LabVIEW

LabVIEW	
Developer(s)	National Instruments
Initial release	1986; 33 years ago
Stable release	LabVIEW NXG 2.1 LabVIEW 2018 / May 2018; 10 months ago
Preview release	NXG 3.0 Beta1
Written in	C, C++, .NET
Operating system	Cross-platform: Windows, macOS, Linux
Type	Data acquisition, instrument control, test automation, analysis and signal processing, industrial control, embedded system design
License	Proprietary
Website	www.ni.com/labview

Laboratory Virtual Instrument Engineering Workbench (**LabVIEW**)[1]:3 is a system-design platform and development environment for a visual programming language from National Instruments.

The graphical language is named "G"; not to be confused with G-code. Originally released for the Apple Macintosh in 1986, LabVIEW is commonly used for data acquisition, instrument control, and industrial automationon a variety of operating systems (OSs), including Microsoft Windows, various versions of Unix, Linux, and macOS.

The latest versions of LabVIEW are LabVIEW 2018 and LabVIEW NXG 3.0, released in November 2018.[2]

Dataflow programming

The programming paradigm used in LabVIEW, sometimes called G, is based on data availability. If there is enough data available to a subVI or function, that subVI or function will execute. Execution flow is determined by the structure of a graphical block diagram (the LabVIEW-source code) on which the programmer connects different function-nodes by drawing wires. These wires propagate variables and any node can execute as soon as all its input data become available. Since this might be the case for multiple nodes simultaneously, LabVIEW can execute inherently in parallel.[3]:1–2 Multi-processing and multi-threading hardware is exploited automatically by the built-in scheduler, which multiplexes multiple OS threads over the nodes ready for execution.

Graphical programming

LabVIEW integrates the creation of user interfaces (termed front panels) into the development cycle. LabVIEW programs-subroutines are termed virtual instruments (VIs). Each VI has three components: a block diagram, a front panel, and a connector pane. The last is used to represent the VI in the block diagrams of other, calling VIs. The front panel is built using controls and indicators. Controls are inputs: they allow a user to supply information to the VI. Indicators are outputs: they indicate, or display, the results based on the inputs given to the VI. The back panel, which is a block diagram, contains the graphical source code. All of the objects placed on the front panel will appear on the back panel as terminals. The back panel also contains structures and functions which perform operations on controls and supply data to indicators. The structures and functions are found on the Functions palette and can be placed on the back panel. Collectively controls, indicators, structures, and functions are referred to as nodes. Nodes are connected to one another using wires, e.g., two controls and an indicator can be wired to the addition function so that the indicator displays the sum of the two controls. Thus a virtual instrument can be run as either a program, with the front panel serving as a user interface, or, when dropped as a node onto the block diagram, the front panel defines the inputs and outputs for the node through the connector pane. This implies each VI can be easily tested before being embedded as a subroutine into a larger program.

The graphical approach also allows nonprogrammers to build programs by dragging and dropping virtual representations of lab equipment with which they are already familiar. The LabVIEW programming environment, with the included examples and documentation, makes it simple to create small applications. This is a benefit on one side, but there is also a certain danger of underestimating the expertise needed for high-quality G programming. For complex algorithms or large-scale code, it is important that a programmer possess an extensive knowledge of the special LabVIEW syntax and the topology of its memory management. The most advanced LabVIEW development systems offer the ability to build stand-alone applications. Furthermore, it is possible to create distributed applications, which communicate by a client–server model, and are thus easier to implement due to the inherently parallel nature of G.

Widely-accepted design patterns

Applications in LabVIEW are usually designed using well-known architectures, known as design patterns. The most common design patterns for graphical LabVIEW applications are listed in the table below.

Common design patterns for LabVIEW applications

Design pattern	Purpose	Implementation details	Use cases	Limitations
Functional Global Variable	Exchange information without using global variables	A shift register of a while loop is used to store the data and the while loop runs only one iteration in a "non-reentrant" VI	• Exchange information with less wiring	• All owning VIs are kept in memory
State machine[4]	Controlled execution that depends on past events	Case structure inside a while loop pass an enumerated variable to a shift register, representing the next state; complex state machines can be designed using the Statechart module	• User interfaces • Complex logic • Communication protocols	• All possible states must be known in advance
Event-driven user interface	Lossless processing of user actions	GUI events are captured by an event structure queue, inside a while loop; the while loop is suspended by the event structure	• Graphical user interface	• Only one event structure in a loop

		and resumes only when the desired events are captured		
Master-slave[5]	Run independent processes simultaneously	Several parallel while loops, out of which one functions as the "master", controlling the "slave" loops	• Simple GUI for data acquisition and visualization	• Attention to and prevention of race conditions is required
Producer-consumer[6]	Asynchronous of multithreaded execution of loops	A master loop controls the execution of two slave loops, that communicate using notifiers, queues and semaphores; data-independent loops are automatically executed in separate threads	• Data sampling and visualization	• Order of execution is not obvious to control
Queued state machine with event-driven producer-consumer	Highly responsive user-interface for multithreaded applications	An event-driven user interface is placed inside the producer loop and a state machine is placed inside the consumer loop, communicating using queues between themselves and other parallel VIs	• Complex applications	

Benefits

Interfacing to devices

LabVIEW includes extensive support for interfacing to devices, instruments, camera, and other devices. Users interface to hardware by either writing direct bus commands (USB, GPIB, Serial) or using high-level, device-specific, drivers that provide native LabVIEW function nodes for controlling the device.

LabVIEW includes built-in support for NI hardware platforms such as CompactDAQ and CompactRIO, with a large number of device-specific blocks for such hardware, the *Measurement and Automation eXplorer* (MAX) and *Virtual Instrument Software Architecture* (VISA) toolsets.

National Instruments makes thousands of device drivers available for download on the NI Instrument Driver Network (IDNet).[7]

Code compiling

LabVIEW includes a compiler that produces native code for the CPU platform. This aids performance. The graphical code is translated into executable machine code by a compiler. The LabVIEW syntax is strictly enforced during the editing process and compiled into the executable machine code when requested to run or upon saving. In the latter case, the executable and the source code are merged into a single file. The executable runs with the help of the LabVIEW run-time engine, which contains some pre-compiled code to perform common tasks that are defined by the G language. The run-time engine reduces compiling time and provides a consistent interface to various operating systems, graphic systems, hardware components, etc. The run-time environment makes the code portable across platforms. Generally, LabVIEW code can be slower than equivalent compiled C code, although the differences often lie more with program optimization than inherent execution speed.[citation needed]

Large libraries

Many libraries with a large number of functions for data acquisition, signal generation, mathematics, statistics, signal conditioning, analysis, etc., along with numerous for functions such as integration, filters, and other specialized abilities usually associated with data capture from hardware sensors is enormous. In addition, LabVIEW includes a text-based programming component named MathScript with added functions for signal processing, analysis, and mathematics. MathScript can be integrated with graphical programming using *script nodes* and uses a syntax that is compatible generally with MATLAB.[8]

Parallel programming

LabVIEW is an inherently concurrent language, so it is very easy to program multiple tasks that are performed in parallel via multithreading. For example, this is done easily by drawing two or more parallel while loops and connecting them to two separate nodes. This is a great benefit for test system automation, where it is common practice to run processes like test sequencing, data recording, and hardware interfacing in parallel.

Ecosystem

Due to the longevity and popularity of the LabVIEW language, and the ability for users to extend its functions, a large ecosystem of third party add-ons has developed via contributions from the community. This ecosystem is available on the LabVIEW Tools Network, which is a marketplace for both free and paid LabVIEW add-ons.

User community

There is a low-cost LabVIEW Student Edition aimed at educational institutions for learning purposes. There is also an active community of LabVIEW users who communicate through several electronic mailing lists (email groups) and Internet forums.

Home Bundle Edition

National Instruments provides a low cost LabVIEW Home Bundle Edition.[9]

Criticism

LabVIEW is a proprietary product of National Instruments. Unlike common programming languages such as C or Fortran, LabVIEW is not managed or specified by a third party standards committee such as American National Standards Institute (ANSI), Institute of Electrical and Electronics Engineers (IEEE), International Organization for Standardization (ISO), etc. Many users have criticised it for its tendency to freeze or crash during simple tasks, often requiring the software to be shut down and restarted.

Slow

Very small applications still have to start the runtime environment which is a large and slow task. This tends to restrict LabVIEW to monolithic applications. Examples of this might be tiny programs to grab a single value from some hardware that can be used in a scripting language - the overheads of the runtime environment render this approach impractical with LabVIEW.[citation needed]

Non-textual

G language being non-textual, software tools such as versioning, side-by-side (or diff) comparison, and version code change tracking cannot be applied in the same manner as for textual programming languages. There are some additional tools to make comparison and merging of code with source code control (versioning) tools such as subversion, CVS and Perforce. [10][11][12]

No zoom function

There was no ability to zoom in to (or enlarge) a VI which will be hard to see on a large, high-resolution monitor, although this feature was released as of 2017.[13][14]

Release history

In 2005, starting with LabVIEW 8.0, major versions are released around the first week of August, to coincide with the annual National Instruments conference NI Week, and followed by a bug-fix release the following February.

In 2009, National Instruments began naming releases after the year in which they are released. A bug-fix is termed a Service Pack, for example, the 2009 service pack 1 was released in February 2010.

In 2017, National Instruments moved the annual conference to May and released LabVIEW 2017 along side a completely redesigned LabVIEW NXG 1.0 built on Windows Presentation Foundation (WPF).

Name-version	Build number	Date
LabVIEW project begins		April 1983
LabVIEW 1.0 (for Macintosh)	??	October 1986
LabVIEW 2.0	??	January 1990
LabVIEW 2.5 (first release for Sun & Windows)	??	August 1992
LabVIEW 3.0 (Multiplatform)	??	July 1993
LabVIEW 3.0.1 (first release for Windows NT)	??	1994
LabVIEW 3.1	??	1994
LabVIEW 3.1.1 (first release with "application builder" ability)	??	1995
LabVIEW 4.0	??	April 1996
LabVIEW 4.1	??	1997
LabVIEW 5.0	??	February 1998
LabVIEW RT (Real Time)	??	May 1999
LabVIEW 6.0 (6i)	6.0.0.4005	26 July 2000
LabVIEW 6.1	6.1.0.4004	12 April 2001
LabVIEW 7.0 (Express)	7.0.0.4000	April 2003
LabVIEW PDA module first released	??	May 2003
LabVIEW FPGA module first released	??	June 2003

LabVIEW 7.1	7.1.0.4000	2004
LabVIEW Embedded module first released	??	May 2005
LabVIEW 8.0	8.0.0.4005	September 2005
LabVIEW 8.20 (native Object Oriented Programming)	??	August 2006
LabVIEW 8.2.1	8.2.1.4002	21 February 2007
LabVIEW 8.5	8.5.0.4002	2007
LabVIEW 8.6	8.6.0.4001	24 July 2008
LabVIEW 8.6.1	8.6.0.4001	10 December 2008
LabVIEW 2009 (32 and 64-bit)	9.0.0.4022	4 August 2009
LabVIEW 2009 SP1	9.0.1.4011	8 January 2010
LabVIEW 2010	10.0.0.4032	4 August 2010
LabVIEW 2010 f2	10.0.0.4033	16 September 2010
LabVIEW 2010 SP1	10.0.1.4004	17 May 2011
LabVIEW for LEGO MINDSTORMS (2010 SP1 with some modules)		August 2011
LabVIEW 2011	11.0.0.4029	22 June 2011
LabVIEW 2011 SP1	11.0.1.4015	1 March 2012
LabVIEW 2012	12.0.0.4029	August 2012
LabVIEW 2012 SP1	12.0.1.4013	December 2012
LabVIEW 2013	13.0.0.4047	August 2013

LabVIEW 2013 SP1	13.0.1.4017	March 2014[15]
LabVIEW 2014		August 2014
LabVIEW 2014 SP1	14.0.1.4008	March 2015
LabVIEW 2015	15.0f2	August 2015
LabVIEW 2015 SP1	15.0.1f1	March 2016
LabVIEW 2016	16.0.0	August 2016
LabVIEW 2017	17.0f1	May 2017
LabVIEW 2017 SP1	17.0.1f1	Jan 2018 [16]
LabVIEW 2018	18.0	May 2018

Repositories and libraries

OpenG, as well as LAVA Code Repository (LAVAcr), serve as repositories for a wide range of Open Source LabVIEW applications and libraries. SourceForge has LabVIEW listed as one of the possible languages in which code can be written.

VI Package Manager has become the standard package manager for LabVIEW libraries. It is very similar in purpose to Ruby's RubyGems and Perl's CPAN, although it provides a graphical user interface similar to the Synaptic Package Manager. VI Package Manager provides access to a repository of the OpenG (and other) libraries for LabVIEW.

Tools exist to convert MathML into G code.[17]

Related software[edit]

National Instruments also offers a product named Measurement Studio, which offers many of the test, measurement, and control abilities of LabVIEW, as a set of classes for use with Microsoft Visual Studio. This allows developers to harness some of LabVIEW's strengths within the text-based .NET Framework. National Instruments also offers LabWindows/CVI as an alternative for ANSI C programmers.

When applications need sequencing, users often use LabVIEW with TestStand test management software, also from National Instruments.

The Ch interpreter is a C/C++ interpreter that can be embedded in LabVIEW for scripting.[18]

The TRIL Centre Ireland BioMobius platform and DSP Robotics' FlowStone DSP also use a form of graphical programming similar to LabVIEW, but are limited to the biomedical and robotics industries respectively.

LabVIEW has a direct node with modeFRONTIER, a multidisciplinary and multi-objective optimization and design environment, written to allow coupling to almost any computer-aided engineering tool. Both can be part of the same process workflow description and can be virtually driven by the optimization technologies available in modeFRONTIER.

See also

- 20-sim
- Comparison of numerical analysis software
- Dataflow programming
- DRAKON
- Fourth-generation programming language
- Graphical programming
- Graphical system design
- LabWindows/CVI
- Lego Mindstorms NXT, whose programming environment, NXT-G is based on LabVIEW, and can be programmed within LabVIEW.
- MATLAB/Simulink
- Virtual instrumentation
- CompactDAQ
- CompactRIO

References

1. ^ *Jeffrey., Travis, (2006). LabVIEW for everyone : graphical programming made easy and fun. Kring, Jim. (3rd ed.). Upper Saddle River, NJ: Prentice Hall. ISBN 0131856723. OCLC 67361308.*
2. ^ *"LabVIEW NXG: Version 3.0 Readme". Manuals. National Instruments.*
3. ^ *Bress, Thomas J. (2013). Effective LabVIEW Programming. [S.l.]: NTS Press. ISBN 1-934891-08-8.*
4. ^ *"Application Design Patterns: State Machines". National Instruments whitepapers. 8 September 2011. Archived from the original on 22 September 2017. Retrieved 21 September 2017.*
5. ^ *"Application Design Patterns: Master/Slave". National Instruments whitepapers. 7 October 2015. Archived from the original on 22 September 2017. Retrieved 21 September 2017.*
6. ^ *"Application Design Patterns: Producer/Consumer". National Instruments whitepapers. 24 August 2016. Archived from the original on 22 September 2017. Retrieved 21 September 2017.*
7. ^ *"3rd Party Instrument Drivers - National Instruments". www.ni.com. Archivedfrom the original on 2014-11-28.*
8. ^ *"LabVIEW MathScript RT Module". www.ni.com. Archived from the original on 2016-08-05.*
9. ^ *"LabVIEW Home Bundle for Windows - National Instruments". sine.ni.com. Archived from the original on 2016-07-04.*
10. ^ *"Archived copy". Archived from the original on 2016-10-28. Retrieved 2016-10-28.*
11. ^ *"Software Configuration Management and LabVIEW - National Instruments". www.ni.com. Archived from the original on 2016-10-29.*
12. ^ *"Configuring LabVIEW Source Code Control (SCC) for use with Team Foundation Server (TFS) - National Instruments". www.ni.com. Archived from the original on 2016-10-28.*
13. ^ *"Can I Zoom In or Out on a LabVIEW Diagram (for Wiring or Viewing Purposes)?". Archived from the original on March 5, 2016. Retrieved February 1, 2016.*
14. ^ *"Add a zoom function (yes, I said zoom. So sue me)". forums.ni.com. Archivedfrom the original on 2016-04-11. Retrieved 2016-03-31.*
15. ^ *"What's New in NI Developer Suite - National Instruments". www.ni.com. Archived from the original on 2014-03-31.*
16. ^ *"LabVIEW 2017 SP1 Patch Details - National Instruments". www.ni.com. Retrieved 2018-05-28.*
17. ^ *"Math Node - A new way to do math in LabVIEW". ni.com. 25 October 2010. Archived from the original on 25 February 2011.*
18. ^ *"Embedding a C/C++ Interpreter Ch into LabVIEW for Scripting". iel.ucdavis.edu. Archived from the original on 2011-05-15.*