Toc405276139)

[2.1 Overview 2](#_Toc405276140)

[2.1.1 Don’t use Alert(), Confirm() and Prompt() in final code 2](#_Toc405276141)

[2.1.2 Use Namespaces 2](#_Toc405276142)

[2.1.3 Use Object Oriented design 2](#_Toc405276143)

[2.1.4 Add Prototype functions to intrinsic Classes only in utility.js 3](#_Toc405276144)

[2.1.5 Use Strict mode 3](#_Toc405276145)

[3 Naming conventions 3](#_Toc405276146)

[3.1 Overview 3](#_Toc405276147)

[3.2 Rules and Recommendations 3](#_Toc405276148)

[3.2.1 Use US-English for naming identifiers 3](#_Toc405276149)

[3.2.2 Use Pascal, Camel and Capital casing for naming identifiers 3](#_Toc405276150)

[3.2.3 Do not use a prefix, similar to the Hungarian notation 3](#_Toc405276151)

[3.2.4 Use suffix to identify type 4](#_Toc405276152)

[3.2.5 Add Callback to callback functions 4](#_Toc405276153)

[3.2.6 Suffix exception classes with Exception 4](#_Toc405276154)

[3.2.7 Do not use an underscore in identifiers 4](#_Toc405276155)

[3.2.8 Use a noun or a noun phrase to name a *class* 4](#_Toc405276156)

[3.2.9 Use singular names of enumeration types 4](#_Toc405276157)

[3.2.10 Variable declarations should always be made using **var.** 5](#_Toc405276158)

[3.2.11 Use Object.freeze() for enumerable. For example: 5](#_Toc405276159)

[3.2.12 Avoid usage of E and e prefix to an enumeration type 5](#_Toc405276160)

[3.2.13 Do not use letters that can be mistaken for digits, and vice versa 5](#_Toc405276161)

# Copyright

Copyright (c) 2013, 2014 mdObject, Inc. and other contributors. All Rights RESERVED

Released under the Microsoft Public License (MS-PL)

http://opensource.org/licenses/MS-PL

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.

See the License for the specific language governing permissions and limitations under the License.

# Introduction

## Objective

This document requires or recommends certain practices for developing programs in the JavaScript language. The objective of this coding standard is to have a positive effect on:

* Avoidance of errors/bugs, especially the hard-to-find ones.
* Maintainability, by promoting some proven design principles.
* Maintainability, by requiring or recommending a certain unity of style.
* Performance, by dissuading wasteful practices.

# General rules

## Overview

## Don’t use Alert(), Confirm() and Prompt() in final code

Don’t use the functions alert(), confirm() or prompt() in the final code. They block the JavaScript thread and therefore the updates in the web application. They can still be very useful and it’s OK to use them during the implementation phase.

## Use Namespaces

A namespace is a container which allows developers to bundle up all functionality under a unique, application-specific name. In JavaScript a namespace is just another object containing methods, properties and objects.

The idea behind creating a namespace in JavaScript is simple: one global object is created and all variables, methods and functions become properties of that object. Use of namespaces also minimizes the possibility of name conflicts in an application.

To create a global object called MyApplication:

// global namespace

var MyApplication = MyApplication || {};

Here in above code sample we have first checked whether MyApplication is already defined (either in same file or in another file). If yes, then use the existing MyApplication global object, otherwise create an empty object called MyApplication which will encapsulate method, functions, variables and objects.

## Use Object Oriented design

Refer to <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Introduction_to_Object-Oriented_JavaScript> for basic concept of Object-oriented programming in JavaScript.

## Add Prototype functions to intrinsic Classes only in utility.js

Prototype functions must be defined in utility.js Example: in utility.js

String.prototype.trim = function(){ ... }

## Use Strict mode

See: <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Strict_mode>

To invoke strict mode for an entire script, put the exact statement "use strict"; (or 'use strict';) before any other statements.

# Naming conventions

## Overview

## Rules and Recommendations

## Use US-English for naming identifiers

Names should be formed from the 26 upper and lower case letters (A .. Z, a .. z), the 10 digits (0 .. 9). Avoid use of \_ (underscore). Avoid use of international characters because they may not read well or be understood everywhere. Do not use $ (dollar sign) or \ (backslash) in names.

## Use Pascal, Camel and Capital casing for naming identifiers

* In Pascal casing the first letter of each word in an identifier is capitalized. For example, **BackColor**.
* In Camel casing only the first letter of the second, third, etc. word in a name is capitalized; for example, **backColor**.
* Two-letter abbreviations have both letters capitalized, e.g. **getIEVersion**().

*Exceptions:*

Identifier: Use Id for example: **getPatientId** ()

* Constant variables with capitals and underscores

The table below provides the casing for the most common types.

|  |  |  |
| --- | --- | --- |
| IDENTIFIER | CASE | EXAMPLE |
| Namespace | Pascal | var MyApplication = MyApplication || {}; |
| Class | Pascal | var Person = function() {}; |
| Enumeration | Pascal | var ColorEnum = {Red: 0, Green: 1, Blue: 2}; |
| Function | Camel | Utility.getIEVersion = function() |
| Local variables | Camel | var patientIndex = 0; |
| Constant variables | Capital | patientView.MAX\_ROWS = 9; |

## Do not use a prefix, similar to the Hungarian notation

var btOpen = new Button(); // bad

var openButton = new Button(); // good

## Use suffix to identify type

The controllers, directive, service, and filter should end with Controller, Directive, Service, and Filter word. For example:

var mainController = new Controller().

## Add Callback to callback functions

Functions that are used to pass a reference to a callback function must be suffixed with Callback. For example:

...

patient.callWebService(patient.queryTagCallback, "QueryTag", tagname);

...

patient.queryTagCallback = function(result){

...

}

## Suffix exception classes with Exception

For example:

function InsightException( errorCode, message )

{

this.errorCode = errorCode;

this.message = message;

}

## Do not use an underscore in identifiers

*Exceptions:*

To separate words in capital letters of constant variables.

## Use a noun or a noun phrase to name a *class*

Also, if the class involved is a derived class, it is a good practice to use a compound name. For example, if you have a class named *Button*, deriving from this class may result in a class named *SubmitButton*.

## Use singular names of enumeration types

For example, do not name an enumeration type Protocol**s** but name it Protocol instead.

## Variable declarations should always be made using **var.**

## Use Object.freeze() for enumerable. For example:

var ColorEnum = {"Red": 0, "Green": 1, "Blue": 2};

if (Object.freeze) {

ColorEnum = Object.freeze(ColorEnum);

}

## Avoid usage of E and e prefix to an enumeration type

## Do not use letters that can be mistaken for digits, and vice versa

To create obfuscated code, use very short, meaningless names formed from the letters O, o, l, I and the digits 0 and 1. Anyone reading code like

bool b001 = (lo == 10 ? I1 == 11 : 101 != 101);

will marvel at your creativity.