APPROVAL SHEET

TITLE : *SALT LabVIEW Coding Standard*

DOCUMENT NUMBER : 1000BS0044 ISSUE: 1

SYNOPSIS : This document describes the requirements for LabVIEW software developed for SALT.

KEYWORDS : Software, Operating System, LabVIEW, MMI, GUI

PREPARED BY : Gerhard Swart

APPROVED : Kobus Meiring
SALT PROJECT MANAGER

DATE : April 2002
This issue is only valid when the above signatures are present.
Printed: 16/09/03 11:58
### ACRONYMS AND ABBREVIATIONS

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>BITE</td>
<td>Built-in Test Equipment</td>
</tr>
<tr>
<td>I/O</td>
<td>Input/Output (Device)</td>
</tr>
<tr>
<td>ICD</td>
<td>Interface Control Dossier</td>
</tr>
<tr>
<td>MMI</td>
<td>Man-Machine Interface</td>
</tr>
<tr>
<td>PC</td>
<td>Personal Computer</td>
</tr>
<tr>
<td>RT</td>
<td>Real-time</td>
</tr>
<tr>
<td>SALT</td>
<td>Southern African Large Telescope</td>
</tr>
<tr>
<td>SDD</td>
<td>Software Design Document</td>
</tr>
<tr>
<td>SDP</td>
<td>Software Development Plan</td>
</tr>
<tr>
<td>SRS</td>
<td>Software Requirement Specification</td>
</tr>
<tr>
<td>SW</td>
<td>Software</td>
</tr>
<tr>
<td>TBC</td>
<td>To Be Confirmed</td>
</tr>
<tr>
<td>TBD</td>
<td>To Be Determined</td>
</tr>
<tr>
<td>TCS</td>
<td>Telescope Control System</td>
</tr>
<tr>
<td>VI</td>
<td>Virtual Instrument (LabVIEW function)</td>
</tr>
</tbody>
</table>
DEFINITIONS

Supplier      The organisation developing a SW Item.

Developer     The technical person(s) working for the supplier organisation responsible for the development of the SW item. This term is used interchangeably with “Supplier”.

Client        SALT (Pty) Ltd, as represented by the appropriate subsystem manager, contracting the development of the SW Item to the supplier.
# TABLE OF CONTENTS

1 Scope .................................................................................................................. 6
2 Referenced Documents .......................................................................................... 7
3 Important LabVIEW Coding Principles ............................................................... 7
   3.1 Modularize and test your Vis ......................................................................... 7
   3.2 Document as you go along ............................................................................ 7
   3.3 Don’t forget dataflow .................................................................................. 8
4 Coding Standards: What Users See ....................................................................... 8
   4.1 VI Hierarchy .................................................................................................. 8
   4.2 User Interfaces: Front Panels with Style ....................................................... 9
       4.2.1 Why worry about the graphical interface appearance? .......................... 9
       4.2.2 Consistency ........................................................................................... 9
       4.2.3 Text ....................................................................................................... 10
       4.2.4 Color .................................................................................................... 10
       4.2.5 Graphics and Custom Controls .............................................................. 11
       4.2.6 Front Panel Layout .............................................................................. 11
       4.2.7 Sizing and Positioning Front Panels ...................................................... 12
   4.3 Controls and Indicators .................................................................................. 13
       4.3.1 Descriptions ............................................................................................ 13
       4.3.2 Labels ...................................................................................................... 13
       4.3.3 Default Values, Ranges, and Coercion ..................................................... 13
   4.4 Attribute Nodes ............................................................................................... 14
   4.5 Key Navigation .................................................................................................. 14
   4.6 VI Setup .......................................................................................................... 14
   4.7 Connector Panes ............................................................................................... 15
5 Coding Standards: What Programmers See .......................................................... 15
   5.1 Wiring Etiquette .............................................................................................. 15
   5.2 Labeling .......................................................................................................... 15
   5.3 Execution Sequence ......................................................................................... 16
       5.3.1 Left-to-Right Layouts ............................................................................. 16
       5.3.2 Data Dependency .................................................................................... 16
       5.3.3 Sequence Structures .............................................................................. 16
       5.3.4 Check for Errors ..................................................................................... 17
   5.4 Optimization .................................................................................................... 18
   5.5 Code Interface Nodes ...................................................................................... 18
       5.5.1 CIN Description Contents ...................................................................... 18
       5.5.2 CIN Source Code .................................................................................. 18
6 Style Checklist ........................................................................................................ 18
1 Scope

This document specifies the requirements for all LabVIEW software that forms part of the Southern African Large Telescope (SALT). It indicates the required coding and documentation practices that are applicable.

This document is based on the LabVIEW Coding Standard developed for Rayodyne LP by Jeffrey Travis. Most of the guideline of that document have been left unchanged, but specific requirements have been added, clearly marked in text boxes.

The purpose of this standard is to ensure that the software used on SALT is maintainable, expandable and that operator interfaces have a common style and layout. This document supplements the SALT Software Standard, which guides the overall software quality and development process.

If any requirements of this document are considered excessive, inappropriate or difficult to meet, developers are invited to seek guidance from the author or to request a concession of such requirements.
2 Referenced Documents

The following documents are referenced in this standard.

SALT-1000BS0010  SALT Software Standard
Internet Applications in LabVIEW, Jeffrey Travis, 2000 Prentice Hall
LabVIEW Coding Standard developed for Rayodyne LP, Issue 1.1
Jeffrey Travis
LabVIEW Development Guideline, NI, July 2000

3 Important LabVIEW Coding Principles

3.1 Modularise and test your VI's

Although it's theoretically possible to be too modular in designing your program, this rarely happens. Make all but the most simple and trivial functions and procedures subVI’s. This gives you a chance to test each of your individual pieces of code before working with the big piece. It also lets you easily re-use code, keep yourself organized, and make the block diagram size manageable. Don’t forget to test each subVI as a top-level VI— and be thorough: test all the extremes of input combinations. If you know all your subVI’s are working, it should be very easy to debug any problems with your top-level VI.

Here is a quick, although not perfect, modularity metric you should perform on your VI’s:

At a monitor resolution of 1024x768, with the block diagram window maximized, for each separate VI, all your code and comments should fit within the window (e.g., no need for scrolling). If this is not the case, you should modularise some of your code.

The previous rule obviously doesn't say too much about the correct functionality of your code, but it will give you a guideline for when your block diagrams are getting too big.

One useful tip: often LabVIEW programmers won't test certain VI's because they require DAQ hardware or other externally-generated inputs. If possible, don't wait until you have the hardware. Write a simple “dummy data” VI to pass data to these VI's in the meantime so you can at least test part of their functionality.

3.2 Document as you go along

You must document your work. Many programmers shun documentation, thinking it will be done later or not needed... until a user (possibly yourself) 2 years later is trying to figure out how this VI works. Take advantage of LabVIEW's built-in documentation capabilities:

a. VI Info. At the very least, you must write a short description for each VI you create. This is extremely valuable when someone is looking at the subVI icons in the block diagram and needs an idea of what they do. In the VI Info, you can also use the special patterns that source control systems like PVCVS or MS Visual Source Safe use, to automatically have your VI update information like revision number, author, last modification date, etc.

   1. Each custom VI shall have adequate comments in the Description Panel and as floating text in the diagram, to allow full understanding of the software functional operation and data flow.

b. Descriptions. Ideally, write a help statement for each front panel control and indicator using the pop-up Description... command. These invaluable comments will appear on the Help window if a user points to the control or indicator in question.

   2. Each front panel object visible to the operator shall have a Description and Tip defined, to provide the operator with basic operational information. Hidden front panels shall use Description and Tip to describe custom data types and units of measure.
c. VI History. This option, available from the Windows menu, is a more sophisticated tool for larger projects. It allows you to enter comments about the changes you’ve made to a VI along the way. The History Window can be quite helpful when more than one person works on a project, since it keeps track of the user, time, and date. Note: If you are using SourceSafe or a similar program, enter the comments as you check in and check out VI’s in SourceSafe instead of VI History.

3. An approved method of maintaining configuration control at VI level shall be used.

d. Front Panel Text. For important indications, just write some text (perhaps with a bold or large font) on the front panel itself. Users can’t miss that one!

e. Comments the block diagram. LabVIEW has one of the most flexible documentation syntax structures: you can place comments anywhere you want on the block diagram. You should have clear explanations throughout your code about what it is doing; for example, write a description inside each frame of a Case structure that explains what is happening in that particular case.

f. Meaningful Icons. You don’t need to be an artist, but you must create icons for each subVI that provide information about what they do. Consider colour-coding icons by functionality (e.g., green for the user interface, yellow for file I/O, etc.) and using text in the icon itself to describe it. The icon descriptions are critical for examining the VI Hierarchy and making sense of it.

4. Each custom VI shall have a unique Icon identifying it. The following colour-code shall be used:
- Generic common VI’s used on multiple subsystems – Green
- Input VI’s (up to and including scaling in engineering units) – Red
- Processing VI’s – Cyan
- Output VI’s (up to and including scaling in engineering units) – Blue
- File handling VI’s – Yellow
- User interface VI’s – Orange
- Other custom VI’s - Magenta
- Top Level VI – White

3.3 Don’t forget dataflow
Always take advantage of the way dataflow programming works. Some tips to remember are:
- Dataflow means that data is carried across wires. When data reaches a terminal of a subVI or a function, that subVI or function will only start executing after ALL of its input terminals have received data.
- Two or more objects or groups of objects on the block diagram that are not connected have no specific execution sequence. Many people new to LabVIEW have a feeling that execution should take place left-to-right, or top-to-bottom. That is not true! There is no way to predict in what order two or more pieces of block diagram will occur unless it’s specified by dataflow.
- If you need to force dataflow, when possible, use single-frame Sequence structures, instead of using multiple-frame Sequence structures that create awkward Sequence locals and hide the code.
- You’ve noticed that many LabVIEW functions have common “threads”: refnums, taskID, error clusters. These threads are designed to string several common VI’s together in a natural dataflow sequence. Normally, you should always have error clusters in your VI’s that can be used as a dataflow thread (in addition to providing error handling functionality). Using threads will reduce the need for Sequence structures.

4 Coding Standards: What Users See
4.1 VI Hierarchy
- Organize your VI’s in the file system to reflect the hierarchical nature of your software. Generally, you should keep all your VI’s as separate .vi files within directories and not use .llb files (llb’s were
implemented to overcome the Windows 3.1 naming limitations, but are no longer necessary).

5. All custom VI’s shall be kept in a directory separate from VI’s which form part of LabVIEW.

- Make the top-level VI’s directly accessible. Place subVI’s in subdirectories and group them to reflect any modular components you have designed, such as instrument drivers, configuration utilities, and file i/O drivers; e.g., "Drivers", "FileIO", "DAQ", "Main", etc. Put your top-level "main" VI above all the other directories so it is clear to the user what VI should be run. Unless the project specifies otherwise, call your top-level VI main.vi. You can always make a shortcut to it with another name for the user's convenience.

6. Keep each type of VI (as defined in Requirement 4) in a separate subdirectory.

7. The top-level VI shall be called “XXXmain.vi”, where XXX is the approved abbreviation for the computer on which the software resides. E.g. SDCmain.vi

8. Stand-alone subsystem local interfaces shall be called “XXXMMI.vi”, where XXX is the approved abbreviation for the computer on which the main control software normally resides e.g. SDCMMI.vi

- Create a directory for all the VI’s for one application and give it a meaningful name. Save the main VI’s in this directory and the subVI’s in a subdirectory. If the subVI’s have subVI’s, continue the directory hierarchy downward.
- When naming VI’s, VI libraries, and directories, Do Not use characters that are not accepted by all file systems, such as slash (/), backslash (\), colon (:), tilde (~), and so on.
- Select Edit»Preferences… to make sure the VI Search Path contains <topvi>\* and <foundvi>\*. The * causes all subdirectories to be searched.
- Avoid creating files with the same name anywhere within your hierarchy, because only one VI of a given name can be in memory at a time. If you have a VI with a specific name in memory and you attempt to load another VI that references a subVI of the same name, the VI will link to the VI in memory.

9. No two different VI’s shall have the same name or Icon.

4.2 User Interfaces: Front Panels with Style

4.2.1 Why worry about the graphical interface appearance?
The front panel of a VI is to a G program what the cockpit is to an airplane. Just as cockpit instruments give the pilot control over even the most technologically complex aircraft, G front-panel instruments give you, the programmer, control over program input and output. No conventional programming environment has anything comparable to a LabVIEW built-in user interface. A user's first contact with your work, and with LabVIEW, is the front panel, so it had better be high quality.

Unfortunately, most engineers and computer scientists are given little, if any, training on user interface design. Learning some of the rudiments of good design principles can greatly enhance the quality, usability, aesthetics, and even safety level of your software. The following principles give you some guidelines on user interface design in G.

4.2.2 Consistency
The most important rules of user interface design are consistency, consistency, and consistency. The user cannot adapt to your style if your application contains significant changes with every front panel. Standardize on a few colours, fonts, and layout practices that are attractive and functional.

Also, professional societies have written standards for human-machine interface design if you are interested in reading further on UI design.
10. Annexure A provides an example of the required front panel colours, font size and control types required for SALT. Each MMI shall be subject to approval by SALT.

4.2.3 Text
Do not be tempted to use all the fonts and styles available. Stick to three standard fonts, application, system, and dialog, unless you have a specific reason to use a different font. For example, monospace fonts, fonts that are not proportionally spaced (e.g., Courier), are useful for string controls and indicators where the number of characters is critical. To set the default font, choose it from the Text Settings drop-down menu in the toolbar without any text or objects selected. You can select all the labels you need to change and set the font in all of them at once using the Text Settings drop-down menu in the toolbar.

11. All text (Front Panel & Diagram) shall be in LabVIEW’s Application Font, set to its default setting, which is 13pt, plain, black, left justified. Font type (italics, bold, etc) may be changed, and text may be capitalised or size increased if necessary.

The actual font used for the three standard fonts varies depending on the platform, your preferences, and video driver settings, when working under Windows. Text might appear larger or smaller. To compensate for this, allow extra space for larger fonts and keep the Size to Text option on the pop-up menu. Use carriage returns to make multi-line text instead of resizing the text frame. You can prevent controls and indicators from overlapping because of font changes on multiple platforms by allowing extra space between controls. Fonts are the least portable aspect of the front panel, so always test them on all your target platforms.

11a. The actual Application font in LabVIEW depends on the user’s particular computer settings, and is the same font assigned to the “Icon” item. Therefore, in Windows Display Properties go to Appearance and under Item select Icon; this will show the font and font size. For Windows 2000 this shall be Tahoma set to size 8 point. Also, in Windows Properties (Settings, Advanced) ensure that the Font Size is set to “Small Fonts”.

Be sure and set control and indicator background labels to "transparent". You can set this as default in LabVIEW ‘Options’.

4.2.4 Colour
In addition, unless you have a good eye or background in graphic arts, use a utility program for helping you choose a set of three colours that go well together. You can get such a utility program from http://www.bettervi.com (there is a shareware version as well as a commercial version).

The following are some simple guidelines for using colour:

- Never use colour as the sole indicator of device state. People with some degree of blindness-blindness (5% of men) might not detect the change. Also, multiplot graphs and charts can lose meaning when printed in black and white. Use line styles in addition to colour, and whenever possible use text labels right on the graphic data so users don’t have to look for the legend.
- Use light grey, white, or pastel colours for backgrounds
- Select bright, highlighting colours only when the item is important, such as an error notification.
- Consider how front panels that are routinely printed will look on a black-and-white printer.
- Be consistent.
4.2.5 Graphics and Custom Controls
You can enhance the functionality of your front panel with imported graphics. You can import bitmaps, Macintosh PICTs, Windows Enhanced Metafiles, and text objects for use as backgrounds or in pict rings and custom controls, as shown in the following figure.

Figure 3. Example of Imported Graphics Used in a Pict Ring

Use a pict ring when a function or mode is conveniently described by a picture.

A custom Boolean control that is transparent in one state appears when the state changes. A completely transparent Boolean is useful for detecting mouse clicks in specified regions of the screen. Check how your imported pictures look when your VI is loaded on another platform. For example, a Macintosh PICT file that has an irregular shape might convert to a rectangular bitmap with a white background under Windows or UNIX.

One disadvantage of imported graphics is that they slow down screen updates. The following suggestions might improve performance:
• Make sure indicators and controls are not placed on top of a graphic object. That way, the object does not have to be redrawn each time the indicator is updated.
• If you must use a large background picture with controls on top of it, try breaking it into several smaller objects and import them separately. Large graphics usually take longer to draw than small ones. For instance, you could import several pictures of valves and pipes individually instead of importing one large picture.

4.2.6 Front Panel Layout
Consider the arrangement of controls on front panels:
• Keep front panels simple to avoid confusing the user. For top-level VI’s that users see, place the most important controls in the most prominent positions.

11. All static text (not varying with software execution) should be black on a grey background.
12. Number/text entry or drop-down lists should be black text on a white background.
13. Text super-imposed on graphics may be coloured.
14. The following colours should only be used as indicated below:
   - Red: To indicate failures or dangerous events/states
   - Orange: To indicate warnings
   - Green: To indicate “good” or ON states
   - “Off” or greyed-out for inactive or OFF states that do not required any operator action or alertness.
   - Blue: Information that is neither good nor bad (e.g. in bar graphs).
   - Any colour flashing is used to attract operator attention, but use of this should be restricted to flashing Orange or Red.
15. Colour shall be used widely to enhance operator awareness of equipment states.
16. The Front Panel background colour shall be grey.
• Use the Align Objects and the Distribute Objects drop-down menus to create a uniform layout.
• Use Edit»Panel Order… to arrange controls in a logical sequence. Refer to the Key Navigation section later in this chapter for more information.
• Do not overlap controls with other controls or with their own label, digital display, or other parts unless you are trying to achieve a special effect. Overlapped controls are much slower to draw and might flash.
• Use simple elements such as rounded rectangles to visually group objects with related functions. Use clusters to group related data. However, do not use clusters for aesthetic purposes only. It makes connections to your VI more difficult to understand.
• For subVI front panels the user does not see, you can place the objects so they correspond to the connector pattern. Generally, inputs should be on the left and outputs on the right.
• Prevent user changing parameters that can cause erroneous operation.
• Grey out controls that are not available.
• Names of items must indicate clearly the interpretation of the TRUE/FALSE state.
• Information displayed must clearly distinguish between required state and actual equipment state.

17. The user interface shall be laid out in the following fashion (see Appendix A):
- The top part of the screen, stretching the full screen width, shall indicate the following:
  o the name of the subsystem/item being controlled by the software
  o the time and date (top right-hand corner)
  o the mode of the subsystem
  o a summary of the health status of the subsystem
  o one or two most important parameters (e.g. position/angle)
- The remaining part of the screen is divided into two sections.
  o The left-hand side shall provide information to the user. It shall have a pictorial representation of the subsystem, animated if possible, to represent the normal movements/changes that can occur. Boolean, numeric, bar and graph indicators shall be neatly positioned to indicate the various high-level states of the subsystem components (e.g. angles, temperatures, ON/OFF states, failures)
  o The right-hand side shall be used for operator interaction and shall contain buttons, knobs, drop-down lists, and numeric/text entry fields as appropriate. Direct feedback of the operator actions (e.g. confirmation of actions) should also be provided close to the appropriate control.
  o Where the information is too much to fit onto the screen at one time, Tabs shall be used to select various pages. The top section of the screen should always be visible and the left-hand part always visible if that particular part of the subsystem can be influenced by the operator action. The right-hand side can have the most variation. Information should always be grouped to minimise the requirement for paging and to group data logically together.
  o Pop-up panels may also be used.

4.2.7 Sizing and Positioning Front Panels
• Front panels should fit on a monitor that is the standard size for most intended users (remember that screen resolution, not the monitor size, determines your space real estate). Don't forget to view your VI on the target resolution.
• Front panels should open in the upper-left corner of the screen for the convenience of users with small screens.
• Place front panels that open automatically in the centre of the screen by selecting the Auto-Centre option in the Windows Options version of the VI Setup dialog box to optimise this for monitors of various sizes.

18. Subsystem MMI’s will be viewed by SALT operators and astronomers on a 21" monitor, and so to allow viewing from a distance, important information shall have a maximum font size of 28 point.
They shall also be sized to 800x600 pixels, to allow the simultaneous display of several (4) MMI’s on the TCS MMI.
4.3.3 appear bold in the on-line Help windows.

Trick: In control Descriptions and in the VI Info box, you can enclose text with `<b>` and `</b>` tags to make it appear bold in the on-line Help windows.

4.3 Controls and Indicators
The following sections guide you on when and how to use various controls and indicators effectively.

4.3.1 Descriptions
Every control and indicator should have a visible label (or caption) and a description (see the section on Documentation), unless the purpose of the control is extremely obvious.

4.3.2 Labels
- The Help window displays labels as part of the connector. Label the most important controls and indicators on a front panel in bold.
- Include the units of a control or indicator's value, where applicable. In particular, always include units on charts and graphs so the user knows what is being displayed.
- The Required/Recommended/Optional setting affects the appearance of the inputs and outputs in the Help window. See the LabVIEW documentation for more information about the Required/Recommended/Optional setting.
- The name of a control or indicator should describe its function. For example, for a ring or labelled slide with options for volts, ohms, or amperes, a name like “Select units for display” is better than “V/O/A” and is certainly an improvement over the generic “Mode.” Of course, long names use valuable space on the block diagram, especially if you use any local variables or Bundle/Unbundle by Name functions. You might prefer to use a caption for a control to give it a long name, but give the control a short name.
- For Booleans, the name should give an indication of which state corresponds to which function, while still indicating the default state.
- Place free labels next to the Boolean to help clarify the meaning of each position on a switch (e.g., on/off). Use `Boolean` text!

Trick: In control Descriptions and in the VI Info box, you can enclose text with `<b>` and `</b>` tags to make it appear bold in the on-line Help windows.

4.3.3 Default Values, Ranges, and Coercion
- Expect the user to supply invalid values to every control. You can check for invalid values in your block diagram or set the control Data Range item to coerce values into the desired range: minimum, maximum, and increment.
- There are some speed and memory usage drawbacks to limiting ranges. The Data Range function adds some execution overhead, as does the Find Range VI and similar VI's.
- Controls should have reasonable default values. A VI should not fail when run with default values.
- Do not set default values of indicators like graphs, arrays, and strings without a good reason because that wastes disk space when saving the VI.
- Use default values intelligently. In the case of high-level file VI's such as the Write Characters to File VI, the default is an empty path that forces the VI to display a File Selection dialog box. This can save the use of a Boolean switch in many cases.

19. All user-entered parameters shall have limit and validity checks performed.
20. Strong preference is given to drop-down lists over text type-in, to reduce the possibility of user error and to speed up data entry.
21. Default values shall be selected to place equipment into a safe state.
22. Settings that may require change later, shall be contained in an “XXX.ini” file which is read during initialisation. Parameters that should be configured by maintenance staff shall be adjustable from within the application software (e.g. calibration data, control loop parameters, park positions). Any values written by software to this file, shall have adequate error checking and be returned with a meaningful default value.
4.4 Property Nodes
Use property nodes to give the user more feedback on the front panel. There are many things you can do to make your VI easier to use, including the following suggestions:

- Set the text focus to the main, most commonly used control.
- Dim or hide controls that are not currently relevant or valid.
- Guide the user through steps by highlighting controls.
- Change screen colours to bring attention to error conditions.

4.5 Key Navigation
Hint: You can set key navigation options from the Key Navigation… item of the pop-up menu of any control.

- Consider including keyboard shortcuts to your programs, especially if the software will be used in environments where the mouse is not as convenient (e.g., manufacturing floor).
- Consider the tab order of controls. If you select Edit»Panel Order…, you can see the order of your front panel controls. This order controls the tab order for your front panel. In general, set the order to read left to right and top to bottom.
- Set the <Enter> key to be the keyboard shortcut to the front panel default control. However, if you have a multiline string control on the front panel, you should not want to use the <Enter> key as a shortcut. For string controls where you are limiting the string to one line (by explicitly setting this option on the pop-up menu), the <Enter> key will automatically update the control without the need to assign it.
- If your front panel has a Cancel button, assign a shortcut to the <Esc> key.
- Also, you might consider using the Key Focus attribute to set the focus programmatically to a specific control when the front panel opens.

4.6 VI Setup
- To access the VI Setup dialog box, pop up on the VI icon and choose VI Setup. Think about the window behaviour and style of every VI that will be visible to the user in your project.
- In the Window Options version of the VI Setup dialog box, select Dialog Box for front panels that should wait for input from the user before the program can continue.
- Turn off the Allow User to Close Window option to keep users from accidentally closing an important front panel while it is running. Disable the scroll bars, the menu bar, and the toolbar unless the user needs them. You can use the keyboard shortcuts for cut, copy, and paste even if the menu is hidden. Hiding menu bars and using dialog box style makes Help and VI descriptions inaccessible. You can add a Help button on the front panel and design it to show the Help window or help file entries programmatically. Remember that you can abort a VI by using the following keyboard shortcuts:
  - <Ctrl-period> (Windows)
  - <Cmd-period> (Macintosh)
  - <meta-period> (Sun)
  - <Alt-period> (HP-UX)
- Hide the Abort button if the user should not abort the VI. Hiding the Abort button disables the keyboard shortcut for aborting the VI. You should provide a front-panel Boolean Stop button for VI’s that loop.

Trick: You can hide the single-stepping and execution highlighting buttons to save a small amount of execution time when the VI is finished by turning off the Allow Debugging option.
• Consider adding custom menus to your VI for advanced applications, rather than using the default LabVIEW menus.

24. The LabVIEW software shall be compiled and run as an executable (see Appendix B for proposed deployment scenario).

25. The LabVIEW “Run”, “Stop” and “Abort” buttons shall not be shown. All software stopping shall be via the “Exit” button (see Requirement 23).

4.7 Connector Panes

Put at least one input and one output on each subVI to define data flow. Error in and error out should normally always be present in your VI’s, providing ideal dataflow connections.

Hint: A common programming mistake is to ignore the state of the "error in". For many VI’s, you should encapsulate your code in a case statement, so that the VI can terminate without executing if the error in was true.

Position connections for inputs on the left and connections for outputs on the right. This conventional left-to-right data flow prevents complicated, unclear wiring patterns.

When several VI’s use the same inputs and outputs, try to place the inputs and outputs in the same location on each VI. For example, refnums are usually located at the top left and right of an icon, and error I/Os are located at the bottom left and right. Placing these inputs and outputs in these locations makes it easier to wire icons together.

On the front panel, you can edit required inputs for subVI’s by clicking on a terminal in the connector pane at the upper right side of the window and choosing This Connection is » from the pop-up menu. If the connector pane is not visible, pop up on the VI icon and select Show Connector. From the This Connection is » submenu, select Required, Recommended, or Optional. By default, inputs are all considered to be Recommended.

If you designate an input as required, it must be wired in a calling VI for the VI to work. This is appropriate for inputs such as refnums, where the VI does not make sense unless the input is wired. You should not make an input required unless it is necessary for the VI to execute properly. Required inputs appear in bold in the Help window. If you make an input optional, the Help window does not display it in simple help mode, which helps to simplify the connector pane in the Help window. With simple help mode turned off, the input appears dimmed. You should use the optional setting for parameters you rarely need to wire. You can specify whether outputs should be recommended or optional, but you cannot mark outputs as required.

26. The use of LabVIEW global variables shall be avoided.

5 Coding Standards: What Programmers See

5.1 Wiring Etiquette

Haphazard wiring can distract the user and make block diagrams difficult to follow. Align and distribute objects to make a block diagram as neat as possible (note that you can use the "align" and "distribute" menu functions with block diagram objects, and not just front panel objects). Employ symmetry and straight lines to make the block diagram easy to read. Most importantly, do not hide objects behind structures or other objects (there's nothing so annoying like wasting time looking for a terminal node only to finally realize it's behind some For Loop). The following are some general wiring tips:

• Avoid routing wires underneath structures or icons, and never route wires through an icon to a
terminal on the other side of the icon.

- Do not use local variables just to avoid having long wires, except for very complex diagrams where the wiring is too dense. Remember that every local variable that reads the data makes a copy of it in memory.

- Reduce the number of pivot points in wires by aligning the source and destination of the wires. Use the cursor keys to remove single-pixel kinks from wires. There are several little-known keyboard shortcuts that can help you fine-tune your wires; consult the LabVIEW documentation.

- Delete excess wires, such as loops.

- Evenly space parallel wires in straight lines and around corners.

5.2 Labelling

- Makes sure you type in some free-floating text inside most structures (For Loops, While Loops, Case and Sequence Structures). This helps the user understand complex segments of code.

- Use text instead of numbers as inputs to Case Structures when possible because it makes the Case statements far more readable. If your input selector is a number, then use it as an index on an array of strings constant that will allow you to provide a text input to the Case structure. Alternatively, you can use an enumerated type as an input.

- Add comments to explain the purpose of each frame. For comments, choose a font size and style that will stand out, or use the Colouring tool to give it a blindness background that stands outs.

- Always label constants because they are not self-explanatory.

- Show the label of a subVI if the icon does not describe the function of the VI sufficiently.

- Use free labels on long stretches of wire to label the signal data. You can use the colouring tool to make the label's border transparent so the wire isn't obscured.

- Paste long comments into small string constants and make them scrollable.

- Place large scrollable text items off to the side of the block diagram to avoid cluttering the screen.

5.3 Execution Sequence

The following sections describe programming concepts that will help you take advantage of the natural data flow in block diagrams.

5.3.1 Left-to-Right Layouts

G was designed to use a left-to-right and sometimes top-to-bottom layout. Your block diagrams should follow this convention. While the positions of program elements do not determine execution order, avoid wiring from right to left. Only data connections, or wires, and structures determine execution order.

5.3.2 Data Dependency

If a section of the block diagram is missing the appropriate inputs or outputs, you might use a single-frame Sequence Structure. Do not overdo it, though.

To impose a pure dataflow model just for the sake of avoiding Sequence Structures completely is as bad as overusing them. Use dataflow programming techniques to create a clear, single-page main program.

5.3.3 Sequence Structures

Avoid overusing Sequence Structures. G has a great deal of inherent parallelism. Using a Sequence Structure guarantees the order of execution but prohibits parallel operations. For instance, asynchronous tasks that use I/O devices, such as GPIB, serial, plug-in boards, can run concurrently with CPU-bound operations. Sequence Structures add no code or execution overhead, but they do restrict parallelism. Actually, your program might execute faster if you can add parallelism by reducing the use of sequences.

Sequences also hide parts of the program and interrupt the natural left-to-right flow of data.

While pure dataflow programming means avoiding Sequence Structures, there are cases where it is appropriate to use them. Use Sequence Structures only if one node must execute before another and cannot be connected by a wire. Refer to the Data Dependency section earlier in this chapter for more information.
Sequence Structures also can be used to conserve screen space, although proper use of subVI’s is better (modularity).

The best alternative to Sequence structures is a State Machine. A State Machine is a Case Structure wired to a counter in a For or While Loop. This technique allows you to jump around in the sequence by manipulating the counter. For instance, any frame can jump directly to an error handling frame. Many, if not most, LabVIEW applications should take advantage of this program structure. A simple example of a state machine is shown in the following figure.

![State Machine Diagram](image)

Figure 4. Simple state machine. Notice the use of enumerated types to clarify the state the loop is in.

5.3.4 Check for Errors
When you perform any kind of I/O, consider the different types of errors might occur. For example, if you are writing a file, what happens if the directory you are writing to does not exist? What happens if a file with the same name is already there?, etc.

Almost all I/O functions return error information. Make sure your program checks for errors and you deal with them appropriately. BridgeVIEW and LabVIEW do not deal with errors automatically because users usually want specific error-handling methods. For example, if an I/O VI in your block diagram times out, you might or might not want your entire program to halt. You also might want the VI to retry for a certain period of time. In BridgeVIEW and LabVIEW, you make error-handling decisions.

Some other examples of situations in which errors frequently occur:

- Incorrect initialisation of communication or data that has been improperly written to your external device
- Broken or improperly working external device or loss of power
- Incorrect file permissions or a lack of disk space

5.4 Optimisation
There are many things you can do to optimise memory usage and execution time of your G program. Generally an advanced topic, optimisation quickly becomes a concern when your program has large arrays and/or critical timing problems. Refer to Chapter 28, Performance Issues, in the G Programming Reference Manual for more information on optimising G programs.

Trick: Pre-allocate arrays, using the "Initialise Array" function, rather than growing their size dynamically in a loop. The performance increase can be magnitudes of order for large arrays.

Avoid using functions that create copies of large data structures (clusters and arrays).

5.5 Code Interface Nodes
A Code Interface Node (CIN) can obscure the function of your VI’s. Use CINs only when absolutely necessary. Include the following information to help your users understand what your CIN does and how to rebuild it.

5.5.1 CIN Description Contents
In the Description pop-up menu item of a CIN, or in a scrolling label next to the node, record the following information:
• Source code filename
• Platform and operating system
• Compiler and version
• Location of source code
• What the code does
• List of other files required to build the CIN
• Other critical information required to maintain the CIN

5.5.2 CIN Source Code
You should enter the same kind of information in the header file with the source code that you enter in the Description… pop-up menu item of a CIN. If the source code is not too long, paste it into a scrollable block diagram string constant.

6 Style Checklist
Use the following checklist to help you maintain consistent style and quality. You might want to copy this checklist to use on all your projects.

VI Checklist
☐ The top-level VI to run is either clearly marked as Main.vi, or you have provided a obvious shortcut (alias on the MacOS) to run the main VI.

☐ Organize VI’s in a hierarchical directory with easily accessible top-level VI’s and subVI’s in subdirectories.

☐ Give VI meaningful names without special characters such as backslash (\), slash (/), colon (:), and tilde (~).

☐ Use standard extensions so Windows and UNIX can distinguish files (.vi, .ctl).

☐ Capitalize initial letters of VI names.

☐ Distinguish example VI’s, top-level VI’s, subVI’s, controls, and global variables by saving them in subdirectories, separate libraries in the same directory, or by giving them descriptive names such as MainX.vi, Example of X.vi, Global X.vi, and TypeDef X.ctl.

☐ Write a VI description (VI Info). Proofread it.
Write a Description for all controls that will be visible to the user. Check them with the Help window open, moving the mouse over each control.

Include your name and the date in the VI description. Note: When using SourceSafe, you can have this be done automatically for you.

Create a meaningful icon.

Make test VI’s that check error conditions, invalid values, and Cancel buttons.

Save test VI’s in a separate directory so you can reuse them.

Load and test VI’s on multiple platforms, making sure labels fit and window size and position are correct.

Front Panel Checklist
- Give controls meaningful names. Use consistent capitalization.
- Make name label backgrounds transparent.
- Check for consistent placement of control names, for example, upper left.
- Use standard, consistent fonts throughout all front panels.
- Use Size to Text for all text for portability and add carriage returns if necessary.
- Use Required, Recommended, and Optional settings on the connector pane.
- Put default values in parentheses after input names.
- Include unit information in names and on any chart or graph axes.
- Write descriptions for controls, including array, cluster, and refnum elements. Remember that you might need to change the description if you copy the control.
- Arrange controls attractively, using the Align Objects and the Distribute Objects drop-down menus.
- Make sure your VI front panels aren't larger than the target platform's monitor resolution (i.e. 800x600)
- Do not overlap controls.
- Use blindness logically and sparingly.
- Use error in, error out clusters where appropriate.
- Consider other common thread controls, such as taskID, refnum, and name.
- Provide a Stop button if necessary. Do not use the Abort button to stop a VI. Hide the Abort button.
- Use rings and enumerations where appropriate. If you are using a Boolean for two options, consider using an enumeration instead to allow for future expansion of options.
- Use Custom Controls or TypeDefs for common controls, especially for rings and enumerations. Include it with VI’s.
In control VI’s, label controls with the same name as the VI, for example, Alarm Boolean.ctl has the default name Alarm Boolean.

Block Diagram Checklist

- Avoid creating extremely large block diagrams. Any block diagram should normally fit within one screen at 1024x768 resolution.
- Label controls, important functions, subVI’s, constants, attribute nodes, local variables, global variables, and structures.
- Add comments. Use object labels instead of free labels where applicable and scrollable strings for long comments.
- Make comment backgrounds transparent or a distinctive blindness to distinguish from name labels.
- Place labels below objects when possible and right-justify text if label is placed to the left of an object.
- Use standard, consistent font conventions throughout.
- Use Size to Text for all text and add carriage returns if necessary.
- Reduce white space in smaller block diagrams but allow at least 3–4 pixels between objects.
- Flow data from left to right. Wires enter from the left and exit to the right, not the top or the bottom.
- Align and distribute functions, terminals, and constants.
- Label long wires with small transparent labels.
- Do not wire behind objects.
- Make good use of reusable, testable subVI’s.
- Make sure the program can deal with error conditions and invalid values.
- Show name of source code or include source code for any CINs.
- Save with the most important or the first frame of structures showing.
- Review for efficiency, especially data copying, and accuracy, especially parts without data dependency.

Other

- When using configuration or initialisation files that your VI reads or writes to, make sure that they don’t end up as read-only files when you deliver your application to the customer. Remember that Source-Safe will mark checked-in files as read-only by default.
- If your VI’s end up moving to another operating system or another version of LabVIEW, first do a Mass Compile on your whole set of VI’s on the new system so that users don’t get the "Save Changes…?" dialog box when they quit LabVIEW or close the application.
Annexure A: Example LabVIEW Front Panel

The Front Panel demonstrates the use of LabVIEW’s Application Font, as defined in Paragraph 4.2.3
Annexure B: Proposed Software Deployment Scenario

Refer to document: SALT 1000BS0062
Annexure C: VI Checklist to assist in verifying adherence to the LabVIEW Coding Standard.

<table>
<thead>
<tr>
<th>VI Setup</th>
<th>Passed/Failed</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Give VI meaningful names without special characters, such as backslash (), slash (/), colon (:), and tilde (~).</td>
<td></td>
</tr>
<tr>
<td>2. Use standard extensions so Windows and UNIX can distinguish files (.vi, .ctl).</td>
<td></td>
</tr>
<tr>
<td>3. Capitalize first letters of VI names.</td>
<td></td>
</tr>
<tr>
<td>4. Top level VI’s should be named XXXmain.vi where XXX is the abbreviation for the computer</td>
<td></td>
</tr>
<tr>
<td>5. Local user interfaces should be called XXXMMI.vi</td>
<td></td>
</tr>
<tr>
<td>6. Defaults and parameters that may change should be stored in a XXX.ini file and read at startup</td>
<td></td>
</tr>
<tr>
<td>4. Write a VI description. Proofread it.</td>
<td></td>
</tr>
<tr>
<td>5. Include your name and/or company and the date in the VI Description on the Documentation page of the VI Properties dialog box.</td>
<td></td>
</tr>
<tr>
<td>6. Choose a connector pane pattern to leave extra terminals for later development. Use consistent layout across related VI’s.</td>
<td></td>
</tr>
<tr>
<td>7. Avoid using connector panes with more than 16 terminals.</td>
<td></td>
</tr>
<tr>
<td>8. The VI icon should be adequately describe the VI, and use the following colour scheme:</td>
<td></td>
</tr>
<tr>
<td>Generic common VI’s – Green</td>
<td></td>
</tr>
<tr>
<td>SALT System wide data Type definitions – Blue</td>
<td></td>
</tr>
<tr>
<td>IO VI’s, up to &amp; include scaling to engineering units – Red</td>
<td></td>
</tr>
<tr>
<td>Processing VI’s – Cyan</td>
<td></td>
</tr>
<tr>
<td>File Handling VI’s – Yellow</td>
<td></td>
</tr>
<tr>
<td>User Interface VI’s – Orange</td>
<td></td>
</tr>
<tr>
<td>Other Custom VI’s - Magenta</td>
<td></td>
</tr>
<tr>
<td>Top level VI - White</td>
<td></td>
</tr>
<tr>
<td>9. Use Required, Recommended, and Optional settings on the connector pane.</td>
<td></td>
</tr>
<tr>
<td>10. Consider VI and window options carefully.</td>
<td></td>
</tr>
<tr>
<td>Hiding menu bars and using dialog box style makes Context Help and VI descriptions inaccessible.</td>
<td></td>
</tr>
<tr>
<td>Hiding Abort and debugging buttons increases performance slightly.</td>
<td></td>
</tr>
<tr>
<td>Set print options to print attractive output in the most useful format.</td>
<td></td>
</tr>
</tbody>
</table>
### VI Source code control

1. When you modify a VI, use the History window to document the changes.

2. The VI should contain revision and source code control info:
   - Revision
   - Author
   - Date of last modification
   - Reason for last modification

3. CIN descriptions should include the information on the source code:
   - Source code filename
   - Compiler & version
   - Location & source code
   - Code function
   - Other files required to build the CIN

### Front Panel

1. Give controls meaningful names. Use consistent capitalization.

2. Make name label backgrounds transparent.

3. Check for consistent placement of control names.

4. Put default values in parentheses after input names.

5. Include unit information in names if applicable, for example, Time Limit (Secs).

6. In system controls, label controls with the same name as the VI, for example, Alarm Boolean.ctl has the default name Alarm Boolean.

7. Use Application font throughout all front panels.

8. Use Size to Text for all text for portability and add carriage returns if necessary.

9. Write descriptions for controls, including array, cluster, and refnum elements. Remember that you might need to change the description if you copy the control.
10. Arrange controls logically. For top-level VI's, put the most important controls in the most prominent positions. For subVI's, put inputs on the left and outputs on the right and follow connector pane terminals.

11. Arrange controls attractively, using the Align Objects and the Distribute Objects pull-down menus.

12. Do not overlap controls.

13. Use colour logically and sparingly, if at all.

14. Use error in and error out clusters where appropriate.

15. Consider other common thread controls, such as taskID, refnum, and name.

16. Use ring controls and enumerated controls where appropriate. If you are using a Boolean controls for two options, consider using an enumerated control instead to allow for future expansion of options.

17. Use custom controls or typedefs for common controls, especially for rings and enums.

**Block diagram**

1. Avoid creating extremely large block diagrams. Limit the scrolling necessary to one direction in order to see the entire block diagram.

2. Label controls, important functions, constants, property nodes, local variables, global variables, and structures.

3. Add comments. Use object labels instead of free labels where applicable and scrollable string constants for long comments.

4. Place labels above objects when possible and right-justify text if label is placed to the left of an object.

5. Use Application font throughout.

6. Use Size to Text for all text and add carriage returns if necessary.

7. Reduce white space in smaller block diagrams but allow at least 3 or 4 pixels between objects.

8. Flow data from left to right. Wires enter from the left and exit to the right, not the top or the bottom.

9. Align and distribute functions, terminals, and constants.

10. Label long wires with small labels with white backgrounds.

11. Do not wire behind objects.

12. Make good use of reusable, testable subVI's.

13. Make sure the program can deal with error conditions and invalid values.

14. Show name of source code or include source code for any CINs.

15. Save with the most important or the first frame of structures showing.
<p>|   | Review for efficiency, especially data copying, and accuracy, especially parts without data dependency. |</p>
<table>
<thead>
<tr>
<th>Block diagram</th>
<th>Passed/Failed</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. The exit via the stop button should shut the program/system down in an orderly and safe fashion.</td>
<td></td>
</tr>
<tr>
<td>2. VI’s should perform the function required in the software design document.</td>
<td></td>
</tr>
<tr>
<td>3. Variable names should follow the a convention.</td>
<td></td>
</tr>
<tr>
<td>4. Appropriate Engineering units should be used.</td>
<td></td>
</tr>
<tr>
<td>5. Data Types should be adequate to give the precision required using Engineering units.</td>
<td></td>
</tr>
<tr>
<td>6. Dataflow – Is it possible that program execution will halt due to data dependency and data being unavailable.</td>
<td></td>
</tr>
<tr>
<td>7. A thread or task should not be allowed to starve others of CPU time.</td>
<td></td>
</tr>
<tr>
<td>8. Error handling – There should be error handling routines to handle errors, particularly with IO.</td>
<td></td>
</tr>
<tr>
<td>9. The program should startup in a safe state.</td>
<td></td>
</tr>
</tbody>
</table>