CODE REVIEW CHECKLIST – AUGUST 7, 2012

PREPARATION

- ☐ Understand your requirements and review your code with them in mind.
- ☐ Run the code before reviewing to ensure it executes and shuts down correctly.
- ☐ Run the VI Analyzer and review results before having others review the code.
- ☐ Consider using the Desktop Execution Trace Toolkit to check for memory leaks and other runtime errors.
- ☐ Ensure that VIs have names that clearly and accurately represent their function. Consider using full words with spaces for VI names. Consider using prefixes to delineate the types of VIs (e.g., use FGV for functional global variables, use TYPE for typedefs, etc.)
- ☐ Projects, libraries, and classes should be well organized. The top-level VI(s) should be obvious in projects.

FRONT PANEL

- ☐ Avoid an aesthetically unpleasing front panel. Ensure the operation of the panel is clear and that results are presented in an understandable way.
- ☐ Consider including text on the front panel which gives instructions for using the VI.
- ☐ Consider arranging the front panel window so that all controls and indicators are visible without scrolling.
- ☐ Include a stop button which gracefully shuts down all parts of the VI. The stop button should be placed in the bottom right corner of the VI using the default appearance
- ☐ Be consistent in using one style for controls. Consider using System or Silver control styles.
- ☐ Avoid making the panel so narrow that menus and the toolbar are truncated.
- ☐ Consider grouping related data into clusters or classes
- ☐ Document default values for controls in parenthesis in the label.
- ☐ Consider specifying engineering units in the labels for controls and indicators.
- ☐ Consider using transparent labels for the background of controls and indicators.
- ☐ Use meaningful labels

- ☐ Place labels consistently
- ☐ Write tip strips or descriptions for UI front panels and controls
- ☐ Set up appropriate keyboard shortcuts for controls, and ensure the tabbing order makes sense.
- ☐ Avoid excessive use of color. Ensure that a colorblind person will be able to operate the front panel.

BLOCK DIAGRAM

- ☐ Avoid aesthetically unpleasing block diagrams. Avoid unnecessary white space on the diagram. Consider the judicious use of the block diagram cleanup tool.
- ☐ Arrange diagrams such that scrolling is minimized.
- ☐ Do not leave front panel terminals unwired in the block diagram.
- ☐ Label Call Library Function nodes with the function they are calling.
- ☐ Avoid excessive diagram colors and wire patterns.
- ☐ Propagate the error cluster.
- ☐ Do not place block diagram objects on top of wires.
- ☐ Use path constants instead of string constants to specify the location of files or directories.
- ☐ Avoid the use of sequence structures
- ☐ Save the VI with the most important frame of multi-frame structures (Case, Event) showing.
- ☐ Label shift registers
- ☐ Ensure data flows through wires from left to right.
- ☐ Consider arranging parallel loops from top to bottom.
- ☐ Do not use local and global variables for wiring convenience.
- ☐ Include delays in continuously running loops that are designated for user input.
- ☐ Avoid using an infinite timeout for event structures, reading from queues, and acquiring data. Ensure that loops can be stopped gracefully if the user wants to shut down the application.
- ☐ Avoid using arbitrary timing delays waiting for an instrument or user to respond, or a trigger to occur.
- ☐ Avoid unnecessary type coercions.

- ☐ Consider labeling case structures, especially true/false case structures.
- ☐ Consider using typedefs for enums and clusters, to make them easier to update them later.
- ☐ Ensure control and indicator terminals are on the top level diagram. Control terminals should be placed at the far left and indicator terminals on the far right of the block diagram.

PROGRAMMING ERRORS

- ☐ Avoid using Use Default if Unwired on tunnels of case and event structures. An exception is when the tunnel contains the Boolean to stop the loop.
- ☐ Avoid depending on automatic error handling.
- ☐ Consider name collisions when using named queues and named notifiers.
- ☐ Close all references.
- ☐ Do not have wires behind objects.
- ☐ Consider validating values of input parameters. For example, if a number should always be less than a certain value, ensure that it is.
- ☐ Use shift registers (not tunnels) to pass references and errors through loops.
- ☐ Ensure For Loops behave correctly if they execute zero times.
- ☐ Avoid putting indicators in case structures.
- ☐ Avoid including code that reallocates memory inside of loops. (Build Array, Concatenate Strings)
- ☐ Consider putting code that opens/closes resources (ie DAQ & File I/O) outside of loops.
- ☐ Encapsulate critical sections in non-reentrant subVIs.
- ☐ Remove unused code.
- ☐ Employ an error handling strategy. Determine the actions for errors: catch, log, terminate, notify user, etc.
- ☐ Consider checking at runtime whether timed loops are able to run at their specified rate.
- ☐ Ensure that relative file paths are appropriately derived for Run-Time and the Development System.

DOCUMENTATION

- ☐ Document the version of LabVIEW used to create the VIs and any needed add-ons. (LabVIEW toolkits, drivers, and/or 3rd party products).

- ☐ Include documentation for block diagrams that use special features/functions/algorithms.
- ☐ Describe how the VI works in the VI Description.
- ☐ Document how to use all controls and indicators on UI VIs. The range of valid input values should be described if not obvious.

SUBVIS

- ☐ Use subVIs to encapsulate and modularize the different parts of your application.
- ☐ Use a consistent connector pane across related VIs.
- ☐ Do not duplicate code. Use subVIs to contain code that you use in more than one part of your application.
- ☐ Ensure the front panel and connector pane have a similar layout
- ☐ Consider creating each subVI such that it has a single clear purpose.
- ☐ Use consistent connector panes on all subVIs that are related, usually 4x2x2x4 or 5x3x3x5.
- ☐ Wire the connector pane appropriately.
    - ☐ Controls on left, indicators on the right.
    - ☐ Path or reference on top left corner.
    - ☐ Error in/out on bottom left/right.
    - ☐ Use required inputs for parameters that must be wired for the VI to do anything meaningful.
- ☐ Create an appropriate icon for subVIs. Don't use the default icon, a blank icon, or an icon that looks like a vi.lib icon. Libraries should have a consistent banner and/or glyph. (more details on icon style can be found at http://www.ni.com/white-paper/6453/en)

REFERENCES:

LabVIEW Style Checklist, LabVIEW 2012 Help

LabVIEW Instrument Driver Development Guidelines,
http://www.ni.com/devzone/idnet/library/instrument_driver_guidelines.htm

The LabVIEW Style Book, by Peter Blume, http://www.bloomy.com/lvstyle/

Compatible with LabVIEW Checklists, https://decibel.ni.com/content/docs/DOC-8981