LAVA Feedback
to the
LabVIEW Development Team

NI Week 2002

Table of Contents

Open Letter to the LabVIEW R&D Team 4
Last Year's List 6
Priorities 6
1.0 Enhancements to G 6
1.1 More native support for OOP (including "active objects") 6
1.2 Introduce TRUE Object-Oriented extensions to G. 6
1.3 User-Defined Events 6
1.4 Ability for G programmer to define their own variable and wire types 6
1.5 Conditional Node on Build Array Tunnel 7
1.6 Allow instantiation of controls and indications at run-rime 7
1.7 LabVIEW Native Tree Control 7
1.8 Additional Array Functions 7
2.0 Enhancements to Editor 7
2.1 Finding *Write* Global and Local variable instances 7
2.2 Relink All Callers to SubVI 7
2.3 New VI/Create Sub VI using a template 8
2.4 Support meaningful Probes on RefNum? wires (that show the Cluster instead of the RefNum? Hex value) 8
2.5 Auto Add Shift Registers for Refnums on Loops 8
2.6 Auto Add Shift Registers for Error In/Out on Loops 8
2.7 Open VI in "safe" mode option 8
2.8 Control Option for Smarter Add-Input 9
2.9 A "Run when loaded" VI option is needed 9
2.10 Allow Menu-Launch VIs to not show their panels 9
2.11 Show more than five icons across in This VI's SubVIs 9
2.12 Make *Edit Format String* dialog tool available from all functions or VIs that use format strings 9
3.0 LabVIEW Data Tools 9
3.1 Update App Note 154 9
3.2 Publish Official ClassID and Data Type strings/enumerations 9
3.3 Better (required) tools for working with variants 10
3.4 Type Descriptor Parsing Tools (or equivalent functionality) 10
3.5 Polymorphic (Undefined) Data Type 10
3.6 Better XML Parsing 10
4.0 Source Code Control 11
4.1 Saving VI in text mode 11
4.2 Unnecessary recompilation of callers ignored as an actual modification. 11
4.3 Integration with CVS 11
5.0 Call Library Functions 11
5.1 Call Library Functions Like property nodes 11
5.2 Other Call DLL Issues 11
5.3 Call Library Function by Reference, which could push linkage errors to run-time 11

6.0 Tools for Tools Developers 12
6.1 Programmatic Interface to the Application Builder 12
6.2 Add LabVIEW directory as possible root for VI search path 12
6.3 Create a better way to generate custom palettes 12
6.4 Allow "App.MenuLaunchVI" to work for File (.\wizard) and Help (.\help) menu VIs as well as Tools menu (.\project) 12
6.5 An application method to refresh menus 12
6.6 An application method to refresh palettes 12
6.7 Make App.MenuLaunchVI public 13
7.0 Enhanced Hierarchy Window Features 13
7.1 Replace All 13
7.2 Open diagram 13
7.3 Relink All 13
7.4 Rename VI 13
8.0 Visual Studio like Project View Window 13
9.0 Other Items 14
9.1 Make the Array Size(s) and Dimension an array property 14
9.2 Better Error Handling 14
9.3 Introduce a standard set of VIs for Publish/Subscribe 14
9.4 Support bundling/unbundling for Cluster-typed RefNums? 14
9.5 Large Files support 14
9.6 VI.FP.Menu_Refnum VI Server Attribute 14
9.7 Expose control properties and methods to ActiveX VI Server interface 14
9.8 Add a digital display for refnums 14
9.9 Allow to build application reusing previous tlb 15
9.10 Option to show an icon on the Front Panel for Typedefs 15
9.11 Build Custom ActiveX Server/Control 15
9.12 Support to build a LabVIEW Application as an NT/XP Service 15
9.13 Fix problems with occurrences 15
9.14 Optional Scroll Bars on Clusters 15
9.15 Manage custom file extensions in a LabVIEW application 16

Open Letter to the LabVIEW R&D Team

LabVIEW R&D Team,

Greetings form the LAVA Group! For those who may not have heard of LAVA (LabVIEW Advanced Virtual Architects), we are strong NI supporters and long-time LV users/programmers who greatly appreciate the efforts that the LV R&D Team makes to create a truly unique, intuitive, innovative, and powerful programming environment that we all enjoy working with.

In the Northern California Bay Area several of the long time LV users and consultants decided to get together in an effort to exchange ideas and discuss design patterns of how to use LabVIEW up to (and sometimes beyond) its limits—hence, LAVA was born. We have been meeting quarterly for nearly 2 years now, and we have been pleased to see some of you at our gatherings.

In preparation for NI Week and in an effort to help support the LV development team by providing prospective from "the field", the LAVA group tasked itself with compiling a list of suggested improvements that we would like to submit to the LV development team. We hope that this list is received in the spirit intended--an effort by a senior user community to provide strong, quality, positive feedback to the LabVIEW development team to help the product grow and mature even more.

Similar to the list we provided last year, our goal is to help the LV development team continue that tradition of continuous improvement and incorporation of novel ideas into LV, creating an even more awesome and useful tool than the wonderful one we all enjoy today. Some suggestions in this year's list are carryovers from last year's list, and we have included that list as an attachment for reference. However, this year we tried to stay away from "bug reports" and focus solely on new functionality.

You may find some of these ideas and suggestions are already addressed in your current development. Although many in the LAVA group are under NI non-disclosure agreements, not all attendees are. Therefore, we decided not to remove items from the list if they were not generally known outside NI and/or non-disclosure agreements. Please do not let this diminish the impact or importance of the remaining suggestions.

We all recognize the NI Week schedule is very full and that the LV R&D Team is usually flooded with this type of information during the conference. However, if you would like clarification or further explanation of what was intended by a certain suggestion, or if for any reason would like to discuss the document, LAVA representatives attending NI Week would be delighted to help in any way we can.

Keep up the fantastic work!

Respectfully,
The LAVA Group

(in alphabetical order from the mailing list: Adam Rofer; Alan Hilton; Ali AlHasan; Allen Smith; Andrew Johnson; Andy Cordes; Bhavnesh Patel; Brad Hedstrom; Brooks McDonald; Bryan Moore; Carl Thompson; Chris Lynch; Christophe Salzmann; Dana Redington; David DeLong; David Weisberg; Dirk DeMol; Dmitry Sagatelyan; Eric Lyness; Eric Low; Eric Nehrlich; Gary E. Helstrom; Gary Johnson; Grace Lim; Herman Griffin; Jack MacKrisken; James Kring; Jamie Smith; Jason Dunham; JC Flores; Jed Davidow; Jeff Long; Joe Damico; John Trager; Kevin E. Smith; Kevin Thompson; Kim Powers; Luckshman Parameswaran; Mark Borodkin; Mark Naley; Martin Vasey; Meg Kay; Michael Monroe; Michael Rushford; Michael Simoneau; Mike Zelinski; Otto Dalmady; Pablo Echavarria; Paul Daley; Peter Tam; Pinyen Chen; Ray Merrill; Richard Jennings; Rosie Abriam; Ryan Talbot; Scot Hannahs; Sorin Grama; Stan Case; Steve Jurovich; Thomas Clark; Tim Dense; TJ Robertson; Todd Gardner; Vance Socci; Veda Murthy; Wayne Larson)

Last Year's List

In an effort to avoid duplication, some items from last year's list that are still not addressed in shipping versions of LV were not put on this year's list, but deserve mention. Please reference the following sections from last year's list: Interoperability 2.2 and 2.3; Security 3.1 and 3.2; New Concepts 4.1, 4.5, 4.7-9; General 5.5-10; Improved Debug Tools 6.1-4; Improved Editor Features 7.1-18; Undesired Behaviors 8.2-6; Examples and NI Provided G Code 9.1 and 9.2.

Priorities

Each suggestion in this year's list is ranked with a priority of 1 (nice to have) to 5 (don't want to live without it). The priorities represent "importance" in the eyes of the LAVA group, not necessarily urgency. We recognize that the LV R&D Team may be able to implement some of these suggestions easier than others, and only the LV R&D team can best determine an implementation schedule that suits its development plan. In each section the suggestions have also been sorted according to priority for easier interpretation.

1.0 Enhancements to G

1.1 More native support for OOP (including "active objects")

Priority: 5

See Section 1 from last year's list. Additionally, however, a consideration was discussed to allow for "active objects". That is to say that sometimes it is desirable to define an "object" by more than just its collection of attributes and methods—it's purpose may require the object itself to be "alive", executing according to its own timing, not just when calling code invokes an object's method. Monitoring objects and timer objects are good examples.

1.2 Introduce TRUE Object-Oriented extensions to G.

Priority: 5

It should be integrated into the compiler. Currently, GOOP is not true OOP. Support Single Inheritance, Polymorphism (the To More Specific/Generic Class type has nothing to do with polymorphic VIs), and Data Encapsulation/Hiding.

1.3 User-Defined Events

Priority: 5

Event Structure should be able to handle user/G programmer-defined Events.

1.4 Ability for G programmer to define their own variable and wire types

Priority: 4

The LV environment is wonderful at helping G programmers write "correct" programs by enforcing strict type checking in its wires. This type checking is

currently restricted to a variable's data structure. Take for instance a DBL that can be sub-typed as a physical quantity of Volts. It would be invaluable to allow the G-programmer to control-click on the terminal, define an application relevant sub-typing (like "Pressure Transducer Reading"), and have the LV environment "keep track" of that variable's additional (user defined) sub-typing to ensure no "cross-wiring" (without explicit type conversions/casting).

1.5 Conditional Node on Build Array Tunnel

Priority: 4
This feature would allow selective construction/filtering of an array inside of a While or For Loop. The idea is to be able to add a Conditional Node to an auto-indexing exit tunnel of a Loop. If the value is True then the element is appended to the array. This could be made fast, by only removing elements after the Loop is finished executing. Sum up the number of TRUE elements in the Boolean Conditional Node array and initialize an array of length = Sum(TRUE elements). Then fill the new array with the corresponding elements in the original, larger array from the auto-index node.

"Conditional Node of Build Array Tunnel" works just like a Shift Register and Build Array Function inside true case of Case Structure. This would be tremendously helpful

1.6 Allow instantiation of controls and indications at run-rime

Priority: 4
See item 4.3 from last year's list.

1.7 LabVIEW Native Tree Control

Priority: 3

1.8 Additional Array Functions

Priority: 3

- An **Empty Array?** function that returns TRUE when at least one dimension of the array is 0.
- A **Sorted Array Indices** function that returns an array of indices to sorted elements instead of a sorted array.

2.0 Enhancements to Editor

2.1 Finding *Write* Global and Local variable instances

Priority: 5
It would be great to search for only the writeable globals (data sinks vs. data sources)

2.2 Relink All Callers to SubVI

Priority: 5

This would save one the hassle of right-clicking on every instance of a VI in a hierarchy in order to relink them all. Yes, this is a dangerous feature, but like ***Remove Bad Wires***; it has a time and a place.

One should be able to see this option (if bad linkage) by right-clicking on the VI in the *hierarchy window as well as in a Calling VIs diagram

2.3 New VI/Create Sub VI using a template

Priority: 4
I don't use 'New VI' or ‘Create Sub VI from selected diagram code' because they create awful VI's. I always have to change the palette, add error in/out, and change the icon. It would be helpful if you could define a VI template as being the LabVIEW standard template for New VI's and if possible have the 'Create Sub VI' tool use this template as closely as possible (only deviate if the template doesn’t have enough input/outputs).

2.4 Support meaningful Probes on RefNum? wires (that show the Cluster instead of the RefNum? Hex value)

Priority: 4
There is some debate on whether it is better to see the reference value or the data to which that reference refers. These are necessarily mutually exclusive options; so, perhaps an implementation could be found that allows the user to specify when the probe is created which they need at that time.

2.5 Auto Add Shift Registers for Refnums on Loops

Priority: 4
When a refnum is wired into a loop (for or while) a shift register should automatically be created (similar to how passing data out of a for loop automatically creates an autoindex node). Currently if you wish to properly refnums through loops you must manually create a shift register every time.

If a shift register is not used a null reference will come out of the output tunnel if a For Loop executes zero times! This is bad.

2.6 Auto Add Shift Registers for Error In/Out on Loops

Priority: 4
When an error cluster is wired into a loop (for or while) a shift register should automatically be created (similar to how passing data out of a for loop automatically creates an autoindex node). Currently if you wish to properly propagate errors through loops you must manually create a shift register every time.

2.7 Open VI in "safe" mode option

Priority: 4
Open without running any code (e.g. "Run when Opened" and CINLoad()) to examine a VI from untrustable source.

2.8 Control Option for Smarter Add-Input

Priority: 3

A build array node could have any number of inputs, wired or unwired. If one were to hover over any portion of the build array node and presses the <CRTL> key, an additional input terminal would be added (up/down arrows) by growing the build array node up and down equally. The position of the wiring tool after the node is grown to include the new input terminal defines the position of that new terminal. Existing terminals are moved up or down (up arrow by wires) as the new terminal displaces them. The new terminal always which remains under the wiring tool as it moves up and down over the node. If the user wires to the new terminal it remains, but if the user releases the <CRTL> key the node reverts back to its original size.

2.9 A "Run when loaded" VI option is needed

Priority: 3

This would allow a VI to run without having its panel opened

2.10 Allow Menu-Launch VIs to not show their panels

Priority: 3

VIs launched from the Tools, Help, or File menus should be allowed to not show their Front Panels. This is different than the ability to open as reentrant because the menu launch feature does not have this option.

2.11 Show more than five icons across in This VI's SubVIs

Priority: 3

Should be very simple to add this feature, it's unfortunate that it has not been updated now that people are writing complex programs with hundreds of subVIs.

2.12 Make *Edit Format String* dialog tool available from all functions or VIs that use format strings

Priority: 2

For example, the **Array to Spreadsheet String** and **Spreadsheet String to Array** functions, and the **Write to Spreadsheet File.vi** and **Read from Spreadsheet File.vi** VIs. Yes, it is probably tricky for NI to add it to VI's that use format strings, but it should be a piece of cake to add it to functions.

3.0 LabVIEW Data Tools

3.1 Update App Note 154

Priority: 5

This should be updated when new data types are added to LabVIEW. It is currently lacking variants 0x53, waveforms 0x54, and refnums, 0x70.

3.2 Publish Official ClassID and Data Type strings/enumerations

Priority: 5

Many developers create their own enumerated constants and their code is then not interchangeable. There are a lot of wasted, duplicated efforts here. LabVIEW should ship with a VI or Object Attribute that outputs a string or enumeration of the *ClassName? as well as the ClassID.

3.3 Better (required) tools for working with variants

Priority: 5
These should be like what Jim, Mike and Mark showed at the last LAVA meeting.

3.4 Type Descriptor Parsing Tools (or equivalent functionality)

Priority: 5
There should be tools such as:

- Get Enumeration names from Type Descriptor Array
- Get Elements from Cluster Data/Type Descriptor
- Get # Elements in Cluster from Type Descriptor
- Get Elements from Array Data/Type Descriptor
- Get # Elements in Array from Type Descriptor (note: actually, the number of elements of an array is not stored in the type descriptor but in the flattened data. JPD)

3.5 Polymorphic (Undefined) Data Type

Priority: 5
A polymorphic data type could be wired to polymorphic inputs of LabVIEW primitives or other polymorphic typed inputs (controls) of SubVIs. It could then be used as a terminal in the connector pane of a VI. Its type would be defined at edit-time when it is used as a subVI and the parent wires a strict type into it and that type is compatible with the primitive wired to the Polymorphic control.

The usefulness of this idea is that you can create "true" polymorphic VIs in LabVIEW. It can be frustrating to use LabVIEW's present polymorphic VI "wrapper" around code. Especially where the only difference between several polymorphic VI members is the data type of a control that is wired up to a polymorphic input of a LabVIEW primitive in the diagram in *exactly* the same way as all the other polymorphic VI members.

3.6 Better XML Parsing

Priority: 4
Although **Flatten to XML** and **Unflatten from XML** functions showed up in LV6.1, the implementation does not allow LV users to take advantage of one of the major reasons why developers would want to use XML, namely "interpretability". The **Unflatten from XML** function mandates that ALL data elements MUST be in the XML string otherwise it throws and error. This makes it no better than the **Unflatten from String** functionality. The behavior should be to "interpret" the XML string and extract the desired data from it. For instance it

should use the “type” input as the data structure and default values, which would then be replaced by any matching content from the XML string.

4.0 Source Code Control

4.1 Saving VI in text mode

Priority: 5
Without compiled code for easier integration with line based SCC software.

4.2 Unnecessary recompilation of callers ignored as an actual modification

Priority: 5
An alternative to the "recompilation" SCC control problem might be to have an option to save a VI with no object content (i.e.--only "source code"--front panel and diagram)

4.3 Integration with CVS

Priority: 4
Many developers are using CVS. This is definitely true with the Open Source community that is using SourceForge.net CVS repository services.

5.0 Call Library Functions

5.1 Call Library Functions Like property nodes

Priority: 4
Property nodes are a perfect combination of icon and text with clear meaning to almost any level of programmer. The call library node’s icon has little meaning and all the configuration data is hidden and unprintable from within LabVIEW. Call library nodes should be made to look more like property/invoke nodes, with configuration/parameter information visible on the block diagram.

5.2 Other Call DLL Issues

Priority: 4
Make it easier to change the DLL path in a CLN when many nodes in memory use it. Currently in 6.0, if you change the path to a DLL when many CLN use it, it always reverts to the one in memory, without message. And the DLL file stays reserved as long as the CLN are in memory so it can't be replaced on file after recompilation unless VIs are removed from memory. The Open/Close DLL suggestion, mentioned below, would solve that.

5.3 Call Library Function by Reference, which could push linkage errors to run-time

Priority: 3
I want to open a reference to a DLL, based on a path argument, on disk at run-time. This would work similar to the Call By Reference. This would push linking errors to run-time, instead of at edit time. I am doing development on a PC for a

project that will run on a Unix system and makes hundreds of calls to a Shared Object file. It takes 5 minutes just to open my project because LabVIEW keeps searching for the .so file.

6.0 Tools for Tools Developers

6.1 Programmatic Interface to the Application Builder

Priority: 5
No user should be required! Always build a programmatic interface to a tool first, and then build the user interface.

6.2 Add LabVIEW directory as possible root for VI search path

Priority: 5
In addition to <vilib>, <userlib> root paths, add <labview> so that we can form search paths from LabVIEW directory such as <labview>\project\mytools* or <labview>OpenG.lib*

6.3 Create a better way to generate custom palettes

Priority: 5
.mnu files and the palette editor are very hard to use. It would be nice if there were a way of overriding only portions of a default palette rather than having to override an entire default palette. It would also be nice to have a way to programmatically modify the palettes (.mnu).

6.4 Allow "App.MenuLaunchVI" to work for File (.\wizard) and Help (.\help) menu VIs as well as Tools menu (.\project)

Priority: 5
For example, I have a class of VIs that I would like to move from the Tools menu to the File and Help menus but I cannot, because they use the App.MenuLaunchVI property to operate on the VI from which they are launched. For example:
Rename VI As… Does a programmatic Save As (rename) and then deletes the original on disk
Open (Suppress Run When Opened)… Suppresses the run when opened VI attribute so that a VI may be opened for editing
Save and Open Copy As… Saves a copy of the VI and then opens the copy so that one doesn't accidentally edit the original.

6.5 An application method to refresh menus

Priority: 3
When a tool is installed in .\project (or .\help or \wizard), the Tools menu is not refreshed until LabVIEW is relaunched. Provide a method to refresh the menus.

6.6 An application method to refresh palettes

Priority: 3

When a VI is installed in a directory that is synchronized with a palette, it is not refreshed until LabVIEW is relaunched. Provide a method to refresh the palettes.

6.7 Make App.MenuLaunchVI public

Priority: 2
It is currently exposed in the web-publishing tool, but it would be nice to make this public.

7.0 Enhanced Hierarchy Window Features

7.1 Replace All

Priority: 4
Should work like the right-click replace feature available on the diagram

7.2 Open diagram

Priority: 4
ctrl-dbl-clicking a VI icon in the hierarchy window should open a VI's diagram, like it is already done when ctrl-dbl-clicking a subVI icon in diagrams.

7.3 Relink All

Priority: 4
Right-clicking on a VI icon in the hierarchy window should show an option for Relinking all instances in memory that require relinking. The Hierarchy window should show whether some instances of a VI require relinking.

7.4 Rename VI

Priority: 3
One should be able to rename a VI in memory via the File menu as well as by right-clicking on it from the hierarchy window. A rename should rename the VI on disk, optionally deleting the original.

8.0 Visual Studio like Project View Window

Priority: 5
In Visual Studio you have a project window where you can traverse into Objects and access member functions/variables. A window like this which allows you to traverse into directories (or objects as we start going that direction) would be very helpful. From such a window you could add all kinds of features, such as right clicking on a VI and renaming it (since all VI's in the project are in the project window you can load them before renaming the VI to ensure a successful update). Another feature might be to right click on a directory/object and "Add New VI" to this object/directory; when this happens you can load a VI template which can be defined for every object/directory or just a globally defined VI template.

Such a window will provide us with a view of how our code is organized. It will encourage us to organize our code into appropriate directories and name our VI's

well. It will also be a good starting point to begin looking at directories as objects instead of just collections of VI's. From there it will be a natural transition to Object Oriented programming.

9.0 Other Items

9.1 Make the Array Size(s) and Dimension an array property

Priority: 5
When using array refnums, the only way to recover the array size(s) is to parse the flattened data, which involves making a copy of the array data. That is inefficient for large arrays. Size(s) should be an array property. It would also be nice to have the dimension of the array available as a property.

9.2 Better Error Handling

Priority: 5
Error Handling: support a better way of sharing Error Code Spaces between LabVIEW/Toolkits/Libraries/Other Development Systems. Something along the lines of Dmitry Sagatelyan's "Design Pattern for Error Handling in LabVIEW Applications".

9.3 Introduce a standard set of VIs for Publish/Subscribe

Priority: 5
This is much better than forcing users to use Queues to implement such functionality since most people would do it in different ways.

9.4 Support bundling/unbundling for Cluster-typed RefNums?

Priority: 4

9.5 Large Files support

Priority: 3
Support handling of very large files (over 2 Gb in size) in LabVIEW.

9.6 VI.FP.Menu_Refnum VI Server Attribute

Priority: 3
A VI's front panel menu refnum should be available from the VI property node--it currently is not.

9.7 Expose control properties and methods to ActiveX VI Server interface.

Priority: 3
The ActiveX interface currently does not expose more than the first couple Layers of VI server.

9.8 Add a digital display for refnums

Priority: 3

9.9 Allow to build application reusing previous tlb

Priority: 3

When rebuilding an application that is ActiveX enable, new GUID and tlb is generated each time. Allow to reuse a tlb so that we can cope with a unique ActiveX server. Also, when uninstalling an application, unregister the ActiveX server.

9.10Option to show an icon on the Front Panel for Typedefs

Priority: 3

A user-set option to show an icon on the Front Panel instead of the actual Control/Indicator instance for a given Typedef. Very useful for nested typedefs. Same for constants on the diagram—would save a lot of diagram space when one does a Bundle by Name.

9.11Build Custom ActiveX Server/Control

Priority: 3

Allow to build a set of VIs as an ActiveX Control, a COM/DCOM component, an Enterprise JavaBean? AND/OR as a platform-independent LabVIEW-specific Component.

9.12Support to build a LabVIEW Application as an NT/XP Service

Priority: 3

9.13Fix problems with occurrences

Priority: 3

Occurrences have been a problem for me for some time. I mentioned this in some info-labview discussions, which you can find at: (http://messages.info-labview.org/2000/05/01/03.html)

The first problem is that I think the occurrences need to be resettable from run to run. Everyone argues that they are like a constant but the essence of these structures is the time that the occurrence is set and it is impossible to tell all wait on occurrence functions to ignore any occurrences before some predetermined time (like the last time the code was started.) I would prefer a "reset occurrence" function that would allow me to 'erase' all previous set occurrences at the start of a run so that occurrences set in a previous execution would not interfere with the current one.

Another problem is that the wait on occurrence occasionally misses a set occurrence completely. This is clearly a bug in the occurrence code. I have seen this numerous times and it has caused enough trouble that I sworn off using occurrences and gone back to polling.

9.14Optional Scroll Bars on Clusters

Priority: 2

This would enable resizing very large clusters so that they don't take up much front panel space, but still allows viewing the hidden data within them.

9.15Manage custom file extensions in a LabVIEW application

Priority: 2

When a user double-clicks a file of a certain extension, it would be nice to have tools to register and handle that file within a built executable, or even in the development environment. The Event structure would be a good place to forward and intercept these Explorer events.