LAVA Group's Suggestions for LabVIEW Improvements

1. GOOP-Related

1.1. Better LV-native support for EVENT-driven OO programming in general.
1.2. Provide an "Object Browser" to interactively browse objects in existence (similar to GOOP Wizard Class Inspector, but covering ALL classes). This could be linked to improved Probe functionality, so that probing a Refnum wire could highlight the instance of the object in the Object Browser where it then may be possible to view the values of the object's attributes.
1.3. Implement Refnum Controls that allow for the selection of a single object instance from drop-down list of all objects of that class in existence (similar to the new VISA Session control).
1.4. Find a way to implement class inheritance in LV objects (for both methods and attributes). In essence, making a LabVIEW++ environment.

2. Interoperability

2.1. Provide native XML support and utilities in LV (e.g.-G Data to XML and XML to G Data).
2.2. Provide full DOM parser support for XML documents.
2.3. Interface with other external environments (e.g.—like JAVA Messaging Service, JMS).
2.4. Ability to compile a VI as an ActiveX control (or platform specific equivalent for non-Windows O/S), embeddable in other non-LV applications.

3. Security

3.1. Add SSL encryption security into Data Socket, LV Web Server, and LV VI Server functionality.
3.2. Provide LV-API for user-login utilities with support for multiple user levels.

4. New Concepts

4.1. Allow for fast OO DB to provide PERSISTENT storage (configuration data) of attributes for application objects (e.g.—attributes of front panel objects: Visible?, Position, Caption, etc.). This will enable data-driven, configurable applications, as well as better recoverability from "crashes".
4.2. Allow for saved "Workspaces" or "Projects" to save and recall the state of all VI's, their window positions, running state, etc. This concept can be expanded to allow for multiple simultaneous workspaces and leads to the possibility of having different versions of the same named VI in the different workspaces.

4.3. Ability to dynamically create controls and indicators at run-time and place them on a VI's front panel. Manipulation of the control/indicator in question would only be possible by reference using Property and Invoke Nodes.
4.4. "Windows-In-Windows" – the ability to embed the front panel of a VI into the front panel of another. This effectively makes it possible to have custom controls that have G code behind them, and is similar to the concept of ActiveX control containers, but pure G.
4.5. Programmatically controllable "Macro Recording" (analogous to Excel Visual Basic for Automation) where the sequence of a user's actions can be "played back" (e.g.—Controls clicked, text entered, menu items selected, etc.).
4.6. Provide a way to programmatically call different DAQ configuration files, like is possible in MAX (and BridgeVIEW with tag files).
4.7. Diagram constants that can be referenced anywhere on a vi's diagram, providing analogous capability to a #DEFINE macro in C. For example, a LV programmer, in a motion control app, may want to reference the encoder counts (in lines per inch) of a motor axis several places in a vi. It may be desired to not have this quantity modifiable by the end user, but if the encoder resolution changes, the programmer should only have to change 1 constant to update the entire vi (rather than search the vi's diagram for related constants, and have to make numerous corresponding changes).
4.8. The addition of radix-like portion of a String control/indicator/constant to show the display format (e.g.—normal, slash codes, hex, password, etc.).
4.9. Providing an easier (and more usable) way to "comment out" LV code. Many programmers already know to use a Case Structure with a boolean constant connected to the Selector terminal to accomplish this; however the compiler will still complain about broken wires and "unrunnable" code in the "commented-out" case that is specifically intended NOT to run.
    4.9.1. Closely related to this (but not necessarily the same), would be the ability to "Ignore This Case/Frame" of a Case or Sequence Structure. This would mean that the compiler would disregard the contents of that Case/Frame, and it would never be executed at run-time.
4.10. Provide for "events" associated with front panel controls (perhaps run "call-back vi's whenever the user accesses a control, analogous to CVI call-back functions).

5. General

5.1. A VI's Menubar refnum should be available from the VI Property Node.
5.2. Better run-time localization support for displaying appropriate labels/captions based on a "global" language setting.
5.3. Include "Break" functionality in a For Loop.
5.4. Build an optional "Wait ms" into the corner of every loop where the timer value can be changed in Edit mode.
5.5. Provide programmatically selectable option for Controls and Indicators to have appearance and behavior of native OS platform. For example, making selections

in a LV lisbox (shift-click) is different than working with all other Windows listboxes (control-click).

5.6. Make it possible to compare 2 VI's with the same name from different directories.

5.7. Add support for "collapsible" menus like those in Office 2000 applications.

5.8. Add Error In and Error Out terminals to LV primitives like Wait ms to allow for data dependency and minimize the need for Sequence Structures to force order of operations.

5.9. Allow X-Y and Waveform Graph controls to accept data input from the front panel. In early versions of LV (< v2.52) the user could <ctrl>-draw with the mouse (the cursor would become a pencil) to enter data into a graph control.

5.10. Having an intelligent mechanism to set the number of significant figures (not just precision) for a numeric control/indicator (like the old Fortran "G" format). The only way to do that now is by making G code that looks at the value, along with the intended number of significant figures, does a bunch of log/exponent calculations with some heuristics, and then sets the precision property on the control/indicator. Although, the "E" notation basically does this, there are some applications where most operators do not understand the E notation and it is inappropriate.

6. Improved Debug Tools

6.1. Ability to select a wire and somehow trace its dependency throughout the entire VI hierarchy.

6.2. Better / more configurable Probes:

6.2.1. Ability to INSERT a value onto a wire when debugging.

6.2.2. Probes with "Code" behind them, so the LV programmer/user can specify the debug behavior.

6.3. Conditional Breakpoints—break only if value is outside a specified range.

6.4. Better documentation of advanced debug tools.

7. Improved Editor Features

7.1. As a general rule if multiple objects are selected, and an edit is applied, all selected objects should exhibit corresponding changes. As a few limited examples:

7.1.1. When selecting several Front Panel Objects and applying an attribute change, that change should be applied to all selected Front Panel Objects (e.g.—"Advanced→Hide Control" would hide ALL selected controls/indicators and "Visible→Label" would hide/show labels of ALL selected control/indicators).

7.1.2. When selecting several Front Panel Objects and resizing one, each selected object would be resized proportionally.

7.1.3. When selecting several Property Nodes on a diagram and changing the attribute selection from A to B, all selected Property Nodes with unwired terminals of attribute A should be changed to B.

7.1.4. When selecting several Property Nodes on a diagram and expanding a node to add terminals, the same number of terminals should be added to all selected Property Nodes.

7.2. Somehow indicate on a case structure, sequence structure, etc. when diagram objects are "hidden" (i.e. outside the bounds of the case or frame). This would essentially be a graphical indication on the diagram of the same information that is already displayed in the Error List box as a hidden object Warning.

7.3. Ability to show wires hidden behind VI's and other diagram structures.

7.4. Be able to resize objects using cursor keys.

7.5. Implement the inverse of "Make Space" (Control-Drag on diagram)—perhaps Shift-Control-Drag could be used to "Remove Space".

7.6. "Re-wiring" functionality—perhaps Control-Click on a wired terminal with the Wiring tool could detach the wire from that terminal and make it available to be rerouted somewhere else.

7.7. "Wire Swapping" functionality—perhaps after starting the "Re-wiring" (described above), if the programmer Control-Clicks on a second wired terminal, the two wires would be swapped in their connections to their respective terminals.

7.8. Make auto-wiring/re-linking/replacement more intelligent. Auto-(re-)wired wires should connect only to terminals of appropriate datatypes (not just to "blindly" mapped terminal numbers).

7.9. Be able to copy and paste LV code from one instance of a LV development environment into another (put the LV code/objects information into the system clipboard, not just a bitmap image).

7.10. When using the Property Node to access a reference to an object in that vi's object hierarchy, allow access to all objects directly without having to use other Property Nodes to descend down the hierarchy. For example, this could work similar to how the Unbundle By Name handles sub-clusters (i.e. [<level 1 object sub-class>:<level 1 object name>] . [<level 2 object sub-class>:<level-2 object name>] . […]).

7.11. Similar to the concept above for obtaining an object reference, it would be nice to have some type of textual "dot-notation" descriptors that could be used to obtain references to objects.

7.12. Have a way to auto-wire (duplicate wiring) from one case in a case structure to all other cases with unwired output tunnels. Quite often data needs to be wired straight through a case structure with many cases (i.e. 30+), because only a few cases actually operate on that data. If a new data element is wired straight through the case, a new output tunnel is created, and then the input must be wired to the output tunnel manually for each and every case; it would be very helpful to automate this.

7.13. Resizeable locals and globals to help manage their footprint on the diagram—perhaps with a "size to text" feature.

7.14. A large number of applications require right-justified numbers. It would be nice to have an easier way to right-justify numeric controls/indicators (while maintaining left-justified labels/captions)—perhaps a preference setting for "right-justified numerics by default".
7.15. A global "Find and Replace" feature for LV objects as well as text.
7.16. Add the features "Save As . . ." and "Replace . . ." to the pop-up menu available in the Hierarchy Window. Replacing from the Hierarchy Window should replace all instances where that VI is called as a sub-VI.
7.17. When Replacing from the diagram, it would be nice to have a list of items used in previous, recent replacements, similar to the concept of the "Recently Opened Files" submenu.
7.18. Having a "Grid" (like very early Mac-only versions of LV) available on front panels and diagrams to help with alignment and visual layout.
7.18.1. Allow for preferences to Hide/Show Grid.
7.18.2. Allow setting for Grid Resolution (e.g. – 0.1in, 0.25in, 10pxls).
7.18.3. Have the ability to "Snap Objects to Grid".

8. Undesired Behaviors

8.1. When duplicating a case or frame, terminals are "copied and pasted" as new terminals. It would be preferable to have the terminals become local variables referenced to the original terminal.
8.2. The Undo/Redo application menu items should be available in a built application to work on user data manipulations.
8.3. Enhance Bug Reporting using recognition as incentive. National Instruments benefits greatly when people take the time to report and explain a bug to an AE. So, it would be beneficial for NI to give some reason to the developer for the bug report. This may be as simple as noting that this CAR was started by *Joe Developer,* as is already shown in NIs database.
8.4. In LabVIEW 6, it has been widely observed that there are a significant amount of "Insane Object" errors. This appears quite often when employing typedefs within clusters and using Unbundle By Name on the diagram. It is VERY DIFFICULT to know which is the offending object; on a complex diagram this usually amounts to a tedious trial-and-error task. This error is immensely troublesome to the programmer, but even more disconcerting when seen by the end-customer of the code. If the underlying cause cannot be identified and fixed, at a minimum, perhaps an option to Delete/Remove Insane Objects could be provided.
8.5. Front panel decorations should not be selected, unless selecting the entire object (similar to clusters).
8.6. "Text.Text" property of a String Control/Indicator only works properly if the String's Display Format is "Normal" or "'\' Codes". It does not provide the proper text sting when the String's Display Format is "Hex" or "Password".

9. Examples and NI provided G Code

9.1. VI's with password protected diagrams should have THOROUGH documentation as to what that VI does. If the diagram is unavailable to see how the code works, it is essential to know what it is supposed to do and what its expected behavior should be, along with any limitations on input/output values.

9.2. The NI provided examples that ship with LV should be cleaned up. The large amount of spaghetti code in the shipping examples sets a BAD precedent with novice users.