Cheatography

NASA Coding Standards for Javascript Cheat Sheet

by [deleted] via cheatography.com/2754/cs/9386/

Introduction

JPL have been developing software for most of unmanned missions in the field of deep space and other planets exploaration. High level of autmatization and long duration of missions led to superior demands to software quality. As a result of JPL amazing experience a set of code guidelines was developed and published recently. Since demands to web-driven software constantly increase and more critical tasks are entrusted to JavaScript, lets apply NASA coding guidelines to JavaScript / HTML applications for higher performance, reliability and the better world..

Credit: http://pixelscommander.com/en/javascript/nasa-coding-standarts-for-javascript-performance/

1, Function Length

No function should be longer than what can be printed on a single sheet of paper in a standard reference format with one line per statement and one line per declaration. Typically, this means no more than about 60 lines of code per function.
This fits perfectly for JavaScript. Decomposed code is better to understand, verify and maintain.

2. Restricting All Code

Restrict all code to very simple control flow constructs – do not use goto statements, setjmp or longjmp constructs, and direct or indirect recursion.
Rule from the C world makes wonder. We definitely will not use goto or setjmp in JS, but what is wrong with recursion? Why NASA guidelines prescribe to avoid simple technique we were studying as early as in school? The reason for that is static code analyzers NASA use to reduce the chance for error. Recursions make code less predictable for them. JavaScript tools do not have such a precept so what we can take out of this rule?
⬛Use constructs which are justified by complexity. If you want to write reliable code – drop to write tricky code and write predictable. Define coding standard and follow it;
⬛Use code analyzers to reduce chance for defect: JSHint/JSLint/Google Closure Tools;
⬛Keep codebase by monitoring metrics: Plato;
⬛Analyze types with Flow/Google Closure Tools.

3. Do not use dynamic memory allocation after ...

Do not use dynamic memory allocation after initialization.
At first glance JavaScript manage memory itself and garbage collection cleaning memory from time to time solving the rest of problems for us. But it is not absolutely correct.

3. Do not use dynamic memory allocation after ... (cont)

Memory leaks often, spoiled JavaScript developers do not have a culture of managing memory, garbage collector decrease performance when run and it is hard to tame. Actually we can get three recommendations from this rule. Two of them are nice to follow in any project and last one fits for performance and reliability critical software.. ⬛Manage your variables with respect. Regularize variables declaration by putting them in the top of scope in order to increase visibility of their usage;
⬛Watch for memory leaks, clean listeners and variables when not needed anymore. Classic article;
⬛Switch JavaScript to static memory mode and have predictable garbage collection behaviour (means no accident performance regression and no sawtooth pattern) by using objects pool.

4. All loops must have a fixed upper-bound.

As JPL explains this makes static analysis more effective and helps to avoid infinite loops. If limit is exceeded function returns error and this takes system out of failure state. For sure, this is quite valuable for software with 20 years uptime! Checks for limit exceeds are curried out by assertions. You may find more details on assertions in fifth rule. If you accept assertions practice use limits for loops as well, you will like it.

5.The assertion density

The assertion density of the code should average to a minimum of two assertions per function.
It is good to put few words here on what assertion is. The simplest parallel for them are unit tests which executes in run time.
1 if (!c_assert(altitude > MAX_POSSIBLE_ALTITUDE) == true) {
2 return ERROR;
3The rule literally says:
}
“Statistics for industrial coding efforts indicate that unit tests often find at least one defect per 10 to 100 lines of code written. The odds of intercepting defects increase with assertion density.”
Great, does this mean that we can treat rule as: “Write unit tests!“? Not exactly. Speciality of assertions is that they execute in run time and closest practice for JavaScript land is a combination of unit tests and run time checks for program state conformity with generating errors and errors handling.
⬛Than higher is tests density than less defects you get. Minimal amount of tests is 2 per function;
⬛Watch for anomalies in system state in run time. Generate and handle errors in case of failures.

6. Data objects must be declared ...

Data objects must be declared at the smallest possible level of scope.
This rule have simple intention behind – to keep data in private scope and avoid unauthorized access. Sounds generic, smart and easy to follow.

7. The return value of non-void functions

The return value of non-void functions must be checked by each calling function, and the validity of parameters must be checked inside each function.
Authors of guideline assure that this one is the most violated. And this is easy to believe because in it`s strictest form it means that even built-in functions should be verified. On my opinion it makes sense to verify results of third party libraries being returned to app code and function incoming parameters should be verified for existence and type accordance.

8. The use of the preprocessor

The use of the pre-processor must be limited to the inclusion of header files and simple macro definitions. The C pre-processor is a powerful obfuscation tool that can destroy code clarity and befuddle many text based checkers.
Using pre-processors should be limited in any language. They are not needed since we have standardized, clean and reliable syntax for putting commands into engine, it makes even less sense taking into account that JavaScript is constantly evolving. Reliable and fast JavaScript should be written in JavaScript. Nice research on determining the cost of JS transpilation

9.The use of pointers

The use of pointers should be restricted. Specifically, no more than one level of dereferencing is allowed. Function pointers are not permitted.

This is the rule JavaScript developer can not get anything from.

10. All code must be compiled

All code must be compiled, from the first day of development, with all compiler warnings enabled at the compiler's most pedantic setting. All code must compile with these setting without any warnings.
We all know it… Do not hoard warnings, do not postpone fixes, keep code clean and perfectionist inside you alive.