LabVIEW™

DASYLab to LabVIEW Migration Guide

NATIONAL INSTRUMENTS™

June 1999 Edition
Part Number 322221A-01

Worldwide Technical Support and Product Information

www.natinst.com

National Instruments Corporate Headquarters

11500 North Mopac Expressway Austin, Texas 78759-3504 USA Tel: 512 794 0100

Worldwide Offices

Australia 03 9879 5166, Austria 0662 45 79 90 0, Belgium 02 757 00 20, Brazil 011 284 5011,
Canada (Ontario) 905 785 0085, Canada (Québec) 514 694 8521, China 0755 3904939, Denmark 45 76 26 00,
Finland 09 725 725 11, France 01 48 14 24 24, Germany 089 741 31 30, Hong Kong 2645 3186,
India 91805275406, Israel 03 6120092, Italy 02 413091, Japan 03 5472 2970, Korea 02 596 7456,
Mexico (D.F.) 5 280 7625, Mexico (Monterrey) 8 357 7695, Netherlands 0348 433466, Norway 32 27 73 00,
Singapore 2265886, Spain (Madrid) 91 640 0085, Spain (Barcelona) 93 582 0251, Sweden 08 587 895 00,
Switzerland 056 200 51 51, Taiwan 02 2377 1200, United Kingdom 01635 523545

For further support information, see the *Technical Support Resources* appendix. To comment on the documentation, send e-mail to techpubs@natinst.com.

Important Information

Warranty

The media on which you receive National Instruments software are warranted not to fail to execute programming instructions, due to defects in materials and workmanship, for a period of 90 days from date of shipment, as evidenced by receipts or other documentation. National Instruments will, at its option, repair or replace software media that do not execute programming instructions if National Instruments receives notice of such defects during the warranty period. National Instruments does not warrant that the operation of the software shall be uninterrupted or error free.

A Return Material Authorization (RMA) number must be obtained from the factory and clearly marked on the outside of the package before any equipment will be accepted for warranty work. National Instruments will pay the shipping costs of returning to the owner parts which are covered by warranty.

National Instruments believes that the information in this document is accurate. The document has been carefully reviewed for technical accuracy. In the event that technical or typographical errors exist, National Instruments reserves the right to make changes to subsequent editions of this document without prior notice to holders of this edition. The reader should consult National Instruments if errors are suspected. In no event shall National Instruments be liable for any damages arising out of or related to this document or the information contained in it.

EXCEPT AS SPECIFIED HEREIN, NATIONAL INSTRUMENTS MAKES NO WARRANTIES, EXPRESS OR IMPLIED, AND SPECIFICALLY DISCLAIMS ANY WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. CUSTOMER'S RIGHT TO RECOVER DAMAGES CAUSED BY FAULT OR NEGLIGENCE ON THE PART OF NATIONAL INSTRUMENTS SHALL BE LIMITED TO THE AMOUNT THERETOFORE PAID BY THE CUSTOMER. NATIONAL INSTRUMENTS WILL NOT BE LIABLE FOR DAMAGES RESULTING FROM LOSS OF DATA, PROFITS, USE OF PRODUCTS, OR INCIDENTAL OR CONSEQUENTIAL DAMAGES, EVEN IF ADVISED OF THE POSSIBILITY THEREOF. This limitation of the liability of National Instruments will apply regardless of the form of action, whether in contract or tort, including negligence. Any action against National Instruments must be brought within one year after the cause of action accrues. National Instruments shall not be liable for any delay in performance due to causes beyond its reasonable control. The warranty provided herein does not cover damages, defects, malfunctions, or service failures caused by owner's failure to follow the National Instruments installation, operation, or maintenance instructions; owner's modification of the product; owner's abuse, misuse, or negligent acts; and power failure or surges, fire, flood, accident, actions of third parties, or other events outside reasonable control.

Copyright

Trademarks

HiQ™, LabVIEW™, `natinst.com`™, National Instruments™, and NI-DAQ™ are trademarks of National Instruments Corporation.

Product and company names mentioned herein are trademarks or trade names of their respective companies.

WARNING REGARDING MEDICAL AND CLINICAL USE OF NATIONAL INSTRUMENTS PRODUCTS

National Instruments products are not designed with components and testing for a level of reliability suitable for use in or in connection with surgical implants or as critical components in any life support systems whose failure to perform can reasonably be expected to cause significant injury to a human. Applications of National Instruments products involving medical or clinical treatment can create a potential for death or bodily injury caused by product failure, or by errors on the part of the user or application designer. Because each end-user system is customized and differs from National Instruments testing platforms and because a user or application designer may use National Instruments products in combination with other products in a manner not evaluated or contemplated by National Instruments, the user or application designer is ultimately responsible for verifying and validating the suitability of National Instruments products whenever National Instruments products are incorporated in a system or application, including, without limitation, the appropriate design, process and safety level of such system or application.

Contents

About This Manual

Chapter 1
Introduction to DASYLab and LabVIEW

Chapter 2
LabVIEW Programming

Chapter 3
Migrating from DASYLab to LabVIEW

Appendix A
DASYLab File I/O Functions

Appendix B
Technical Support Resources

Glossary

Figures

Table

Activity

About This Manual

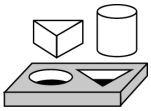
This manual provides an introduction to LabVIEW and G programming for users familiar with DASY*Lab*. It also contains information on how to use the tools provided with the DASY*Lab* to LabVIEW Migration Toolkit that help you move your existing DASY*Lab* applications to LabVIEW.

This manual assumes you are familiar with the Windows NT/98/95 operating system.

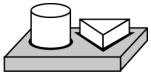
Conventions Used in This Manual

The following conventions are used in this manual:

» The **»** symbol leads you through nested menu items and dialog box options to a final action. The sequence **File»Page Setup»Options** directs you to pull down the **File** menu, select the **Page Setup** item, and select **Options** from the last dialog box.



This icon denotes the beginning of an activity.



This icon denotes the end of an activity.

This icon denotes a tip, which alerts you to advisory information.

This icon denotes a note, which alerts you to important information.

bold Bold text denotes items that you must select or click on in the software, such as menu items and dialog box options. Bold text also denotes parameter names.

italic Italic text denotes variables, emphasis, a cross reference, or an introduction to a key concept. This font also denotes text that is a placeholder for a word or value that you must supply.

`monospace` Text in this font denotes text or characters that you should enter from the keyboard, sections of code, programming examples, and syntax examples. This font is also used for the proper names of disk drives, paths, directories,

programs, subprograms, subroutines, device names, functions, operations, variables, filenames and extensions, and code excerpts.

Tutorial Tutorial references provide pointers to related activities in other documents or in the *LabVIEW Online Reference*.

Related Documentation

The following documents contain information you might find useful as you read this manual:

- *LabVIEW QuickStart Guide*
- *G Programming Quick Reference Card*
- *LabVIEW User Manual*
- *LabVIEW Online Reference*, available by selecting **Help»Online Reference**.
- *G Programming Reference Manual*
- *LabVIEW Data Acquisition Basics Manual*
- *LabVIEW 5.1 Addendum*

1

Introduction to DASYLab and LabVIEW

The DASY*Lab* to LabVIEW Migration Toolkit provides tools and information to help DASY*Lab* users to become familiar with and use LabVIEW to extend the functionality and range of their development tools. The migration tools included with this documentation are designed to help you transition from DASY*Lab* to LabVIEW with existing applications and develop new applications in LabVIEW.

DASY*Lab* is an easy-to-use development environment for creating applications that perform measurements, analyze acquired data, and generate reports. DASY*Lab* applications are built in a graphical environment by configuring different program modules with dialog windows and arranging them by connecting them with wires that pass data from one module to the next. DASY*Lab* includes support for a range of data acquisition devices, GPIB interfaces, and other hardware.

LabVIEW is a graphical programming environment that uses a graphical diagram similar to DASY*Lab* for developing measurement and automation systems. LabVIEW is a complete programming environment that enables you to create more extensive, powerful, and flexible test systems.

Installation

To become familiar with LabVIEW and learn how to convert an existing application or create a new application, it is best to have DASY*Lab* and LabVIEW installed. Follow the directions provided with each application for installation.

To install the DASY*Lab* to LabVIEW Migration Toolkit, run the `setup.exe` program on the installation CD and follow the directions in the installer.

DASYLab to LabVIEW Migration Toolkit

The migration tools consist of the following three components:

- *DASYLab to LabVIEW Migration Guide*
- DASY*Lab* File I/O Functions for LabVIEW
- Example files, located in the `LabVIEW\Examples\DASYLab` folder

This manual describes the differences between DASY*Lab* and LabVIEW and introduces the LabVIEW environment and programming methodology in terms familiar to DASY*Lab* users. This information is not intended to completely document LabVIEW and should be used in conjunction with the LabVIEW documentation. This manual also introduces the terminology used in LabVIEW and describes it in terms used in DASY*Lab.*

You can use the DASY*Lab* File I/O functions (VIs) to access your existing DASY*Lab* data files (`.ddf`) from within a LabVIEW application. These VIs support the default DASY*Lab* data file format and streaming data files written directly from a data acquisition driver in DASY*Lab.* You also can exchange data between DASY*Lab* and LabVIEW data by reading and writing it in pure binary format without additional timing and channel information. For more information on the DASYLab File I/O functions, see Appendix A, *DASYLab File I/O Functions*.

The example files that accompany the DASY*Lab* to LabVIEW Migration Toolkit provide you with several different approaches to converting DASY*Lab* experiments to LabVIEW VIs. This examples directory includes LabVIEW and DASY*Lab* files.

Comparison of DASYLab and LabVIEW

DASY*Lab* and LabVIEW are development tools designed to build measurement applications that use a graphical diagram to define the execution of the application, rather than a text-based syntax like Visual Basic or C. Although DASY*Lab* and LabVIEW look similar, there are some underlying differences in the design and execution of applications created in these two environments.

LabVIEW data wires contain simple data types, like Booleans, strings, floating-point numbers, and integers, as well as arrays and structures of these types that represent any type of information imaginable. DASY*Lab* processes data in defined data blocks, which are usually the result of a measurement or derived data blocks of measurements.

DASY*Lab* data wires pass complete data blocks, which are complex data structures that contain the measurement data along with additional information such as scan rate and channel information. In LabVIEW, you handle this additional information in the design of the program.

LabVIEW contains a number of different graphical programming structures—loops, sequences, and case structures—that make it possible for you to direct program execution and respond to different conditions in an application. In DASY*Lab*, such program control is only partially possible using indirect methods like the trigger and relay modules. Although LabVIEW does not provide as much automatic data handling of measurement information as DASY*Lab*, it contains a more flexible and powerful programming interface that allows you to define advanced and complex programs quickly.

The LabVIEW graphical programming language, called *G*, provides all the tools familiar in most traditional text-based programming languages. With these tools, developers can define a wider range of process, program structures, and functions than is possible in the DASY*Lab* flowchart.

Tutorial Complete the activities Chapters 1 through 3 of the *LabVIEW QuickStart Guide* to learn and practice basic programming features in LabVIEW. The *LabVIEW QuickStart Guide* contains specific information and activities you can use to learn LabVIEW.

Virtual Instruments

An application in LabVIEW is called a *virtual instrument*, or *VI*. A VI is analogous to an experiment in DASY*Lab*. Unlike DASY*Lab* experiments, LabVIEW VIs do not have global settings to set sample rate, block size, and so on. All hardware selection or control in LabVIEW is done directly in the function blocks in the block diagram.

Also, unlike the DASY*Lab* experiment and its single flowchart, each VI consists of two components—the *front panel*, which is analogous to the user interface of the application, and the *block diagram*, which is analogous to the code. The two parts are stored together in one file for each VI. These two parts are created and edited in the two main windows of the LabVIEW environment.

The front panel of the VI is used to create and configure the user interface and its objects, such as buttons, sliders, and graphs of an application. Creating the front panel is the first step when building a new application in LabVIEW.

The block diagram is the graphical source code of the application. It is similar to the flowchart in DASY*Lab*, except that you do not have specific function blocks in the block diagram to create and operate each user interface object. A block diagram consists of different function blocks that define the operation of the application.

SubVIs

When you use a VI inside another block diagram, it is called a *subVI*, similar to the black box module in DASY*Lab*, or a subroutine in other programming languages. For example, a VI that measures the temperature from a sensor connected to a data acquisition card can be converted into a subVI and used in another VI as part of a larger monitoring or test system, as shown in the following figure.

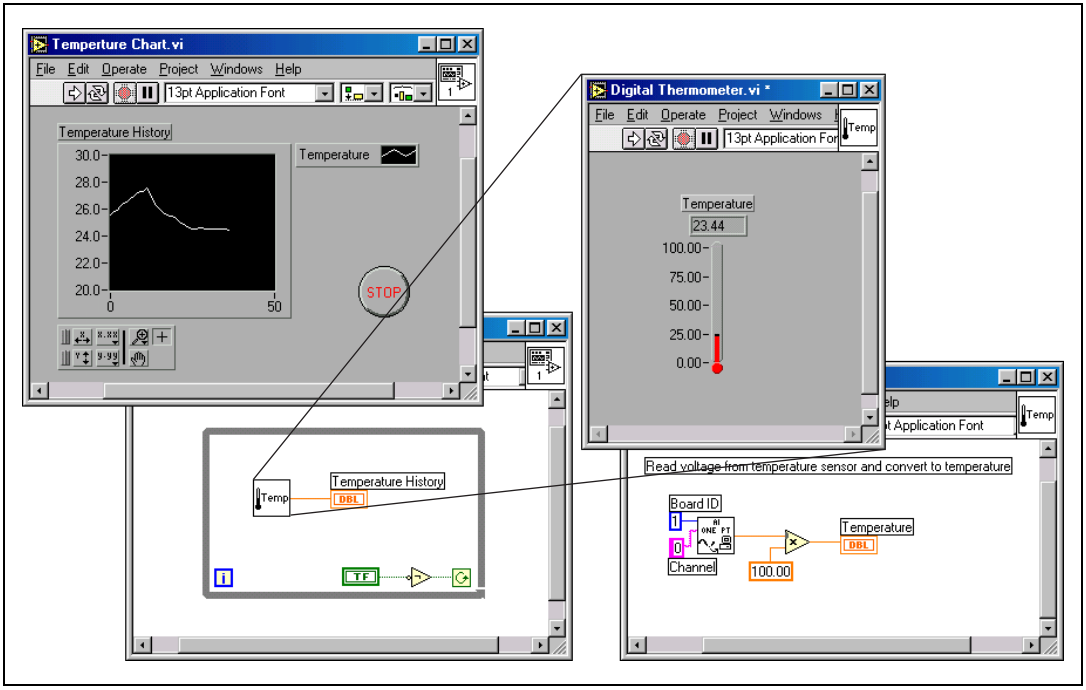


Figure 1-1. Reuse of Digital Thermometer VI as a SubVI in Temperature Chart VI

Tip Use subVIs in your applications to simplify debugging and keep the block diagram easy to read. Any group of functions that performs a specific, defined task should be combined into a subVI.

Another use for subVIs is displaying multiple front panels in an application. In DASY*Lab*, you can create and save multiple layouts with one flowchart and display the different layouts at runtime. In LabVIEW, because each VI or subVI has one front panel (which can contain several different elements), you use different subVIs to display more than one user interface. The front panel of the main application (top-level VI) is used to navigate through the application, with subVI front panels as the user interfaces for different parts of the application.

One benefit of the subVI architecture is the ability to execute, test, and debug each subVI individually. Because each subVI is an independent VI, you can set all the inputs on the front panel and run it as an independent program to examine the output (front panel) for the proper results.

LabVIEW includes a large group of subVIs for hardware I/O, analysis, and more. LabVIEW also contains a large selection of built-in basic function modules such as arithmetic, string and array manipulation, comparison, type conversion, and so on. You use these functions and subVIs to build your application.

Tip Basic functions are indicated by a yellow icon background.

Tutorial Learn more about subVIs by completing Activities 3, 4, and 6 in the *LabVIEW Online Reference*, available by selecting **Help»Online Reference** and selecting **Learning LabVIEW with Activities** from the contents page.

Front Panel

You use the front panel to develop and customize the user interface of an application. The available objects are found in the **Controls** palette, available by selecting **Windows»Show Controls Palette**, or right-clicking, or *popping up*, on the background of the front panel.

The **Controls** palette is an example of a floating palette in LabVIEW. You can keep visible any subpalette by selecting the thumbtack in the top left corner of the palette. This provides quicker repeated access to the controls or functions in the selected palette.

The user interface objects are organized into groups, or *subpalettes*, such as numeric controls and indicators, Boolean controls and indicators, string and table controls and indicators, lists, and so on. To place objects on the front panel, select an object in the **Controls** palette with your mouse and drag the object to the front panel.

Tools

When you develop an application in the flowchart in DASY*Lab*, you use one tool for all operations, such as configuring or moving a module or creating a wire. The mouse cursor automatically changes shape depending on its location or the operation being performed.

In LabVIEW, there are several tasks you can perform with the mouse cursor, and you select from different tools to perform these operations. To select these different tools display the **Tools** palette by selecting **Windows»Show Tools Palette**, which is shown in the following figure.

You can rotate through the most commonly used tools by using the Tab key. You also can use the space bar to switch between the Operating and Positioning tools only.

For more information on the **Tools** palette, see Chapter 2, *Editing VIs*, of the *G Programming Reference Manual*.

Controls and Indicators

Each object on the front panel can serve either as an input or as an output. Inputs, called *controls* in LabVIEW, are used to pass a value through the user interface into the block diagram. Outputs, called *indicators*, are used to display values coming from the block diagram on the user interface. Each user interface object has a default direction, either control or

indicator, which you can change by popping up on the object and selecting **Change to Indicator** or **Change to Control**.

For each object you create on the front panel, a corresponding icon called a *terminal* is created on the block diagram. The terminals are used to connect, or wire, the front panel objects to the functions blocks in the block diagram.

Tip On the block diagram, control terminals are displayed with a thick border and indicator terminals have thin borders.

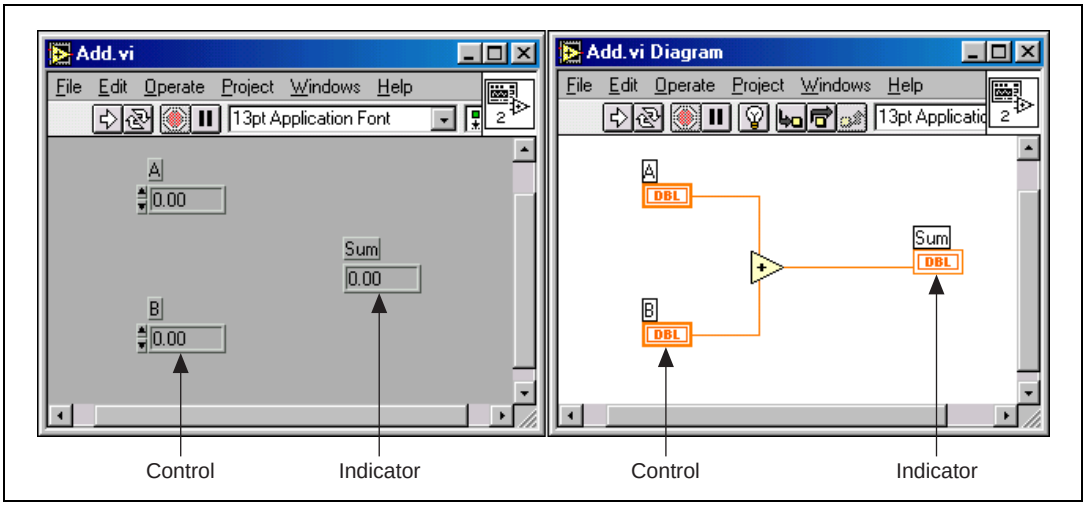


Figure 1-2. Controls and Indicators on the Front Panel and Block Diagram

Each control is given a name when you place it on the front panel. You can modify the name immediately after creating a new control, or you can pop up on the object and select **Show»Label** to modify the name at a later point. The label is shown next to the corresponding terminal on both the front panel and the block diagram.

See the *LabVIEW QuickStart Guide* and Chapter 2, *Editing VIs*, of the *G Programming Reference Manual* for detailed descriptions of the front panel editor and user interface objects.

Block Diagram

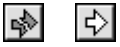
The LabVIEW block diagram consists of subVIs and functions that define the operation of the VI. Data wires are used to pass data from one function to the next and to define the order of operation. Terminals for each of the user interface controls are used to pass data to and from the front panel. LabVIEW also includes other programming structures that extend the flexibility of the block diagram. These include loops, case and sequence structures, nodes to access DLLs, and more.

The following figure shows the available basic functions and subVIs in the **Functions** palette, available by selecting **Windows»Show Functions Palette**, or by popping up on the background of the block diagram. The functions are organized into subpalettes such as numeric, Boolean, string, file I/O, and so on. You also can customize or configure the **Functions** palette and add your own functions.

For more information about the **Functions** palette, see Chapter 17, *Introduction to the Block Diagram*, in the *G Programming Reference Manual*.

Running a VI

The execution of a LabVIEW VI is different from the execution of a DASY*Lab* experiment. In DASY*Lab*, the experiment runs in a continuous mode when you start the experiment and continues to run until you stop it. In LabVIEW, VIs run only once. Continuous run or repetition of parts of a block diagram are done using loops.



As you build a VI, LabVIEW continuously checks the syntax of the program to determine if there are any errors. If an error exists, a broken Run button is shown in the toolbar in place of the Run button. Pressing the broken Run button displays a list of errors. Double-clicking on an error message shows the location of the error. When everything is correct, the Run button is shown.

Tip For debugging purposes, you can use the Continuous Run button to execute a simple block diagram repeatedly. You should be careful using this mode, because you can get into situations where it is difficult to stop the VI.

You can stop the VI by including a stop or quit button on your user interface. The stop button on the toolbar also stops the VI immediately, which can leave hardware, files, and communication in an unknown state, and is therefore not recommended.

LabVIEW Terminology

Many of the concepts, tools, and methodologies in DASY*Lab* and LabVIEW are similar, although each environment uses different terminology. Table 1-1 lists commonly used DASY*Lab* and LabVIEW terms with each environment and a short description or clarification.

Table 1-1. DASY*Lab* and LabVIEW Terminology

DASY*Lab*	LabVIEW	Description
Experiment	VI	Term used for a basic program in each environment. An experiment contains global settings, such as sample rate and block size, which do not exist for VIs.
Layout	Front Panel	The front panel corresponds to the user interface of the program.
Flowchart Worksheet	Block Diagram	The block diagram defines the operation of the program.
Module	Function	A function in LabVIEW is a basic block diagram function block that is defined as part of the LabVIEW environment. You cannot access the code of a function.
Black Box Module	SubVI	A subVI is a function block that is stored as a separate VI with its own front panel and block diagram. You can access the front panel and block diagram of a subVI by double-clicking on the icon.

Table 1-1. DASY*Lab* and LabVIEW Terminology (Continued)

DASY*Lab*	LabVIEW	Description
—	Control	A control is a front panel object used to pass data to the block diagram of the VI.
—	Indicator	An indicator is a front panel object used to return or display data from the block diagram of the VI.
—	Terminal	A terminal is an icon on the block diagram that represents a front panel object; the inputs or outputs of a block diagram function block (subVI or function).
Recorder	Waveform Chart	Chart display used to record slow signals. New data points are continuously appended to the end of the existing plots. Charts are used for continuous updates.
Y/t Chart	Waveform Graph	Graph display for signals with linear x scaling. Used to display time signals and other signals with a constant X increment. Graphs are used to display data at the end of an acquisition.
X/Y Chart	XY Graph	Graph display used to display signal with separate X and Y components. Graphs are used to display data at the end of an acquisition.
—	Icon	The image that represents a function or subVI on the block diagram.
—	Node	Nodes are the execution elements of a block diagram. Data is passed through wires to and from nodes.
Wire	Wire	A representation of data being passed from one node to another in the LabVIEW block diagram. Each wire has a defined data type and dimension.
Data Block	Data Type (string, integer, float, Boolean, array, cluster, and so on)	All data items used in LabVIEW have a defined data type, such as string, Boolean, integer, and so on. The data type is determined by the source of the data (wire) or by a conversion function. Arrays and clusters are more complex data types.
—	While Loop	A while loop is a programming structure that is repeated while a Boolean value is true; analogous to while loop in other programming languages.

Table 1-1. DASY*Lab* and LabVIEW Terminology (Continued)

DASY*Lab*	LabVIEW	Description
—	For Loop	A for loop is a programming structure that is repeated a given number of times or once for each element of an array; analogous to for loop in other programming languages.
—	Case Structure	Programming structure that allows selection of a specific section of code based on a Boolean or integer value; analogous to If-Then-Else or Switch statements in other programming languages.
—	Sequence Structure	Programming structure that allows you to define the exact execution order of a section of a block diagram.
Global Variable Global String	Local Variable	A LabVIEW block diagram terminal that provides read or write access by name to front panel objects.
—	Global Variable	A block diagram terminal that provides access to a global variable created in a global variable file (special VI); multiple VIs can access the same global variables. Global variables in LabVIEW are different from DASY*Lab* global variables.
DASY*Lab* Extension	CIN (Code Interface Node) DLL (Dynamic Link Library)	Function block written in C specifically for LabVIEW to extend the LabVIEW functionality. Generic DLL that can be accessed from the block diagram to extend the LabVIEW functionality.
Routing/ Autorouter		There is no autorouting feature in LabVIEW. While creating a wire, you can click on the block diagram to tack down a wire in a specific location.
Black Box Import and Export Modules	SubVI Connector Pane	The connector pane of a subVI takes the place of the Black Box Import and Export Modules. As you create a subVI, you use the connector pane to define the inputs and outputs of the subVI.
Experiment» Hardware Setup and **Experiment Setup**	Data Acquisition VI Parameters	The settings made in the Hardware and Experiment Setup dialog boxes in DASY*Lab* are set directly as parameters of the data acquisition VIs in LabVIEW.

Table 1-1. DASY*Lab* and LabVIEW Terminology (Continued)

DASY*Lab*	LabVIEW	Description
Password Protection	VI Info	Individual VIs and subVIs can be password protected in the VI Info dialog box by selecting **Windows»Show VI Info**.
Screen Lock	VI Setup	You use the options in the VI Setup dialog box, available by popping up on the icon in the taskbar and selecting **VI Setup**, to disable elements of the LabVIEW environment while the application is running.
Window Arrangement	Front Panel, Multiple VIs, Attribute Node	The front panel, as well as multiple VIs, are used to create different arrangements of user controls and indicators. You can use the attribute node to change different settings of a control at runtime such as color, size, and position.
Display Windows, Tree Window	VI Hierarchy	There is no direct corollary in LabVIEW. The VI Hierarchy window lists all the VIs and subVIs used and their calling relationship.
Versions: Lite, Basic, Full	Versions: Base, Full, Professional	There are three versions of LabVIEW (Base, Full, Professional) with increasing sets of VIs and other development tools such as source code control, a standalone executable builder, and so on.
Module Groups	**Functions** Palette	Functions and VIs in LabVIEW are grouped by class (numeric, string, file I/O, and so on) in the **Functions** palette of the block diagram.
File Formats	VI File Format	LabVIEW uses one file format for VIs and subVIs. You can exchange VIs among different platforms that LabVIEW supports (Windows NT/98/95, Macintosh, UNIX, Concurrent PowerMAX, and Linux). Starting with LabVIEW 5.1, you can save VIs in file formats for previous versions of LabVIEW.
DASY*Lab* Net (Client, Server)	VI Server	VI Server can be used to control a remote copy of LabVIEW. With VI Server, you can load VIs, start and stop them and perform other environment operations.
Net Import and Net Export Modules	DataSocket	The DataSocket VIs in LabVIEW are used to share data among LabVIEW applications on different computers. You can also use the low level TCP/IP functions to perform more customized network communication among applications.

2

LabVIEW Programming

LabVIEW programs, like applications developed in DASY*Lab*, are based on a graphical diagram. However, there are some differences in the graphical methodology between these two environments. In general, most of the flowchart concepts that exist in DASY*Lab* also apply in LabVIEW.

LabVIEW goes beyond DASY*Lab* by integrating more programming concepts, such as programming structures and data types. Generally speaking, LabVIEW combines the ease of use of the DASY*Lab* graphical environment with the flexibility of text-based programming languages like Visual Basic or C.

Unlike text-based applications that execute line by line from top to bottom, LabVIEW uses a principle called *dataflow* to determine the execution order of the functions in the block diagram.

Because of the graphical nature of the LabVIEW program code, you can develop parallel sections in a block diagram that are not linked to one another and where no data dependency exists among function blocks. In this case, the execution system determines an order of execution for these unrelated sections of code.

Data Types

Like DASY*Lab*, LabVIEW passes data from one function block to another using data wires. In DASY*Lab*, all data passed between modules is stored in data blocks that contain numeric or Boolean data. The data inside each data block is stored in an array of single-precision floating-point numbers.

In addition to the data, each data block in DASY*Lab* contains and passes a collection of other information such as a start time and sample rate, which is used by each module in DASY*Lab* to process or display the data.

LabVIEW stores and passes all data types with a defined data type, such as string, Boolean, integer, and more complex types, such as arrays and clusters. All data types have an identifying color used for the wires and terminals of that specific data type. Each terminal also has a unique symbol that identifies the data type and representation. Each data wire contains

only the information given to it by its source. The data wire does not contain additional information as in DASY*Lab.*

The LabVIEW programming model of using defined data types follows text-based programming languages, although LabVIEW handles all memory tasks for you, greatly simplifying program development. This flexibility in choosing specific data types and creating data structures enables you to optimize memory usage and create more advanced program structures.

Data Type Conversion

LabVIEW automatically converts between any of the numeric data types, integers, and floating-point numbers to match the data type of an input terminal. When an automatic conversion occurs, a small gray dot appears at the location of the data terminal. Other conversions are not done automatically.

You also can manually convert any numeric data types using the conversion functions in the **Functions»Numeric»Conversion** palette. Additional conversion functions for strings and Booleans are available and can be found in the corresponding palettes.

Polymorphism

Many of the LabVIEW functions are *polymorphic*, which means they can accept inputs of different data types (strings, numbers, scalars, and arrays). For example, the Add function can add two scalar values, two arrays, or an array with a scalar value. The comparison functions also are polymorphic, and support strings, scalars, and arrays.

Simple Data Types

Simple data types in LabVIEW include scalar values of strings, Booleans, integers, floating-point numbers, and so on. Integers have different representations of 8-, 16-, and 32-bit integers as well as signed and unsigned. Floating-point numbers have representations of single (4 byte), double (8 byte) and extended (16 byte) precision.

The data type of a front panel object is determined by the type of object. For example, knobs and slides are numeric, buttons are Boolean, and text boxes are strings. Objects that represent numeric data can have any representation of integer or floating-point numbers. You can change the representation of an object by popping up on the object and selecting **Representation**.

Arrays

An *array* is a data type that contains multiple items of the same data type. For example, a waveform acquired from a data acquisition card is stored in an array of numbers. Arrays in LabVIEW can have any number of dimensions. You can have an array of any data type, except another array, including reference numbers (refnums) and clusters. If you need to create an array of arrays, add an extra dimension to the original array.

In comparison, DASY*Lab* stores all its data, even single values, in 1D arrays, which are contained in the data blocks. In LabVIEW, you define whether a data item is a scalar (a single value of a data type) or an array.

There is a wide range of functions available in LabVIEW to manipulate arrays, including creating arrays, reshaping arrays, sorting arrays, and more. There are also functions to easily extract individual elements, as well as whole columns out of a multi-dimensional array. On the block diagram, arrays are indicated by thicker wires, with increasing thickness for increasing number of dimensions.

Clusters

Clusters are a combination of multiple data items of different types. You can combine any data types in a cluster, including scalars, strings, arrays, and other clusters.

Clusters are used to create data structures, which represent a logical grouping of information. You can pass a cluster with all its data items on one wire, which greatly simplifies the block diagram. LabVIEW includes functions to easily combine (bundle) different data items into one cluster or to extract (unbundle) data items from a cluster.

Clusters are used frequently throughout LabVIEW, and certain default clusters have been defined. The most commonly used cluster is the error cluster, which is used to pass status and error information between functions and subVIs.

Graphs

Another application of clusters is to bundle information together before passing it to the waveform or XY graphs. The waveform graph can accept formatted waveform information that includes an initial X and incremental X value. These two scalar values in addition to a 1D or 2D array are used to plot the waveform data with the X axis scaled accordingly, as shown in Figure 2-1.

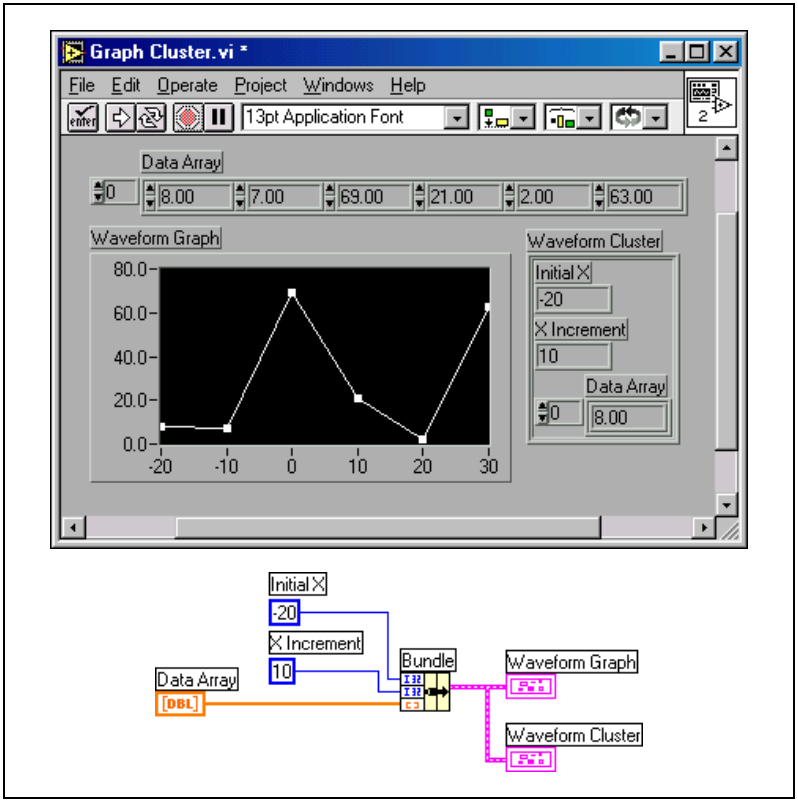


Figure 2-1. Bundling Data to a Waveform Cluster and Waveform Graph

Enumerations

An *enumeration* is an unsigned integer data type in which specific integer values are associated with descriptive strings. Enumerated types are useful for creating selection lists with a predefined list of items. For example, you can create a list of different tests that can be performed. Each test is an item in the list and can be associated with a corresponding program structure in the block diagram, as shown in Figure 2-2.

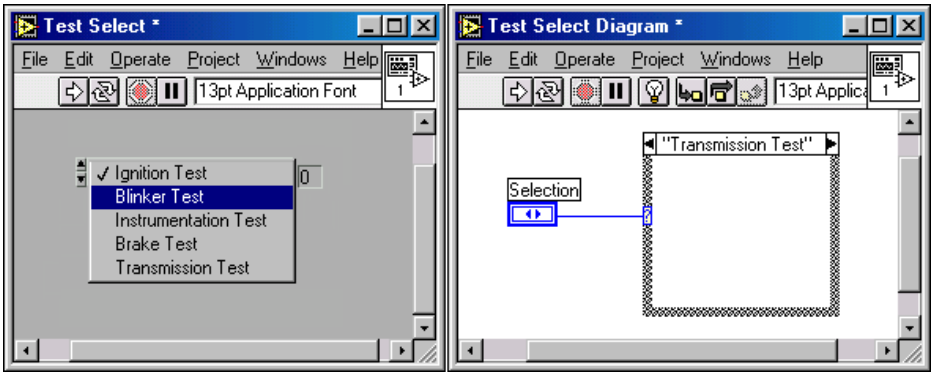


Figure 2-2. Example of an Enumerated Data Type

Refer to Chapter 14, *Array and Cluster Controls and Indicators*, of the *G Programming Reference Manual* for more information on enumerated types.

Paths and Reference Numbers

The path data type is used for the path of a filename or directory. This data type is similar to a string but is used only with file I/O VIs. You can use the path control and indicator to enter or display a pathname on the front panel or to create input and output terminals on a subVI.

The reference number data type, or *refnum*, is analogous to a pointer or a handle in traditional programming languages. A refnum refers to an object in memory that has no common representation. For example, once you open a file for reading or writing, it is assigned a reference number. Other file I/O functions can then use this reference number to access the same file.

Programming Structures

DASY*Lab* is based on a model of continuous execution, where the flowchart is executed repeatedly until the user stops the application. In LabVIEW, the block diagram is executed only once and stops when all functions have been executed.

LabVIEW supports several different programming structures that provide the ability to repeat a section of the block diagram a number of times, branch into different sections of the block diagram based on a variable, and order the sequence of the block diagram.

While Loop

A *While Loop* is a structure that repeats a section of code when a specific condition is True. It is comparable to a Do While loop or a Repeat-Until loop in text-based programming languages.

The While Loop is a resizable box you use to execute code until a Boolean value passed to the *conditional terminal* is False. The VI checks the conditional terminal at the end of each iteration. Therefore, the While Loop always executes at least once. The *iteration terminal* is an output numeric terminal that outputs the number of times the loop has executed. The iteration count always starts at zero. If the loop runs once, the iteration terminal outputs 0. The following figure shows a While Loop and indicates the iteration and conditional terminals.

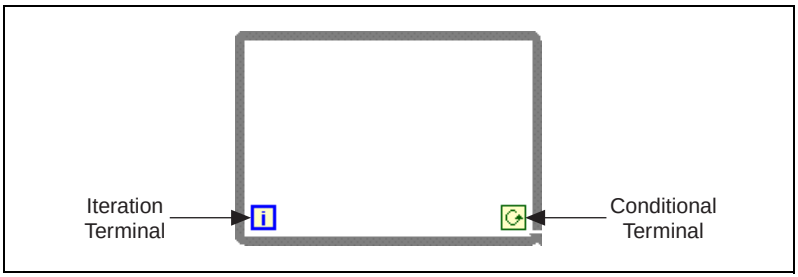


Figure 2-3. While Loop with Terminals

In Figure 2-4, the VI calculates a random value in every iteration of the loop and multiplies it by the scaling value, which is passed into the loop from a control on the front panel. The loop continues running as long as the scaled random value is greater than 0.05 and displays the current iteration value on the front panel. Once the value is less than the threshold, the loop stops and outputs the final scaled value, which is displayed on the front panel.

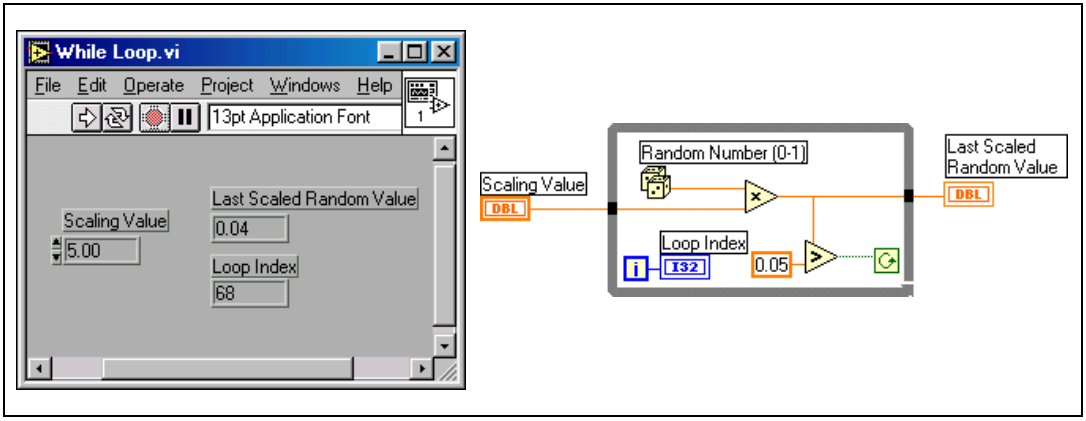


Figure 2-4. Example of While Loop Used in an Application

Tutorial To learn more about using While Loops, complete Activity 8 in the *LabVIEW Online Reference*, available by selecting **Help»Online Reference** and selecting **Learning LabVIEW with Activities** from the contents page.

For Loop

The *For Loop* repeats a section of the block diagram a defined number of times. The For Loop includes two terminals. The *count terminal*, which you connect with a number value, determines how often the loop is executed. The iteration terminal returns the current iteration value of the loop, similar to the While Loop. The following figure shows a For Loop and indicates the count and iteration terminals.

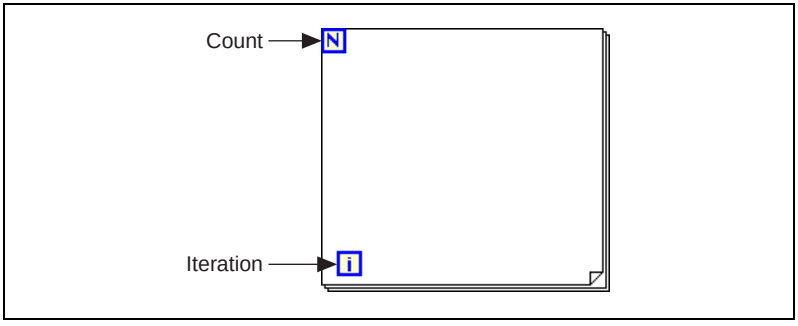


Figure 2-5. For Loop with Terminals

Indexing

You can use the For Loop to build an array of values by calculating each element of the array inside the loop and connecting a wire from these values to a function or other terminal outside the loop. This is possible because the tunnel that passes the values from inside the loop to the outside accumulates each value and passes all of them as an array at the completion of the For Loop. This behavior of the tunnel is called *indexing*. By default, indexing is enabled for every tunnel created on a For Loop. You can disable this behavior by popping up on the tunnel and selecting **Disable Indexing**. Once disabled, such a tunnel returns the last calculated value from the loop.

Tutorial To learn more about using For Loops, complete Activities 13, 18, and 19 in the *LabVIEW Online Reference,* available by selecting **Help»Online Reference** and selecting **Learning LabVIEW with Activities** from the contents page.

Case Structure

You use *Case structures* to run different sections of code depending on the value of a specific variable. Case structures are analogous to Case or Switch statements in traditional programming languages.

Case structures consist of one visible frame and one or more hidden frames in memory. The frame includes a selector terminal that determines which frame is executed. You can connect either a Boolean or integer value to the selector. With a Boolean selector, the Case structure has a True and a False case. With an integer selector, the Case structure can have several different cases. The top of the frame displays which frame is currently displayed. The following figure shows a Case structure.

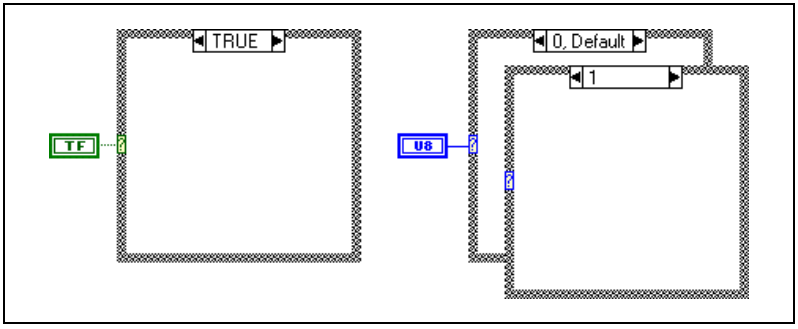


Figure 2-6. Case Structures

Tutorial To learn more about using Case structures, complete Activity 14 in the *LabVIEW Online Reference*, available by selecting **Help»Online Reference** and selecting **Learning LabVIEW with Activities** from the contents page.

Sequence Structure

Sequence structures are similar to Case structures in that they have multiple frames, with only one frame visible at a time. However, the Sequence structure executes all its frames in order. It is used to define the order in which different processes with no data dependency between them are executed. Sequence structures also can be used to arrange logical steps of sequences in a small space on the block diagram.

When the Sequence structure is created it has only one frame. You use the pop-up menu of the Sequence structure to add or delete individual frames. Values wired out of the sequence are not available outside the sequence until the whole sequence is complete, even if they are wired out of an earlier frame of the sequence. The following figure shows a Sequence structure.

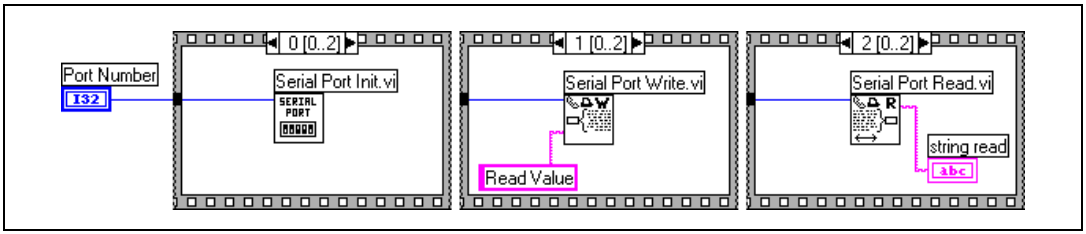


Figure 2-7. Sequence Structure with All Frames Shown

Tutorial To learn more about using Sequence structures, complete Activity 15 in the *LabVIEW Online Reference*, available by selecting **Help»Online Reference** and selecting **Learning LabVIEW with Activities** from the contents page.

Local and Global Variables

LabVIEW uses local and global variables to access data without passing it by wire. Global variables in LabVIEW and DASY*Lab* are not the same thing. A LabVIEW local variable corresponds in usage to the DASY*Lab* global variable.

Local Variables

Local variables in LabVIEW serve many of the same purposes as global variables in DASY*Lab*. They allow you to access a value from any place on your block diagram referenced only by name.

A local variable is a copy of a terminal for a front panel object. When you create an object on the front panel, a corresponding terminal is created on the block diagram. If the object is a control, you can read the terminal. If it is an indicator, you can write a value to it.

For more information about local variables, see Chapter 23, *Global and Local Variables*, of the *G Programming Reference Manual*.

Global Variables

The LabVIEW environment supports multiple VIs running at the same time. With *global variables*, you can set up named front panel objects that can be accessed from multiple VIs at the same time. The global variable exists in a separate front panel that contains no block diagram and is used only to store the data that is accessed from other VIs.

For more information about global variables, see Chapter 23, *Global and Local Variables*, of the *G Programming Reference Manual*.

3

Migrating from DASYLab to LabVIEW

This chapter outlines and describes a number of specific methods for converting a DASY*Lab* diagram into LabVIEW.

Basics of Converting a Program

Before converting existing DASY*Lab* applications to LabVIEW, you should be familiar with the basics of LabVIEW programming and VI design. Although it might seem easiest to mimic the design of the DASY*Lab* flowchart as directly as possible, it may be more efficient to redesign an application to better fit LabVIEW programming structures.

Converting a DASY*Lab* application to LabVIEW is similar to building a LabVIEW VI. The following steps are typical sequence of a conversion:

1. Evaluate the basic design and operation of the DASY*Lab* experiment and outline the design of a corresponding LabVIEW VI.
2. For the main VI, build a front panel in LabVIEW that contains corresponding objects to all the user interface objects used in the original application. If more than one layout or arrangement of windows is used in DASY*Lab*, you most likely need to create more than one VI with corresponding front panels.
3. Develop sections of the block diagram that match linear sections of the DASY*Lab* flowchart. Linear sections are sections executed in sequence without any branching, looping, or other flow control. You might need to build subVIs to match Black Box modules or to encapsulate other logical code sequences.
4. Combine or connect the different sections of the block diagram with the appropriate programming structures.

There are some basic things to remember during the conversion of an application.

- LabVIEW VIs do not have any global settings that correspond to the settings in the experiment setup of DASY*Lab*. Configuration of timing

and I/O parameters are included directly in the block diagram as functions or parameters of different I/O functions. In addition, you can use multiple timing settings and I/O parameters in different parts of your application or for different I/O operations.

- In DASY*Lab*, all modules are configured using dialog windows. In LabVIEW, user interface controls are configured in a similar manner through pop-up menus. On the block diagram, all configuration of functions is done through the selection of the specific function and by passing in corresponding parameters.

DASYLab Experiment Execution

Once a DASY*Lab* experiment is started, it runs continuously until the user stops it. A LabVIEW block diagram runs once and then stops. You can create the continuous behavior in LabVIEW by placing the entire block diagram inside a While Loop and placing a stop button on the front panel to stop the operation of the loop and application when desired. The DASY*Lab* flowchart in Figure 3-1 can be converted into the LabVIEW block diagram in Figure 3-2. The LabVIEW VI's front panel is also shown.

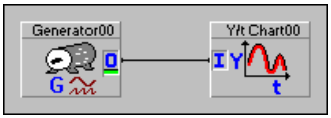


Figure 3-1. DASY*Lab* Simple Generator Flowchart

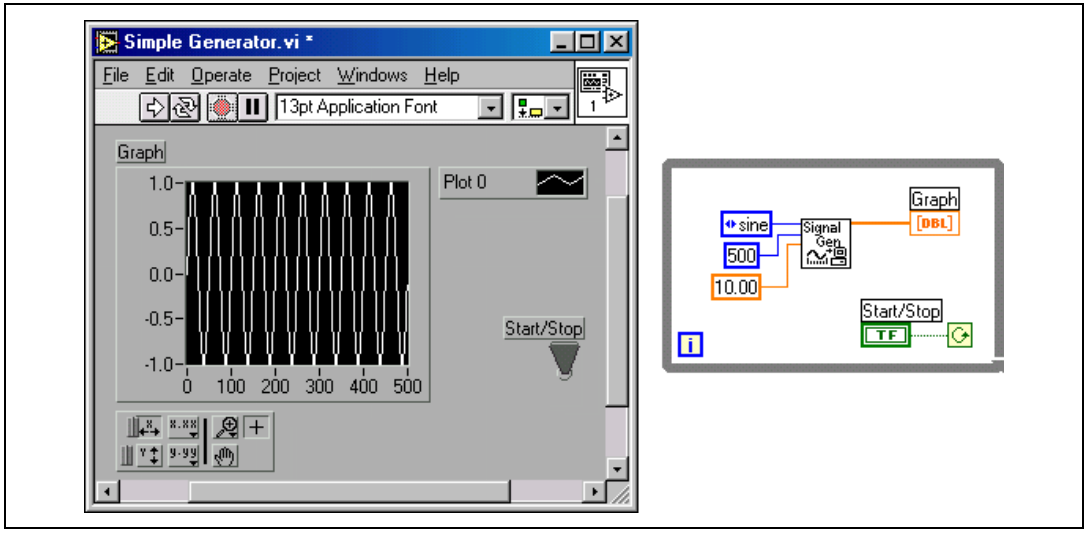


Figure 3-2. LabVIEW Simple Generator VI

Passing DASYLab Data Blocks

In DASY*Lab*, the data wires pass complete blocks of data at a time. The blocks contain qualifying information in addition to the actual data itself, such as a scan rate, time stamp, channel information, units, and so on. LabVIEW data wires pass only the information that is explicitly part of the data type.

Normally, when a signal is acquired or simulated, the function returns only the pure waveform. Other information, like the sample rate of the signal, is available as separate information and needs to be manually passed with the data for later display or processing. You can combine different pieces of information like this in a cluster, which acts similar to the DASY*Lab* data block. With the cluster, you can pass all the information with one wire in your block diagram.

For displaying data on a waveform graph, which corresponds to a Y/t chart in DASY*Lab*, there is a standard cluster format that includes the initial X value (time stamp), the incremental or delta X value, such as the sample period of a signal, and the data to plot. The data can be a 1D array for one channel or a 2D array for multiple channels.

The source of the initial X value (X_0) and incremental X value (ΔX) is defined by the developer of the block diagram. Commonly, the function supplying the data returns the sample rate or sample period. The X_0 value is frequently set at 0 for fast acquisitions. You also can read back the current system time for a time stamp similar to DASY*Lab*.

Tip You can use the Bundle function to create clusters with any elements you want to pass within the block diagram and into subVIs. With the Unbundle function, you can separate a cluster into its parts to use each data item individually.

Figure 3-3 shows how you can use the Bundle function to combine a waveform with a sample rate and X_0 value of 0 in a cluster so it is scaled correctly on the graph. The AI Acquire Waveform VI returns the sample rate used for the acquisition.

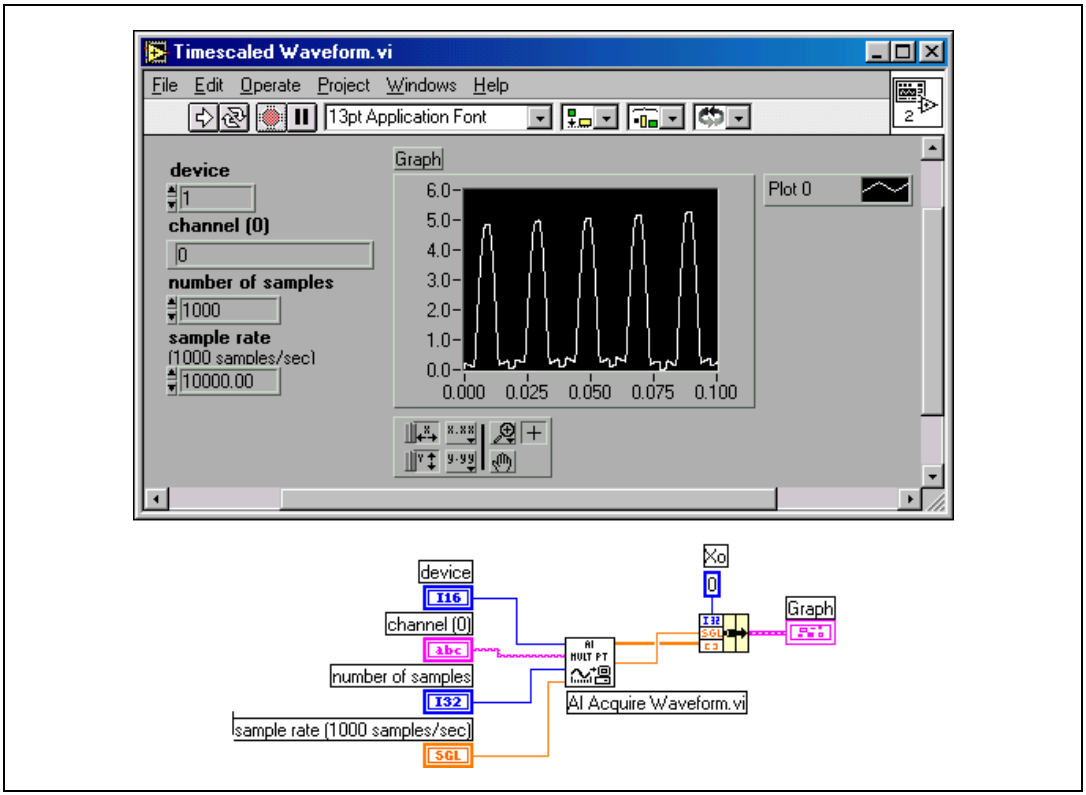


Figure 3-3. Timescaled Waveform VI

The example in Figure 3-4 acquires data from multiple channels and returns a 2D array and replaces the X_0 value with a time stamp of the system clock. In this case, the X_0 value of the graph is the current system time.

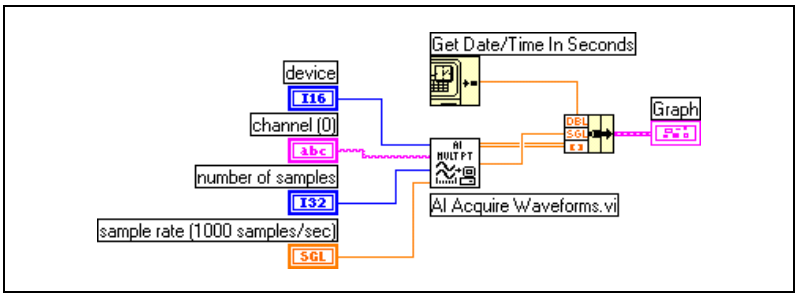


Figure 3-4. Timescaled Waveform VI with Added System Clock Time Stamp

You can apply different scaling to individual channels by building a cluster for each channel with the appropriate X_0 and ΔX information and then building an array out of the clusters, as shown in Figure 3-5. Each element of the array is a cluster that represents one waveform.

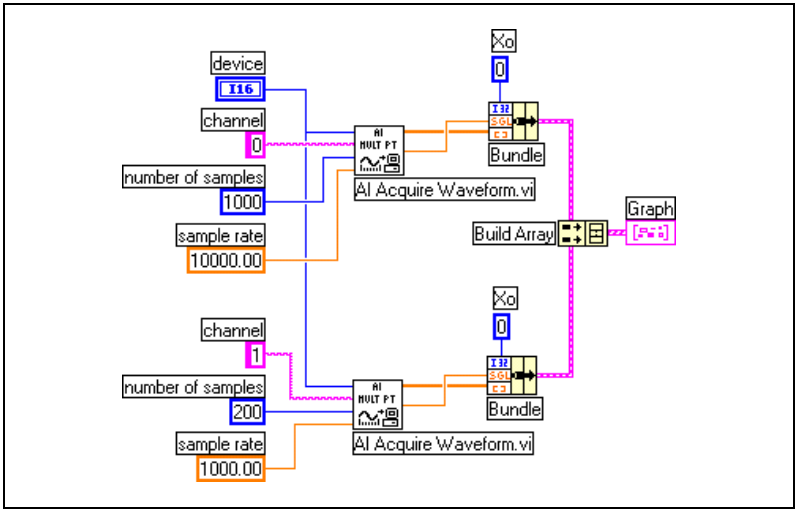


Figure 3-5. Applying Separate Scaling to Different Channels of Data

Error Cluster

One common cluster used throughout LabVIEW is the error cluster, which can be created on the front panel as a control or an indicator from the **Controls»Arrays & Cluster** palette. You use the error cluster to pass error and warning information through a sequence of VIs, such as a set of file I/O or data acquisition functions. Once an error has occurred, all subsequent functions that receive this error cluster skip their operation and pass on the error information.

Tip The input and output terminals for the error cluster are normally on the bottom corners of VIs.

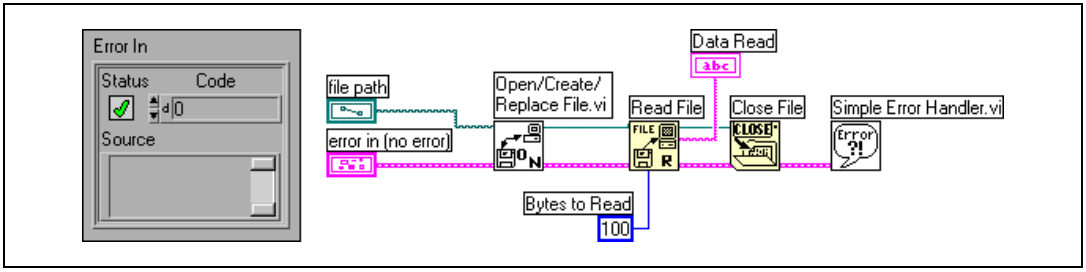


Figure 3-6. Error Cluster and Set of File I/O VIs Using the Error Cluster

Input/Output Operations

In DASY*Lab*, input/output operations are an integral part of the flowchart. Global settings, like the sample rate and block size, determine how fast or how often data is acquired from devices or instruments or how often outputs are updated.

In LabVIEW, all control of I/O devices, such as data acquisition and GPIB instrument control, is handled and configured through corresponding subVIs. Settings such as the sample or update rate are parameters of these subVIs and can be set independently for each process. For example, you can acquire data at 1 kHz with one acquisition and at 10 kHz on a different acquisition within a single application.

Data Acquisition

Note The LabVIEW information in this section is specific to National Instruments data acquisition (DAQ) devices and the NI-DAQ driver.

In DASY*Lab*, data acquisition is performed according to the capabilities of the data acquisition driver and the DASY*Lab* interface to the driver. The timing of the operations, including sample rate, block size, and buffered size, is configured in the experiment setup. Most analog input operations are performed in a continuous mode. An acquisition starts when the experiment is started, and the acquisition continues to run in the background throughout the whole experiment. In every cycle of the flowchart, a block of data is retrieved from the ongoing acquisition, and the data is processed in the modules on the flowchart.

In LabVIEW, data acquisition processes are explicitly started and stopped using different VIs in the block diagram. Analog input and other operations can be run in single point, buffered, and continuously buffered mode, determined by the specific VIs and parameters of the VIs.

The data acquisition VIs in LabVIEW are organized into easy, intermediate, and advanced levels. Each level offers the same functionality, but the intermediate and advanced levels offer more parameter options to select specific operations such as triggering, different gain settings per channel, and so on. The easy I/O VIs are easy to use but do not offer all the options or capabilities possible in the driver and hardware. Each level of DAQ VIs is built on top of the next lower level, so that you can start with the easiest and work your way into the intermediate and advanced VIs as your applications require it. The *LabVIEW Data Acquisition Basics* manual contains detailed information about the LabVIEW DAQ VIs and their use.

Simple LabVIEW DAQ Applications

At the easy I/O level, only one VI is used to perform an operation, such as acquiring a number of samples or updating a digital output line. Each of the easy I/O VIs offers a set of basic parameters for selecting a device, channels or ports, and other basic settings. The output of an easy I/O VI is a wire with the measurement data. The easy I/O VIs also perform automatic error checking and display an error message if an error occurs.

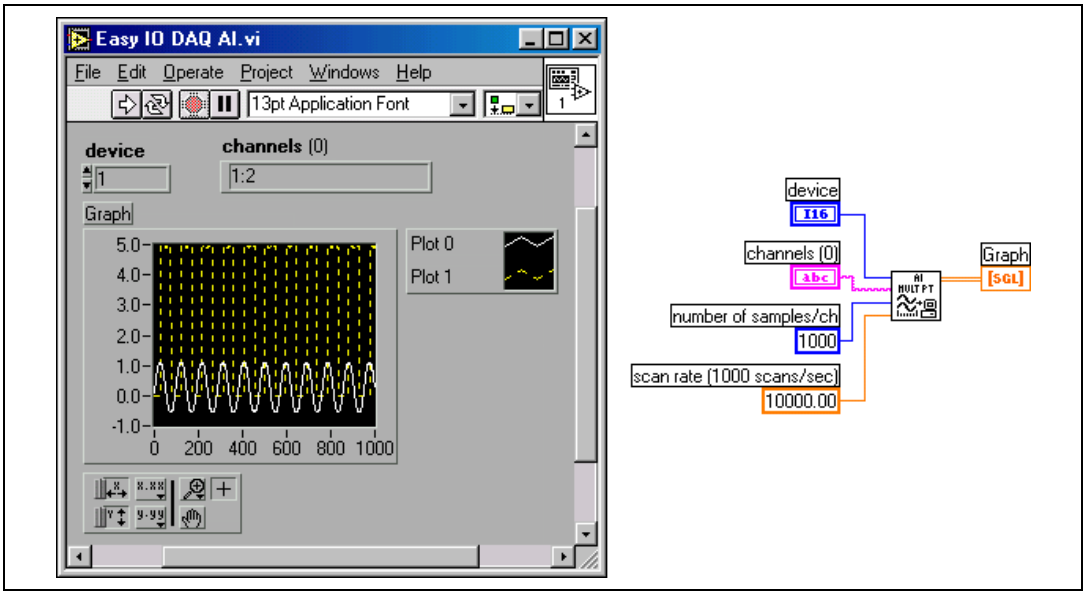


Figure 3-7. Easy I/O DAQ Analog Input Application

Converting DASYLab Diagram Flow Control to LabVIEW

This section addresses specific issues of converting modules or sections of a flowchart from DASY*Lab* to LabVIEW and describes an example DASY*Lab* flowchart and the corresponding block diagram in LabVIEW.

Continuous Data Processing—Shift Registers

In DASY*Lab*, flowchart execution and data processing is handled in a continuous manner, with the flowchart and data in each module processed continuously. For example, many of the analysis and other modules can process data from different data blocks in a continuous manner. You can find the maximum of all the data passed into the Statistics module or find the time between consecutive peaks in a signal, even if the peaks are located in consecutive data blocks.

LabVIEW VIs handle and process only the data currently passed into the VI. Using specific programming tools in LabVIEW, such as loops and shift registers, you can create the same behavior as DASY*Lab.* Shift registers enable you to calculate values in one iteration of a loop and pass them to other iterations. When you use shift registers, you can pass data of any type contained in a wire. Using the shift register in subVIs, a subVI can remember information from previous calls to the same subVI.

Example: Calculating the Running Maximum of a Signal

The DASY*Lab* flowchart in Figure 3-8 calculates the running maximum of a signal. The Statistics module is configured for Running mode. The generator generates the signal with an amplitude set with a slider.

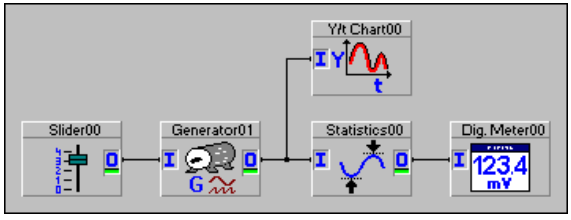


Figure 3-8. DASY*Lab* Flowchart Finding the Running Maximum of a Signal

The LabVIEW block diagram in Figure 3-9 performs the same operation. The signal is generated, with the amplitude dependent on a slider. The Array Max & Min function finds the largest value in the current array. Then, the Max & Min function returns the larger value of two values passed

in, the maximum of the current array and the maximum retrieved from the left shift register. The output, which is the new maximum, is displayed on the front panel and passed into the right shift register.

In the first iteration of the loop, the shift register is initialized with the value 0.00, as passed in by the constant wired into the left shift register from outside the While Loop. In the next iteration of the loop, the new maximum is returned from the right shift register to the left shift register.

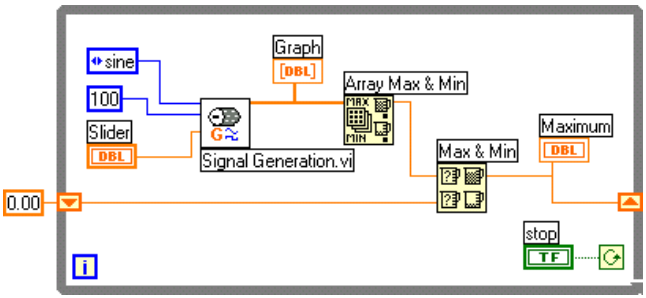


Figure 3-9. LabVIEW Block Diagram Calculating the Running Maximum of a Signal

In this example, you can build a subVI that calculates the running maximum and remembers the previous maximum into the next call. Figure 3-10 illustrates a VI and subVI that perform in this manner. The subVI uses a While Loop and shift registers to store the maximum value from one iteration of the subVI and feeds the value into the next call. The loop executes once for each call to the subVI. The shift register is uninitialized and returns the maximum value from the previous call to the subVI. The index parameter and Case structure are used to ignore the shift register and return the maximum of the current data set for the first call to the subVI.

Tip When using shift registers for this purpose, set the execution option of the VI to **Reentrant Execution**. This creates a separate memory space for each instance of the subVI on a block diagram. Otherwise, the same memory is used for each copy of the subVI, and incorrect results occur. To set a VI for reentrant execution, right-click on the icon in the upper right corner of the block diagram of the VI and select **VI Setup…**, select **Execution Options** from the top pull-down menu, and select **Reentrant Execution** from the available checkboxes.

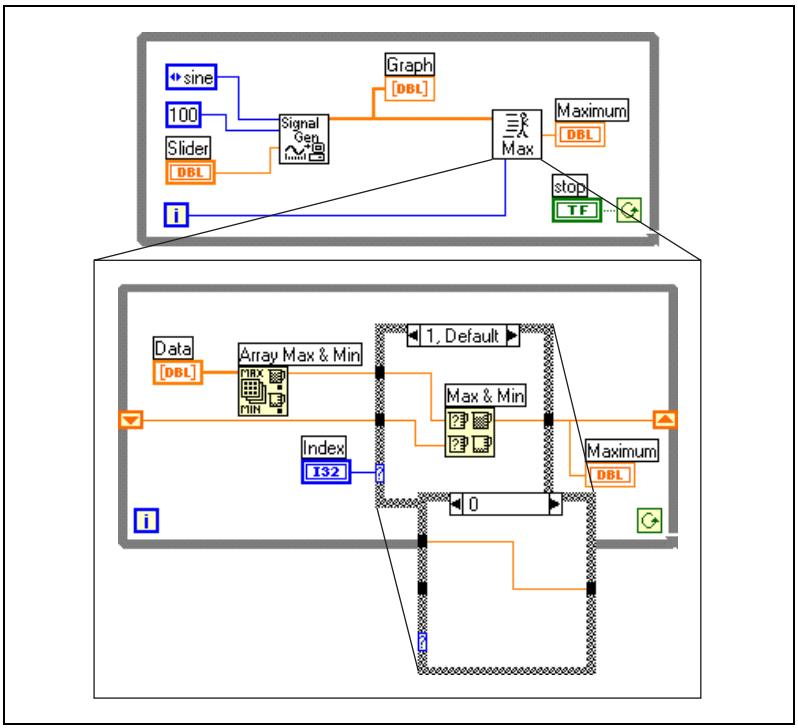


Figure 3-10. VI and SubVI to Calculate a Running Maximum

Deciding whether to put the shift register into your main VI or build a separate subVI depends on the specific application. In cases where you have a very simple operation that you will use in different places, like the running maximum, it is better to build a subVI.

Triggering

In LabVIEW, if you are using a data acquisition device, you can perform several trigger functions on your signal using the different options provided in the NI-DAQ driver and VIs. If you are using these functions for program flow control and/or to trigger signals from other sources, you can use the Case structure and Signal Processing functions for the same purpose.

Data Acquisition Triggering

The DAQ VIs, driver, and hardware include support for triggering from separate digital channels and triggering off acquired analog signals. With analog triggering, you have the option of performing triggering on the hardware, if your device supports this option, or triggering in software.

Software triggering is performed in the DAQ driver and can be selected in the DAQ VIs. Triggering performed in hardware is configured with the intermediate level AI Start VI or the Advanced Level Triggering VI. Analog software triggering, called *conditional retrieval* in LabVIEW, is configured in the Read VI. For triggering, you can specify parameters such as trigger level, hysteresis, rising and falling slope, pre/post trigger, start/stop trigger, and more, depending on the operation and device used.

LabVIEW includes many DAQ example applications, found in the `LabVIEW\Examples\DAQ` directory, that show different options for triggering. Consult these and the *LabVIEW Data Acquisition Basics* manual for more information.

Program Flow Control

In DASY*Lab*, the Trigger and Relay modules are used for triggering on signals from other sources or for program flow control. In LabVIEW, you use the Case structure and the Signal Processing and Array Manipulation functions in LabVIEW to select which operations LabVIEW performs in response to the result of your measurement.

The array manipulation functions are used to move and manipulate data in arrays, such as taking subsets of arrays, extracting columns and rows out of 2D arrays, and so on. The Signal Processing functions are used to process the data in arrays, such as calculating statistical values, finding peaks and valleys, performing windowing, filtering and spectrum analysis, and more.

Printing

Printing in DASY*Lab*, as well as report generation, is handled through the layout. You can print individual controls or the output from a message module. Printing is normally initiated from the Action module.

In LabVIEW, printing is handled through a specific VI created for this purpose. You can print the front panel of a VI, either at completion of running the specific VI or through a special function call.

To select an option to print the front panel at completion, pop up on the icon in the upper right corner of the block diagram of the VI and select **VI Setup**. Select **Execution Options** from the top pull-down menu and select **Print Panel When VI Completes Execution** from the list of available checkboxes. The VI front panel is printed when the VI has run, either as a main VI or as a subVI.

You also can print the front panel of any VI in memory by using the Print Panel VI, found in the **Functions»Application Control** palette. This VI requires only the name of the VI to be printed, and this VI does not have to be displayed on the screen. For example, all subVIs are loaded in memory when run, but the front panels are not normally shown. You can print these panels using the Print Panel VI.

Report Generation

If you need more advanced or detailed report generation in your LabVIEW application, you can use a specific report generation tool like HiQ, which is distributed with LabVIEW.

Note Only users who purchase the LabVIEW Full Development System or LabVIEW Professional Development System receive HiQ. If you purchased the LabVIEW Base Package and are interested in National Instruments HiQ, you can find more information on the National Instruments Web site (`www.natinst.com/hiq`).

HiQ is a separate environment with a notebook interface designed for creating reports and doing interactive data analysis and display. The **Communication** functions palette includes a set of VIs designed to work with and control the HiQ environment.

File I/O

File input and output is handled in LabVIEW, similar to DASY*Lab*, with a set of specific functions and VIs. The **Functions»File I/O** palette includes several simple and complex VIs and functions similar to the data acquisition functions.

Using the low-level VIs, you can freely define other formats, including binary and ASCII data. LabVIEW supports another file type called a datalog file (not to be confused with DATALOG, the developers of DASY*Lab*). A datalog file consists of a sequence of records. Each record consists of a cluster, must be identical, and can contain any LabVIEW data type. You can read a datalog file by specifying the individual record you wish to retrieve.

This toolkit also includes several additional VIs that enable you to access data saved to DASY*Lab* `.ddf` files. For more information on these VIs, see Appendix A, *DASYLab File I/O Functions*.

Messages

The message module in DASY*Lab* is used to move a text message from an application to a number of different recipients, including a dialog box, printer, or file. In LabVIEW, each recipient is handled in a different manner. Files can be written using the file I/O VIs.

For displaying dialog box windows from your application, LabVIEW includes a One Button Dialog and Two Button Dialog function that are used to display any text message. With the two button dialog, you can use a case structure to respond to either button pressed by the user.

Actions

Action modules in DASY*Lab* are used to initiate different asynchronous actions throughout an application, such as switching between layouts, resetting modules, printing, and others.

In LabVIEW, actions that apply to specific objects on the front panel are handled through an attribute node. An *attribute node* is a programmatic interface to the attribute and settings of a front panel object. You create an attribute node from the pop-up menu of the control. The attribute node is displayed on the block diagram and can be resized to include any number of attributes for the specific control. For more information on attribute nodes, see Chapter 22, *Attribute Nodes*, of the *G Programming Reference Manual*.

Network Control and Interaction

DASY*Lab* Net allows control of and interaction with DASY*Lab* environments running on other computers connected on a network. In LabVIEW, these same capabilities are possible with the application control functions.

You can build fully distributed applications with different components of an application running on different computers. With VI Server, you can run VIs on a remote machine as if you were running them locally, passing in values, executing the remote VI, and reading back results. For more information, see Chapter 21, *VI Server*, of the *G Programming Reference Manual*.

Multiple Layouts and Window Arrangements

In many applications, the user can switch between multiple views that show different data sets or different views of the same data. DASY*Lab* handles this by using multiple layouts or windows arrangements and using the Action module to select among different display options.

LabVIEW handles displaying multiple windows or user interfaces by using multiple VIs, one for each view. Normally when you call a subVI, values are passed to the subVI block diagram, processed, and other values returned without displaying the front panel of the subVI. In the VI Setup of each subVI, you have the option to show the front panel of the subVI when the subVI is called and also close it afterward. To make the front panel of a subVI display when called, pop up on the subVI, select **SubVI Node Setup**, and select **Show Front Panel when called**.

Tip Programming using subVIs and different block diagrams for different parts of your application is significantly different from a DASY*Lab* flowchart, and requires some practice and well thought-out application design. Review some of the examples in LabVIEW `Examples` directory that use multiple VIs to become familiar with selecting or calling subVIs.

LabVIEW Tools Beyond DASYLab

LabVIEW includes the following additional functions and tools that do not have parallels in the DASY*Lab* environment.

Menu Bars

You can edit the menu bars with the Menu Editor, available by selecting **Edit»Edit Menu**.You can edit the existing menu bar or create your own. The settings for your menu bar are saved in a separate file with the `.rtm` extension and can be directly applied to the current VI.

The **Functions»Application Control»Menu** palette contains a set of functions to read and react to interaction with the menu bar and to dynamically change the options in the menu bar. See the *LabVIEW Online Reference* available by selecting **Help»Online Reference** and the menu bar examples in `Examples\General\menubar.llb` for more information on the Menu functions.

Open Network Communication

LabVIEW includes functions and VIs for a variety of network and interprocess communication options. You use these communication tools to interact among your LabVIEW application and many other applications, servers, and programs. Several of these functions can be used to communicate to other computers connected through a network. LabVIEW supports the following communication methods:

- TCP/IP
- UDP
- DDE
- ActiveX automation

The ActiveX automation functions are similar to the application control functions of the VI Server, including property and invoke nodes. With these functions, you can interact with and control other applications that can act as an ActiveX automation server in the same manner as you control another LabVIEW application or VI.

LabVIEW is an ActiveX automation server, so you can control and automate the LabVIEW environment and call different VIs from other applications and programming environments that act as ActiveX automation clients, such as Visual Basic and Visual C++, Microsoft Excel, and many more.

For more information on communication, refer to the *LabVIEW Online Reference*, available by selecting **Help»Online Reference**, and Chapters 20 – 24 of the *LabVIEW User Manual*.

ActiveX Controls

To extend the range of user interface controls and to add more functionality to the LabVIEW environment, you can import ActiveX controls to a LabVIEW application. ActiveX controls are self-contained software components that can be added to standard ActiveX container applications such as LabVIEW.

For more information about Active X controls, see Chapter 16, *ActiveX Controls*, of the *G Programming Reference Manual*. Additional information about ActiveX events is also available in the *LabVIEW 5.1 Addendum*.

DLLs

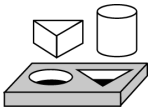
Another option to add functionality to LabVIEW is to access standard DLLs and call functions from these libraries. You can call any standard Windows DLL from the LabVIEW block diagram using the Call Library Function in the **Functions»Advanced** palette.

Tip If you plan to use a DLL or function repeatedly, it is a good idea to build a separate subVI as a wrapper for each function call in the library and use these subVIs in your block diagrams. See Chapter 25, *Calling Code from Other Languages*, in the *G Programming Reference Manual* for additional information on calling DLLs from LabVIEW.

Instrument Drivers

Instrument drivers are libraries of special functions written to access specific instruments connected to the computer through different buses, including GPIB, VXI, and serial bus.

Instrument drivers encapsulate both the instrument-specific command language and parsing routines in easy-to-use measurement-focused functions. Each driver is written for a specific instrument or series of instruments, and you can integrate the functions from a driver into your application to quickly start using the instrument. There are more than 650 free instrument drivers available for LabVIEW that are provided on the instrument driver CD or that can be downloaded from the National Instruments Web site (`www.natinst.com`). New drivers are added to the Web site as they become available. Instrument manufacturers might also have LabVIEW instrument drivers available for their instruments that are not in the list.

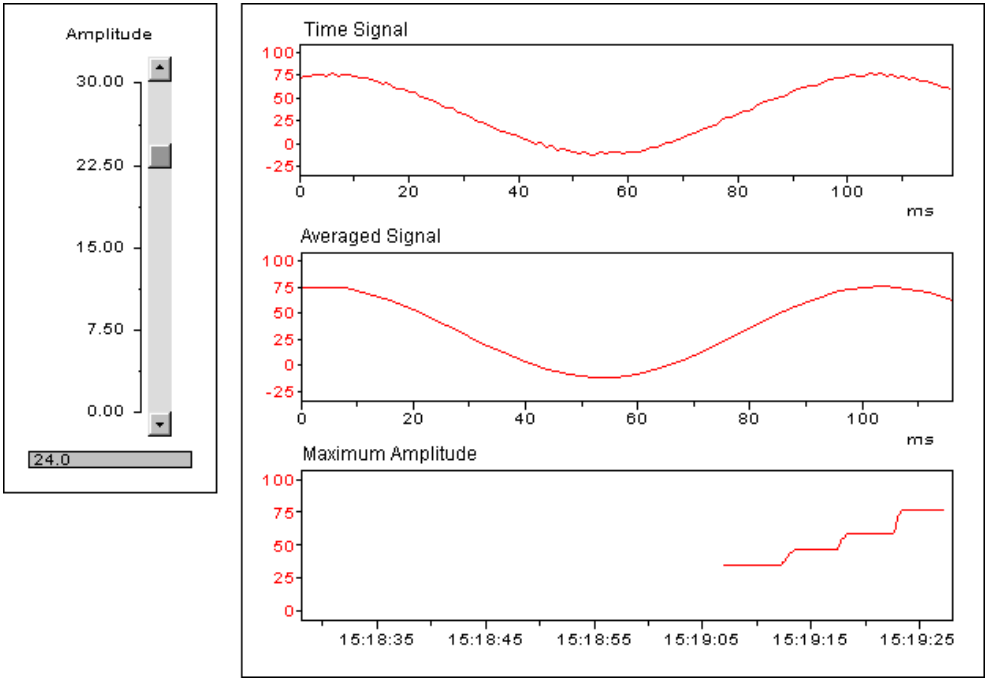


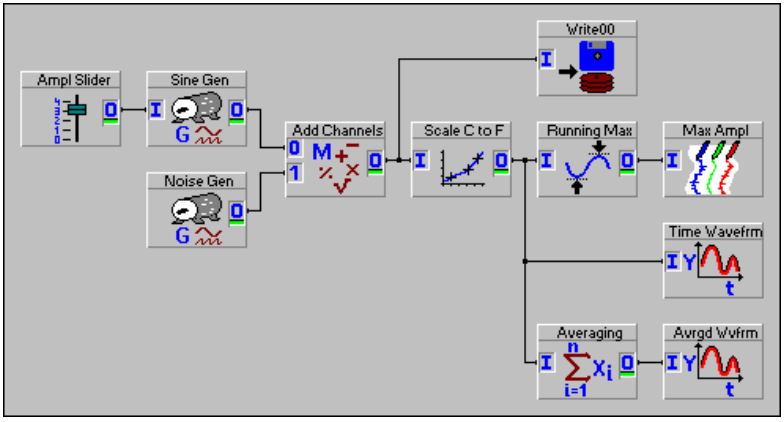
Activity 3-1. Converting a DASYLab Experiment to LabVIEW

The following activity describes converting a DASY*Lab* application into LabVIEW.

The following graphic shows the DASY*Lab* application flowchart and corresponding layout for `sample application1.dsb`. The application generates a sine signal with a varying amplitude set by a slider. It also generates a noise signal, adds these two signals together, and stores the resulting waveform in a file. The waveform is then scaled from Celsius to Fahrenheit and is displayed on a graph. The running maximum of the waveform is calculated and displayed on a recorder. A running average of

the waveform is made and displayed on another graph. The global settings of the experiment are 1000 Hz sample rate and a block size of 120.





Front Panel

The first step in developing the LabVIEW application is to build the user interface. Follow these steps to develop the UI.

1. Add a Waveform Graph from the **Controls»Graph** palette and label it `Time Waveform`.

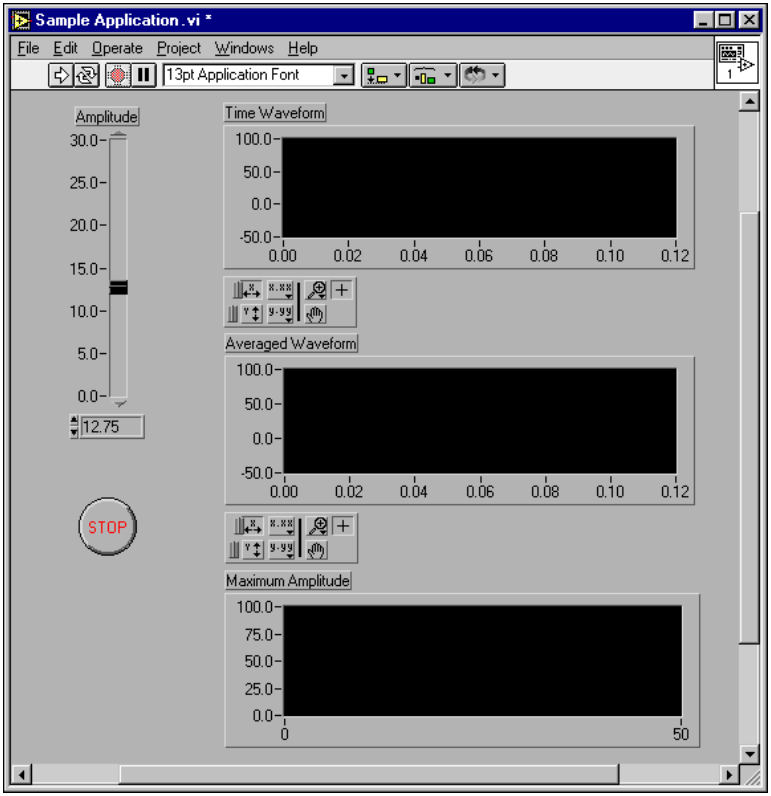
2. Add a Waveform Graph from the **Controls»Graph** palette and label it `Averaged Waveform`.

3. Add a Waveform Chart from the **Controls»Graph** palette and label it `Maximum Amplitude`. Change the Y axis maximum to 100.

4. Add a slider from the **Controls»Numeric** palette and label it `Amplitude`. Change the maximum to 30.

5. Add a Stop button from the **Controls»Boolean** palette.
6. Save your VI as `Sample Application.vi`.

The following graphic shows an example of what the front panel of your VI might look like.



Block Diagram

A While Loop allows the application to run continuously, which is done automatically in the DASY*Lab* application. The While Loop repeats the graphical code inside the loop until the Stop button is pressed.

7. Place a While Loop from the **Functions»Structures** palette on the block diagram. Most of the functions of the VI will be placed within the While Loop, so size the While Loop accordingly.

8. Wire the Boolean control of the Stop button to the termination icon of the While Loop through a logic Not function, found on the **Functions»Boolean** palette. Doing this ensures that while the button is FALSE, the loop continues to run.

Signal Generation

The Signal Generation VI creates the waveform, similar to the Generator in DASY*Lab*. The Phase In and Phase Out parameters create a continuous signal matching the phase of one data block to the next data block. A shift register on the While Loop passes the Phase Out data from one iteration of the loop to the Phase In parameter of the next iteration of the loop.

9. Add the Signal Generation VI found in the `Examples\DASYLab\Activity` folder, by selecting **Functions»Select a VI**. Create constants for the Signal Generation VI parameters by popping up on the appropriate terminal and selecting **Create Constant**. Use the following values:
   - waveform type—`sine`
   - number of samples—`120`
   - frequency—`10.0`
   - sample rate—`1000`
10. Wire the Amplitude slider control to the amplitude input terminal of the Signal Generation VI.

11. Calculate the duration of one data block and wire it to the Duration input of the Signal Generation VI using a Divide function, found in the **Functions»Numeric** palette. Divide the number of samples (120) by the effective sample rate (1000).
12. Create a shift register on the While Loop by popping up on the border of the While Loop and selecting **Add Shift Register**. Wire the Phase Out terminal from the Signal Generation VI to the right shift register, and the Phase In terminal of the Signal Generation VI to the left shift register.

13. Create a second Signal Generation VI. Create constants to configure it to generate noise with amplitude 1.0 and the same number of samples as the first Signal Generation VI.

14. Add the two signals using an Addition function from the **Functions»Numeric** palette.

Write Data to File

Use the Write to Spreadsheet File VI to write the waveform data to a file.

15. Inside the While Loop, create the Write to Spreadsheet File VI, found in the **Functions»File I/O** palette. Wire the combined waveform to the 1D Data input of the Write to Spreadsheet File VI.
16. Create a constant for the Append to file? input of the VI and set the constant to True.

If no filename is specified on the VI, it prompts the user for a filename every time the VI is called.

17. Create a File Dialog function from the **Functions»File I/O»Advanced File Functions** palette outside and to the left of the While Loop.
18. For the Prompt input of the function, pop up on the input and select **Create Constant** to create a string constant and enter the string, `Select File Name to write data to:`.
19. For the select mode input of the function, right-click on the input and select **Create Constant** to create an enum and select `new or existing file`.
20. Wire the path terminal of the function into the While Loop and to the file path input of the Write to Spreadsheet File VI.

Data Scaling

The combined waveform is scaled using the conversion from Celsius to Fahrenheit. This can be done using the arithmetic functions in LabVIEW. These functions are polymorphic, meaning they can process scalar and array data.

21. Create a Multiply function from the **Functions»Numeric** palette. Wire the waveform to one input of the Multiply function and create a constant with value 1.8 on the other input.

22. Create an Add function from the **Functions»Numeric** palette. Wire the output of the Multiply function to one input of the Add function and create a constant with value 32.0 on the other input.

Display Time Waveform

To display the time waveform generated by the scaling, you need to associate the proper sample period with the generated data before passing it to the graph. The association is handled using a Bundle function that creates a cluster (data structure) that contains the data, an X_0 and ΔX value. X_0 is used for the X value of the first point in the data and corresponds to the data block time stamp in DASY*Lab*. The ΔX value specifies the incremental X value on the graph. For a time waveform this is the sample period (1/sample rate) of the signal.

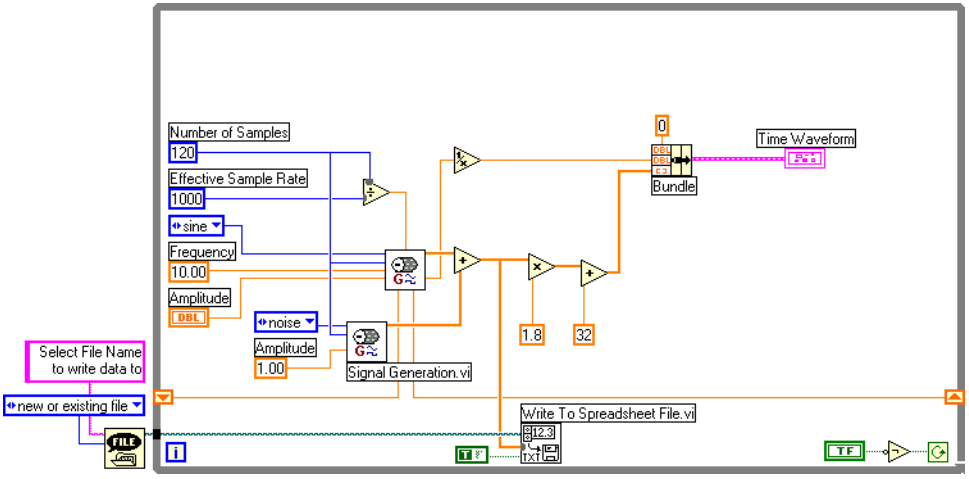
23. Create a Bundle function from the **Functions»Cluster** palette. Resize it for three inputs. Create a numeric constant with value 0.0 and wire it to the first input of the Bundle function as the X_0 value.

24. Calculate the sample period from the sample rate output of the Signal Generation VI. Do this by adding a Reciprocal (1/x) function from the **Functions»Numeric** palette and wiring it to the second input of the Bundle function.

25. Wire the scaled time waveform to the third input of the Bundle function and wire the Bundle output to the Time Waveform graph indicator.

Checkpoint

Your VI is not finished yet, but this is a good point to save your progress and arrange any wires, functions, and subVIs so your block diagram can be easily read. The following graphic shows an example of an effectively organized block diagram up to this point in this activity.



Loop Timing

Use the Wait Until Next ms Multiple, or Metronome, function to control the timing of the While Loop. In the DASY*Lab* application, this is handled by the Generator module, which releases data according to the global settings. Without the Metronome function, the While Loop runs as fast as possible, which can cause your computer to appear slow and unresponsive.

26. Add a Wait Until Next ms Multiple function from the **Functions»Time & Dialog** palette. Calculate the loop delay in milliseconds by multiplying the data block duration by 1000. Wire the output of the Multiply function to the Wait Until Next Multiple ms function.

Running Average

There is no pre-built VI or function in LabVIEW for calculating a running average, so you must create this functionality using arithmetic functions. In this activity, you average four data points at time, so the 120 data points are reduced to 30 averaged data points.

27. Create a For Loop from the **Functions»Structures** palette inside the While Loop. The For Loop runs once for each average calculated.

28. Calculate the number of loop iterations by dividing the number of samples (120) by the number of samples per average (4). Create a Divide function from the **Functions»Numeric** palette for this purpose. Wire the output of the Divide to the For Loop count input.

29. Inside the For Loop, place an Array Subset function from the **Functions»Array** palette and a Mean VI from the **Mathematics»Probability and Statistics** palette.

30. The Array Subset function is used to extract four points at a time. Wire the constant value 4 to the Array Length input of the Array Subset function. Create a Multiply function and wire the output of the Multiply function to the Array Index input of the Array Subset function.

31. Wire the output of the Array Subset function to the input of the Mean VI. This calculates the mean for each four sample subarrays.

32. Wire the output of the Mean VI to the outside of the For Loop. At the boundary all of the average values are stored in a new array for display outside of the loop.

Display Average Array

Display the running average array on a graph. Use the Bundle function to combine the array data with the scaling information for proper scaling on the graph.

33. Create another Bundle function from the **Functions»Cluster** palette and resize it for three inputs. Create a numeric constant, set it to 0, and wire it to the first input of the Bundle function.
34. The effective sample period for the averaged data is four times the actual sample rate (one fourth the number of samples), so use a Multiply function to multiply the actual sample rate from the other Bundle function by 4 and wire it to the second input of the Bundle.

35. Wire the averaged data array from the For Loop to the third input of the Bundle function and wire the Bundle output to the Averaged Waveform graph indicator.

Calculate and Display the Running Maximum Amplitude

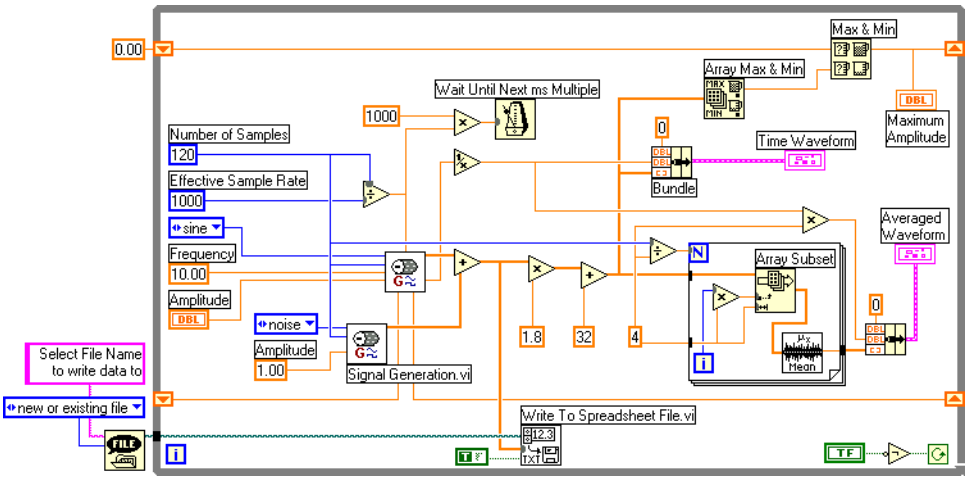
The Array Max & Min function and Max & Min function, combined with a shift register are used to calculate the running maximum amplitude of the signal. The Array Max & Min function is used to find the maximum amplitude in each waveform. The Max & Min function compares the current maximum amplitude with the previous maximum to find the new maximum amplitude, which is passed to the next iteration of the loop using the shift register.

36. Add an Array Max & Min function from the **Functions»Array** palette. Wire the scaled time waveform to the Array Max & Min input.

37. Add a Max & Min function from the **Functions»Comparison** palette. Wire the output of the Array Max & Min function to one input of the Max & Min function.
38. Create another shift register on the While Loop. Wire the left shift register to the other input of the Max & Min function. Wire the output from the Max & Min function to the right shift register.
39. Create a numeric constant with value 0.00 outside the While Loop and wire it to the left terminal of the shift register. This initializes the value of the shift register when the application is started.
40. Wire the output of the Max & Min function to the Maximum Amplitude chart.

Clean Up the Block Diagram

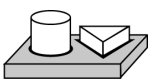
The functionality of your VI is now finished, so this is a good point to arrange any wires, functions, and subVIs so your block diagram can be easily read. The following graphic shows an example of an effectively organized block diagram of the completed Sample Application VI.



Running and Operating the VI

41. Save the VI.
42. Run the VI. Change the amplitude of the generated signal using the slider control. The time signal and averaged waveform are displayed on the graphs. The maximum amplitude of the signal is displayed on the chart.

To simplify the block diagram, you can combine different logical groups of VIs and functions into their own subVIs. In addition, this allows you to reuse these subVIs in other applications. To build subVIs from an existing block diagram, select a portion of the block diagram and select **Edit»Create SubVI**. `Sample Application2.vi` in the `Examples\DASYLab` directory shows this VI using subVIs.



End of Activity 3-1.

A

DASYLab File I/O Functions

The DASY*Lab* to LabVIEW Migration Toolkit includes a number of functions installed in the **Functions** palette for exchanging data between DASY*Lab* and LabVIEW using files. Data can be written by DASY*Lab* in its native format (DDF) and be read in LabVIEW. The I32 binary format can be used to exchange data in both directions without timing information.

The most commonly used of these functions is the Read DASY*Lab* DDF File VI, which returns data in the same blocks into LabVIEW as it was written in DASY*Lab.* This VI returns both scaled and time-scaled data for processing in your block diagram. The scaled data includes the pure data in measured units. The time-scaled data includes timing information and returns the data in LabVIEW clusters suitable for passing to the waveform graph. Data from each channel is returned in a separate cluster, and all channels are combined in an array of these clusters.

Each call to this VI returns one block of data. Making repeated calls to the VI and using the Read Offset parameter (pass in last value from Mark After Read) you can retrieve consecutive data blocks as in DASY*Lab.*

The toolkit includes several examples located in the `Examples\DASYLab` directory that illustrate how to use the different DASY*Lab* File I/O functions.

Read DASYLab DDF File VI

The Read DASY*Lab* DDF File VI reads data blocks from files stored in DASY*Lab* format or streaming data format (`.ddf`).

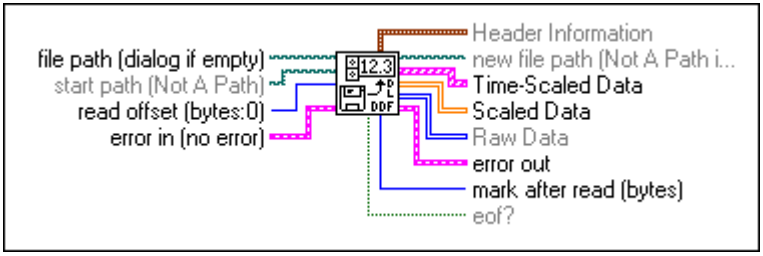
Note This VI does not support the Microstar DAP streaming data format or multiplexed data channels.

The Read DASY*Lab* DDF File VI can be called repeated times by passing the offset from the previous call of the VI into the next VI using a shift register on a for or while loop.

Data is returned in the following three forms:

- Raw data (streaming data format only)—Integer data stored directly from the driver
- Scaled data—Voltage or other scaled units without time information
- Time-scaled data—Scaled data with X_0 and ΔX information ready for graphing

If the header information and data are stored in separate files, specify the header file name to open, and the Read DASY*Lab* DDF File VI automatically opens the data files. The data file must have the same name, a `.ddb` extension, and must be located in the same directory as the header file.



file path (dialog if empty) is the pathname of the file. If filepath is empty (default value) or is Not A Path, the VI displays a file dialog box from which you can select a file. Error 43 occurs if the user cancels the dialog box.

start path (Not A Path) is the pathname to the initially displayed directory (or folder) in a file dialog. The default value is Not A Path, which is the path to the last directory (or folder) shown in a file dialog box.

read-offset (bytes:0) is the position in the file, measured in bytes, at which the VI begins reading.

error in (no error) describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster in **error out**.

status is TRUE if an error occurred before the VI was called or FALSE if not. If **status** is TRUE, code is a nonzero error code. If **status** is FALSE, code can be 0 or a warning code.

code is the number identifying an error or warning. If **status** is TRUE, **code** is a nonzero error code. If **status** is FALSE, **code** can be 0 or a warning code. Use the Error Handler VIs to look up the meaning of this code and to display the corresponding error message.

source is a string that indicates the origin of the error, if any. Usually, **source** is the name of the VI in which the error occurred.

Header Information contains general information about the data in the file. It is read once at the beginning of the file.

Start Time is the start time of data in the file.

Sample Period is the sample period of data in the file.

Number of Channels is the number of channels stored in the file.

new file path (Not A Path if cancelled) is the path of the file from which the VI read data. You can use this output to determine the path of a file that you opened using the file dialog box. **new file path** returns Not A Path if the user selects **Cancel** from the dialog box.

Time-Scaled Data is an array of clusters. It contains data scaled to its proper units and includes timing information so that it can be directly passed to a LabVIEW waveform graph. Each channel of data includes its own timing information.

Scaled Data contains the data from the file scaled to its proper units.

Raw Data is binary data read from a streaming data file written directly from the data acquisition in DASY*Lab*. The data is unscaled.

Scaled Data is the cluster containing time-scaled data for one channel.

Start is the start time for the data block.

Interval is the time interval (sample period) for the data block.

Data is the scaled data.

error out is a cluster that describes the error status after the VI executes. If an error occurred before the VI was called, **error out** is the same as error in. Otherwise, **error out** shows the error, if any, that occurred in the VI. Use the Error Handler VIs to look up the error code and to display the corresponding error message.

status is TRUE if an error occurred or FALSE if not. If **status** is TRUE, **code** is a nonzero error code. If **status** is FALSE, **code** can be 0 or a warning code.

code is the number identifying an error or warning. If **status** is TRUE, **code** is a nonzero error code. If **status** is FALSE, **code** can be 0 or a warning code. Use the Error Handler VIs to look up the meaning of this code and to display the corresponding error message.

source is a string that indicates the origin of the error, if any. Usually, **source** is the name of the VI in which the error occurred.

mark after read (bytes) is the location of the file mark after the read. It points to the byte in the file following the last byte read.

eof? is TRUE when you reach the end of the file while reading.

Read All From DASYLab DDF VI

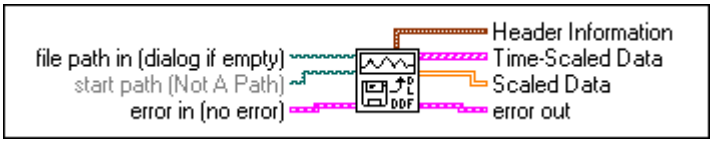
The Read All From DASY*Lab* DDF VI reads the complete data set files stored by DASY*Lab* in DASY*Lab* format or streaming data format (`.ddf`). This VI does not support the Microstar DAP streaming data format. It does not support multiplexed data channels.

The Read All From DASY*Lab* DDF VI is called once to read the entire data set in the file. Depending on the size of the file, this can take some time. With very large files, this can cause a memory overflow if not enough memory is available to store the entire data set. In such cases, use the Read From DASY*Lab* DDF VI to read individual data blocks at a time.

Data is returned in the following two forms:

- Scaled data—Voltage or other scaled units without time information
- Time-scaled data—Scaled data with X_0 and ΔX information ready for graphing

If the header information and data are stored in separate files, specify the header file name to open, and the Read All From DASY*Lab* DDF VI automatically opens the data files. The data file must have the same name, a `.ddb` extension, and must be located in the same directory as the header file.



file path in (dialog if empty) is the pathname of the file. If filepath is empty (default value) or is Not A Path, the VI displays a file dialog box from which you can select a file. Error 43 occurs if the user cancels the dialog box.

start path (Not A Path) is the pathname to the initially displayed directory (or folder) in a file dialog box. The default value is Not A Path, which is the path to the last directory (or folder) shown in a file dialog box.

error in (no error) describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster in **error out**.

status is TRUE if an error occurred before the VI was called or FALSE if not. If **status** is TRUE, **code** is a nonzero error code. If **status** is FALSE, **code** can be 0 or a warning code.

code is the number identifying an error or warning. If **status** is TRUE, **code** is a nonzero error code. If **status** is FALSE, **code** can be 0 or a warning code. Use the Error Handler VIs to look up the meaning of this code and to display the corresponding error message.

source is a string that indicates the origin of the error, if any. Usually, **source** is the name of the VI in which the error occurred.

Header Information contains the header information from file.

Start Time is the start time of data in file.

Sample Period is the sample period of data in file.

Number of Channels is the number of channels stored in file.

Time-Scaled Data is the cluster of arrays containing complete data from file, combined with timing information. Time-scaled data can be passed directly to a LabVIEW waveform graph.

Scaled Data is the cluster containing time-scaled data for one channel.

Start is the start time for the data block.

Interval is the time interval (sample period) for the data block.

Data is the scaled data.

Scaled Data is the array containing the complete data set of the file. No timing information is included.

error out is a cluster that describes the error status after the VI executes. If an error occurred before the VI was called, **error out** is the same as error in. Otherwise, **error out** shows the error, if any, that occurred in the VI. Use the Error Handler VIs to look up the error code and to display the corresponding error message.

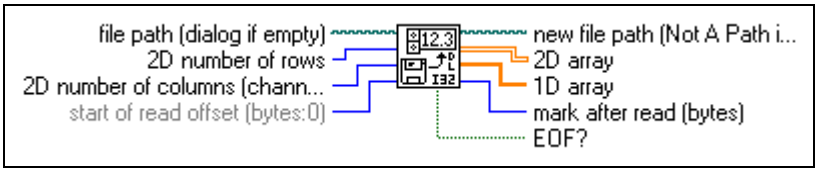
status is the number identifying an error or warning. If **status** is TRUE, **code** is a nonzero error code. If **status** is FALSE, **code** can be 0 or a warning code. Use the Error Handler VIs to look up the meaning of this code and to display the corresponding error message.

code is the number identifying an error or warning. If **status** is TRUE, **code** is a nonzero error code. If **status** is FALSE, **code** can be 0 or a warning code. Use the Error Handler VIs to look up the meaning of this code and to display the corresponding error message.

source is a string that indicates the origin of the error, if any. Usually, **source** is the name of the VI in which the error occurred.

Read DASYLab I32 File VI

The Read DASY*Lab* I32 File VI reads DASY*Lab* binary (I32) files. The byte order of the data in the files is reversed from the normal LabVIEW binary file format, therefore the byte order is reversed in this VI. You must specify the number of columns (channels) and number of rows (samples per channel) to read. For files with only one channel of data, set the number of rows to 0 and specify the number of samples to read in the number of columns parameter.



file path (dialog if empty) is the pathname of the file. If the filepath is empty (default value) or is Not A Path, the VI displays a File dialog box from which you can select a file. Error 43 occurs if the user cancels the dialog box.

2D number of rows is the number of rows to create if the data is to be returned in the 2D array output. If the value is 0 (default), the data is returned in **1D array**.

2D number of columns (channels)/1D count (all as 1D: –1) is the number of columns to create if the data is to be returned in the 2D array output, provided that **2D number of rows** is greater than 0. Otherwise, this is the number of single-precision numbers to read and return in a 1D array.

start of read offset (bytes:0) is the position in the file, measured in bytes, at which the VI begins reading.

new file path (Not A Path if cancelled) is the path of the file from which the VI read data. You can use this output to determine the path of a file that you opened using the file dialog box. **new file path** returns Not A Path if the user selects **Cancel** from the dialog box.

2D array contains the single-precision numbers the VI writes to the file if 1D data is not wired or empty.

1D array contains the single-precision numbers read from the file if 2D number of rows equal 0; otherwise, this output is empty.

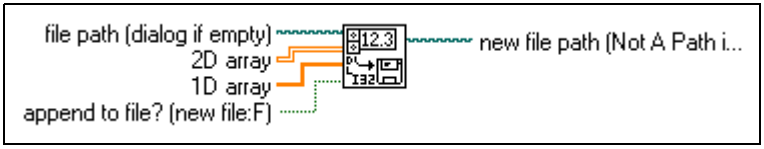
mark after read (bytes) is the location of the file mark after the read; it points to the byte in the file following the last byte read.

EOF? is TRUE if you attempt to read past the end of file.

Write DASYLab I32 File VI

The Write DASY*Lab* I32 File VI writes a 2D or 1D array of single-precision numbers (SGL) to a DASY*Lab* binary file (I32), or appends the data to an existing file. This VI opens or creates the file beforehand and closes it afterwards. You can use this VI to write scaled data from data acquisition VIs.

The byte order of the data in the DASY*Lab* I32 files is reversed from the normal LabVIEW binary file format, therefore the byte order is reversed in this VI.



file path (dialog if empty) is the pathname of the file. If filepath is empty (default value) or is Not A Path, the VI displays a file dialog box from which you can select a file. Error 43 occurs if the user cancels the dialog box.

2D array contains the single-precision numbers the VI writes to the file if **1D data** is not wired or empty.

1D array contains the single-precision numbers read from the file if **2D number of rows** equal 0. Otherwise, this output is empty.

append to file? (new file:F) Set to True to append the data to a existing file. Set **append to file?** to FALSE (default value) to write the data to a new file or to replace an existing file.

new file path (Not A Path if cancelled) is the path of the file from which the VI read data. You can use this output to determine the path of a file that you opened using the file dialog box. **new file path** returns Not A Path if the user selects **Cancel** from the dialog box.

B

Technical Support Resources

This appendix describes the comprehensive resources available to you in the Technical Support section of the National Instruments Web site and provides technical support telephone numbers for you to use if you have trouble connecting to our Web site or if you do not have internet access.

NI Web Support

To provide you with immediate answers and solutions 24 hours a day, 365 days a year, National Instruments maintains extensive online technical support resources. They are available to you at no cost, are updated daily, and can be found in the Technical Support section of our Web site at `www.natinst.com/support`.

Online Problem-Solving and Diagnostic Resources

- **KnowledgeBase**—A searchable database containing thousands of frequently asked questions (FAQs) and their corresponding answers or solutions, including special sections devoted to our newest products. The database is updated daily in response to new customer experiences and feedback.
- **Troubleshooting Wizards**—Step-by-step guides lead you through common problems and answer questions about our entire product line. Wizards include screen shots that illustrate the steps being described and provide detailed information ranging from simple getting started instructions to advanced topics.
- **Product Manuals**—A comprehensive, searchable library of the latest editions of National Instruments hardware and software product manuals.
- **Hardware Reference Database**—A searchable database containing brief hardware descriptions, mechanical drawings, and helpful images of jumper settings and connector pinouts.
- **Application Notes**—A library with more than 100 short papers addressing specific topics such as creating and calling DLLs, developing your own instrument driver software, and porting applications between platforms and operating systems.

Software-Related Resources

- **Instrument Driver Network**—A library with hundreds of instrument drivers for control of standalone instruments via GPIB, VXI, or serial interfaces. You also can submit a request for a particular instrument driver if it does not already appear in the library.
- **Example Programs Database**—A database with numerous, non-shipping example programs for National Instruments programming environments. You can use them to complement the example programs that are already included with National Instruments products.
- **Software Library**—A library with updates and patches to application software, links to the latest versions of driver software for National Instruments hardware products, and utility routines.

Worldwide Support

National Instruments has offices located around the globe. Many branch offices maintain a Web site to provide information on local services. You can access these Web sites from `www.natinst.com/worldwide`.

If you have trouble connecting to our Web site, please contact your local National Instruments office or the source from which you purchased your National Instruments product(s) to obtain support.

For telephone support in the United States, dial 512 795 8248. For telephone support outside the United States, contact your local branch office:

Australia 03 9879 5166, Austria 0662 45 79 90 0, Belgium 02 757 00 20,
Brazil 011 284 5011, Canada (Ontario) 905 785 0085,
Canada (Québec) 514 694 8521, China 0755 3904939,
Denmark 45 76 26 00, Finland 09 725 725 11, France 01 48 14 24 24,
Germany 089 741 31 30, Hong Kong 2645 3186, India 91805275406,
Israel 03 6120092, Italy 02 413091, Japan 03 5472 2970,
Korea 02 596 7456, Mexico (D.F.) 5 280 7625,
Mexico (Monterrey) 8 357 7695, Netherlands 0348 433466,
Norway 32 27 73 00, Singapore 2265886, Spain (Madrid) 91 640 0085,
Spain (Barcelona) 93 582 0251, Sweden 08 587 895 00,
Switzerland 056 200 51 51, Taiwan 02 2377 1200,
United Kingdom 01635 523545

Glossary

A

array	Ordered, indexed list of data elements of the same type.
Attribute Node	Special block diagram nodes you can use to control the appearance and functionality of controls and indicators.
auto-indexing	Capability of loop structures to disassemble and assemble arrays at their borders. As an array enters a loop with auto-indexing enabled, the loop automatically disassembles it with scalars extracted from one-dimensional arrays, one-dimensional arrays extracted from two-dimensional arrays, and so on. Loops assemble data into arrays as they exit the loop according to the reverse of the same procedure.

B

block diagram	A pictorial description or representation of a program or algorithm. The block diagram, which consists of executable icons called nodes and wires that carry data between the nodes, is the source code for the VI.
Boolean controls and indicators	Front panel objects used to manipulate and display Boolean (TRUE or FALSE) data.
Bundle node	Function that creates clusters from various types of elements.

C

Case structure	Conditional branching control structure, which executes one and only one of its subdiagrams based on its input. It is the combination of the IF, THEN, ELSE, and CASE statements in control flow languages.
cluster	Set of ordered, unindexed data elements of any data type including numeric, Boolean, string, array, or cluster. The elements must be all controls or all indicators.
conditional terminal	The terminal of a While Loop that contains a Boolean value that determines whether the VI performs another iteration.
coercion dot	Gray dot on a terminal indicating that one of two terminals wired together has been converted to match the data type of the other.

conditional retrieval	A method of triggering in which you to simulate an analog trigger using software. *Also called* software triggering.
conditional terminal	Terminal of a While Loop that contains a Boolean value that determines whether the VI performs another iteration.
control	Front panel object for entering data to a VI interactively or to a subVI programmatically.
Controls palette	Palette containing front panel controls and indicators.
count terminal	Terminal of a For Loop whose value determines the number of times a For Loop executes its subdiagram.

D

data flow	Programming system consisting of executable nodes in which nodes execute only when they have received all the required input data and produce output automatically when they have executed. LabVIEW is a dataflow system.
data type	Format for information. In BridgeVIEW, acceptable data types for tag configuration are analog, discrete, bit array, and string. In LabVIEW, acceptable data types for most functions are numeric, array, string, and cluster.
datalog file	File that stores data as a sequence of records of a single, arbitrary data type that you specify when you create the file. While all the records in a datalog file must be of a single type, that type can be complex; for instance, each record can be a cluster containing a string, a number, and an array.

E

Enable Indexing	Option that allows you to build a set of data to be released at the termination of a While Loop. With indexing disabled, a While Loop releases only the final data point generated within the loop.

F

For Loop	Iterative loop structure that executes its subdiagram a set number of times. Equivalent to conventional code: `For i = 0 to n – 1, do....`
front panel	The interactive user interface of a VI. Modeled from the front panel of physical instruments, it is composed of switches, slides, meters, graphs, charts, gauges, LEDs, and other controls and indicators.

function	Built-in execution element, comparable to an operator, function, or statement in a conventional language.
Functions palette	Palette containing block diagram structures, constants, communication features, and VIs.

G

G	The graphical programming language used to develop LabVIEW applications.
GPIB	General Purpose Interface Bus is the common name for the communications interface system defined in ANSI/IEEE Standard 488.1-1987 and ANSI/IEEE Standard 488.2-1987. Hewlett-Packard, the inventor of the bus, calls it the HP-IB.
global variable	Non-reentrant subVI with local memory that uses an uninitialized shift register to store data from one execution to the next. The memory of copies of these subVIs is shared and thus can be used to pass global data between them.

I

I/O	Input/output. Transfer of data to or from a computer system involving communications channels, operator input devices, and/or data acquisition and control interfaces.
indicator	Front panel object that displays output.
instrument driver	VI that controls a programmable instrument.
iteration terminal	Terminal of a For Loop or While Loop that contains the current number of completed iterations.

L

label	Text object used to name or describe other objects or regions on the front panel or block diagram.
list box	Box within a dialog box listing all available choices for a command. For example, a list of file names on a disk.
local variable	Variable that enables you to read or write to one of the controls or indicators on the front panel of your VI.

M

menu bar	Horizontal bar that lists the names of the main menus of an application. The menu bar appears below the title bar of a window. Each application has a menu bar that is distinct for that application, although some menus (and commands) are common to many applications.

N

node	Execution element of a block diagram, such as a function, structure, or subVI. *See also* data flow, wire.
numeric controls and indicators	Front panel objects used to manipulate and display or input and output numeric data.

P

palette	A display of icons that represent possible options. *See also* Controls palette, Functions palette, subpalette, Tools palette.
polymorphism	Ability of a node to adjust automatically to data of different representation, type, or structure.
pop up	To call a special menu by right-clicking an object.
pop-up menu	Menu accessed by right-clicking an object.

R

refnum	Identifier of a DDE conversation or open file that can be referenced by related VIs.
representation	Subtype of the numeric data type. Representations include signed and unsigned byte, word, and long integers, as well as single-, double-, and extended-precision floating-point numbers, both real and complex.

S

scalar	Number capable of being represented by a point on a scale. A single value as opposed to an array. Scalar Booleans and clusters are explicitly singular instances of their respective data types.
Sequence structure	Program control structure that executes its subdiagrams in numeric order.

shift register	Optional mechanism in loop structures used to pass the value of a variable from one iteration of a loop to a subsequent iteration.
structure	Program control element, such as a While Loop.
subpalette	A palette contained in an icon of another palette.
subVI	VI used in the block diagram of another VI; comparable to a subroutine.

T

terminal	Object or region on a node through which data passes.
Tools palette	Palette containing the tools you can use to edit and debug front panel and block diagram objects.
toolbar	Bar containing command buttons you can use to run and debug VIs.

V

VI	*See* virtual instrument.
VI library	Special file that contains a collection of related VIs for a specific use.
virtual instrument	A program in the graphical programming language G that models the appearance and function of a physical instrument.

W

waveform chart	An indicator that plots data points at a certain rate.
While loop	Loop structure that repeats a section of code until a condition is met. Comparable to a Do loop or a Repeat-Until loop in conventional programming languages.
wire	Data path between nodes. *See also* data flow.
Wiring tool	Tool used to define data paths between terminals. Resembles a spool of wire.