Parallax Gold Standard Checklist

1 Introduction

Parallax Semiconductor has a minimum set of requirements that an object written for the Parallax Propeller must meet in order to be certified as a “gold object” and accepted into the gold library. At a minimum, a Gold Standard object should implement well-written code, include proper documentation, and function as described in the object’s description. This checklist quantifies the functionality, readability, and consistency of a submitted object.

2 Coding Best Practices

A gold object should employ professionally commented code. Each public method in an object should begin with a document comment that describes the intended results of that method call.

2.1 Launching a Cog

Any object that launches a cog should have a public **`start`** method that takes care of any required initializations and a public **`stop`** method that shuts the running cog down gracefully. An object should have the ability to stop any running cog that it starts.

Programs should dynamically launch a new cog with the **`cognew`** command, and save the return value for later use when stopping the cog. Spin programs should use the **`cogstop`** command with the cog’s saved ID. PASM code should use **`cogid`** to stop itself. Once stopped, code should update its variables to reflect the termination of a cog.

If a COG requires an initialization function to reset internal state data between calls, the method should be named **`init`** and its usage properly documented.

2.2 Using Locks

An object that utilizes the Propeller’s lock features should make use of the **`locknew`** command to dynamically check out a new lock. If an object is to reach a terminus where the lock is no longer needed, it should use the **`lockret`** command to return the checked out lock.

2.3 Pin assignments

If an object uses any I/O pins, these pins should be provided as arguments to the **`start`** method. Any special considerations of the pin usage must be documented. This includes restrictions such as contiguous grouping of pins or starting at a specific number.

2.4 Clock Speed

Objects should not assume a particular clock speed. Instead, objects should dynamically check the clock speed register **`clkfreq`** and adjust accordingly. Any minimum or maximum clock speeds should be documented. If the object cannot be run in **`RCSLOW`** or **`RCFAST`** for some reason, please include that in the documentation.

2.5 Required Hardware

You should include information detailing the specific hardware or development environment needed to use your code. For example, if the code requires an SD card or other special hardware, include that information. It is also recommended you provide which Propeller based board the object and demos will run on, such as Propeller Demo board, Quickstart, PPDB, or Propeller BOE. Any helpful information on successfully duplicating your test environment is strongly suggested.

Item	Score	Max
Public **`start`** method, with any initializations included		5
`cognew` is used to start a new cog		5
Flags to indicate the running state of a cog		5
Public **`stop`** method that resets the object initializations		5
`locknew` and **`lockret`** are used safely		5
Hardware pins are **`start`** method arguments		5
Clock speed is dynamically checked and limitations noted		5
Required support and development board info included		5

Table 1.

3 Code Conventions

3.1 Method Naming

Method names should concisely reflect the function of the method. A method's name should always start with a capital letter and subsequent words should start with a capital letter (this style is often called UpperCamelCase).

It is helpful when naming methods to keep a probable object name (nickname) in mind for the object. For example, a simple object that lets a developer pause a cog's execution might be nicknamed `time` and call a method named `PauseMS`. The resulting `time.PauseMS(duration)` call gives additional context to the developer as to the functionality of the `time` object and `PauseMS` method, without requiring the developer to look at the object's source code to understand what the method call will do.

3.2 Variable Naming

Variable names should concisely reflect the use of the variable. A variable name should be descriptive yet concise. Variable names should always start with a lower-case character or underscore separator. If using more than one word in the variable name, capitalize the first letter of following words or use an underscore separator to make each word stand out. Capitalize the first letter of an acronym, except when the acronym starts the variable name. For example: `high_time_ms`, `numberOfRotations`, `ledArray`, or `bigLed`.

3.3 Commands and Reserved Keywords

All uses of reserved keywords and commands should remain in lower-case form, except **`PUB`**, **`PRI`**, **`VAR`**, **`CON`**, **`OBJ`**, and **`DAT`**.

3.4 White Space and Indentations

Indentation in Spin code must be a minimum of two spaces or one tab per block level. Within an object and it's demos, the indentation amount should be consistent. In Propeller Assembly, instructions should be indented enough to allow for label names, instruction conditions, the instruction itself, and instruction effects to fit neatly on a line. Each assembly instruction should be in line with the preceding instruction. Use white space and indentations to keep the code functional and easily readable. Try to keep large block comments limited to 92 characters wide as this makes paginating the object convenient.

3.5 Constant Declarations

Constant names should concisely reflect the use of the constant. Completely capitalize all defined constants.

3.6 Code Width

You should make a best effort to ensure code and comments are less than 80 characters wide for users on smaller screens and PDF pagination.

Item	Score	Max
Method names are consistent and in UpperCamelCase		5
Variable names are consistent and in lowerCamelCase		5
Block keywords are CAPITALIZED		5
Keywords are lowercase		5
Block body code is correctly indented		10
Constants names are consistent and CAPITALIZED		5
Code and comments are not more than 80 characters wide		5

Table 3.

4 Documentation

Gold standard Objects must have proper documentation. You may include the documentation embedded within the object, however you must also generate nicely formatted documentation applicable for pagination and/or web use.

The *Spin2Html* tool is provided to allow you to generate external document files from embedded documentation in SPIN files.

In source documentation:

- The beginning of the object should document overall functionality, inputs and outputs, resources allocated, object history, and other relevant information. This section should use the SpinDoc style. This section may contain:
    - Usage guidelines and short examples (hints about calling interface best practices)
    - Resources allocated
        - Number of COGs required
        - Amount of HUB RAM required
        - Stack size required
        - I/O pin usage
        - Number of locks
        - Any special clock frequency requirements
    - I/O pin considerations (including alignment and direction)
    - External device part numbers, sources, and datasheet links
    - Object version history and authorship
    - The performance of the code
- Dependencies on other SPIN objects
- Methods must include SpinDoc style descriptions of the functionality, the parameters (using the **@param** tag), the return value (using the **@return** tag), and should contain example method calls (using the **@example** tag). Any additional considerations (such as side effects, run time, etc.) should be documented as well.
- Use code type comments **'** or **{ }** to provide additional information on a line-by-line basis. Code type comments should be used when a line is important or unclear (such as complex operations, algorithms, or operational nuances).
- If necessary, include a schematic drawing and a description of the required connections needed to interface the Propeller with related hardware. Format the schematic as `schematic.jpg` or `schematic.gif`. The associated description should include an example of calling the **`start`** method that uses the pins in the schematic.

Item	Score	Max

Method arguments and result		5
Purpose of method		5
Usage hints and best practices		5
Number of COGs used is enumerated		5
Amount of HUB RAM required *[longs]*		5
Stack size required *[longs]*		5
I/O pin usage *[if any]*		5
Number of locks *[if any]*		5
Clock frequency requirements *[if any]*		5
I/O pin alignment *[eg: video, bulk data xfer]*		5
Part numbers, datasheets for external devices *[if any]*		5
Dependencies on other SPIN objects *[if any]*		5

Table 4.

5 Demo Code

An object should include at least one demonstration program. At a minimum, the demo program should illustrate a very simple example of how to properly use the object. (i.e.: you hook it up like this, run this program, and these are the results you will see). Additional demo programs can be included with the object to demonstrate advanced features of the object. For all demo programs, connection references between the Propeller and the external peripheral should remain consistent. Demo programs should use a serial terminal (such as the Parallax Serial Terminal) as the standard method for user input and program output.

All Demo programs must follow the same code practices outlined in other sections of this document.

Item	Score	Max
Demo program available		100
Demo program uses a PC serial connection		10
Demo program is clearly and consistently commented		10

Table 5.

6 Licensing

Any object in the Gold Standard Library must be released under the *MIT* license or *Attribution 3.0 Unported* license. If MIT licensed, please insert the complete text of the MIT license at the bottom of the object in a **`DAT`** section. Code must have one, and only one, type of license.

Item	Y/N
Code is MIT licensed	
Code is Attribution 3.0 Unported (CC BY 3.0) licensed	
Each file (object, demo, ...) includes a licence copy	

Table 6.

7 Submission

All Gold Standard submissions should be provided as a ZIP file of a directory containing the object and supporting files.

If you have an object named `foobotron` then the ZIP file should contain a directory named `foobotron` so that the object can be stored hierarchically in the library.

The archive should contain your object, schematic, any supporting files necessary for the object, examples, and any additional documentation you may provide. The schematic does not need to be provided in multiple formats, the example below simply includes both for completeness.

Supplemental documentation should be provided in PDF, TXT, HTML, or Open Document Text (ODT).

Additional files may also be included if you feel they are valuable, such as design files from schematic capture packages, Gerber files, or other collateral.

Here is an example layout:

```
foobotron/
foobotron/foobotron.spin
foobotron/schematic.spin
foobotron/schematic.pdf
foobotron/example1.spin
foobotron/example2.spin
foobotron/documentation.odt
```

Item	Y/N
Object provided in named directory	
Object source code	
Schematic	
Example	
Documentation	

Table 7.