LabVIEW™ Connectivity Exercises

Course Software Version 2010
May 2011 Edition
Part Number 325628A-01

Copyright

© 2004–2011 National Instruments Corporation. All rights reserved.
Under the copyright laws, this publication may not be reproduced or transmitted in any form, electronic or mechanical, including photocopying, recording, storing in an information retrieval system, or translating, in whole or in part, without the prior written consent of National Instruments Corporation.

National Instruments respects the intellectual property of others, and we ask our users to do the same. NI software is protected by copyright and other intellectual property laws. Where NI software may be used to reproduce software or other materials belonging to others, you may use NI software only to reproduce materials that you may reproduce in accordance with the terms of any applicable license or other legal restriction.

For components used in USI (Xerces C++, ICU, HDF5, b64, Stingray, and STLport), the following copyright stipulations apply. For a listing of the conditions and disclaimers, refer to either the `USICopyrights.chm` or the *Copyrights* topic in your software.

Xerces C++. This product includes software that was developed by the Apache Software Foundation (`http://www.apache.org/`). Copyright 1999 The Apache Software Foundation. All rights reserved.

ICU. Copyright 1995–2009 International Business Machines Corporation and others. All rights reserved.

HDF5. NCSA HDF5 (Hierarchical Data Format 5) Software Library and Utilities
Copyright 1998, 1999, 2000, 2001, 2003 by the Board of Trustees of the University of Illinois. All rights reserved.

b64. Copyright © 2004–2006, Matthew Wilson and Synesis Software. All Rights Reserved.

Stingray. This software includes Stingray software developed by the Rogue Wave Software division of Quovadx, Inc.
Copyright 1995–2006, Quovadx, Inc. All Rights Reserved.

STLport. Copyright 1999–2003 Boris Fomitchev

Trademarks

LabVIEW, National Instruments, NI, ni.com, the National Instruments corporate logo, and the Eagle logo are trademarks of National Instruments Corporation. Refer to the *Trademark Information* at `ni.com/trademarks` for other National Instruments trademarks.

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

Members of the National Instruments Alliance Partner Program are business entities independent from National Instruments and have no agency, partnership, or joint-venture relationship with National Instruments.

Patents

For patents covering National Instruments products/technology, refer to the appropriate location: **Help»Patents** in your software, the `patents.txt` file on your media, or the *National Instruments Patent Notice* at `ni.com/patents`.

National Instruments
Not for Distribution

National Instruments
Not for Distribution

Worldwide Technical Support and Product Information

ni.com

Worldwide Offices

Visit ni.com/niglobal to access the branch office Web sites, which provide up-to-date contact information, support phone numbers, email addresses, and current events.

National Instruments Corporate Headquarters

11500 North Mopac Expressway Austin, Texas 78759-3504 USA Tel: 512 683 0100

For further support information, refer to the *Additional Information and Resources* appendix. To comment on National Instruments documentation, refer to the National Instruments Web site at ni.com/info and enter the Info Code feedback.

Contents

National Instruments Not for Distribution

National Instruments
Not for Distribution

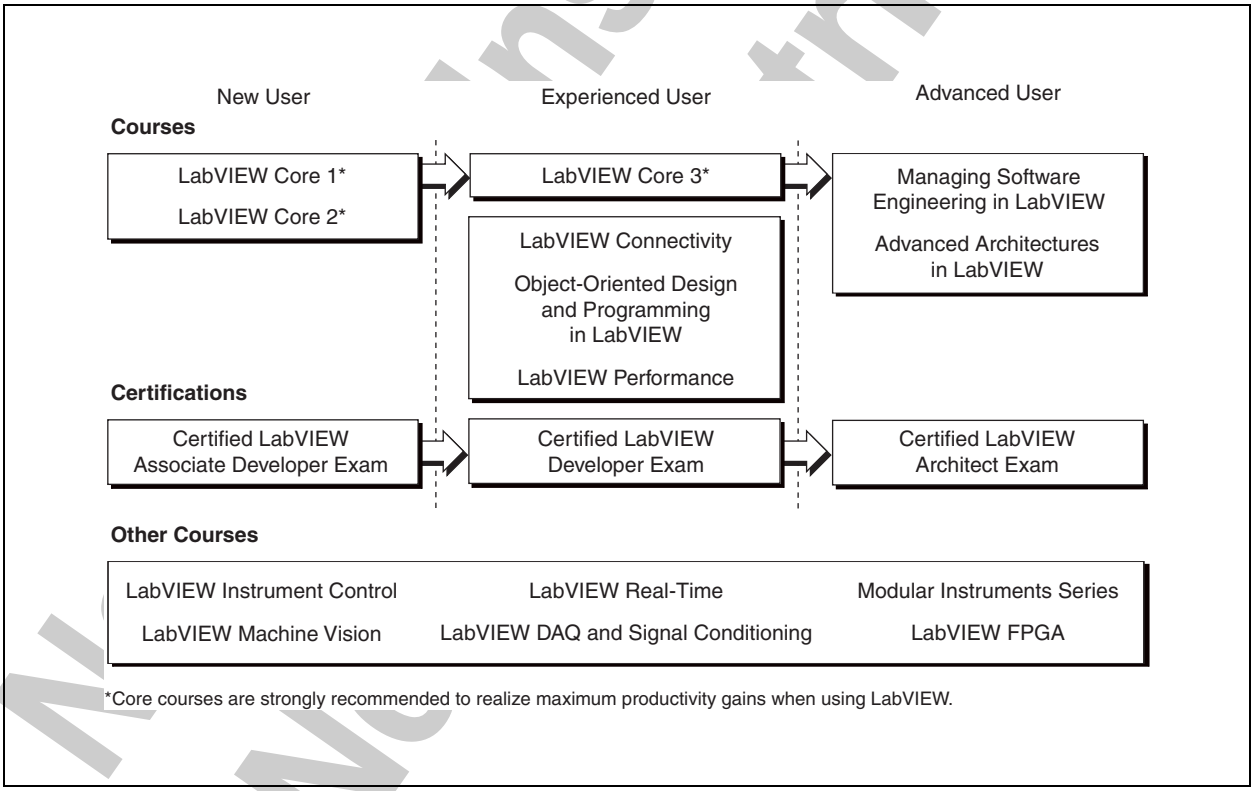
Student Guide

Thank you for purchasing the *LabVIEW Connectivity* course kit. This kit contains the materials used in the two-day, hands-on *LabVIEW Connectivity* course.

You can apply the full purchase price of this course kit toward the corresponding course registration fee if you register within 90 days of purchasing the kit. Visit `ni.com/training` to register for a course and to access course schedules, syllabi, and training center location information.

A. NI Certification

The *LabVIEW Connectivity* course is part of a series of courses designed to build your proficiency with LabVIEW and help you prepare for exams to become an NI Certified LabVIEW Developer and NI Certified LabVIEW Architect. The following illustration shows the courses that are part of the LabVIEW training series. Refer to `ni.com/training` for more information about NI Certification.



National Instruments Corporation

B. Course Description

The *LabVIEW Connectivity* course teaches you how to use advanced connectivity in VIs. This manual assumes you are familiar with Windows, that you have experience writing algorithms in the form of flowcharts or block diagrams, and that you have taken the *LabVIEW Core 1* and *LabVIEW Core 2* courses or you are familiar with all the concepts contained therein. This course also assumes that you have one year or more of LabVIEW development experience.

In the course manual, each lesson consists of the following sections:

- An introduction that describes the purpose of the lesson and what you will learn
- A discussion of the topics
- A summary or quiz that tests and reinforces important concepts and skills taught in the lesson

In the exercise manual, each lesson consists of the following sections:

- A set of exercises to reinforce topics
- Self-study and challenge exercise sections or additional exercises

Note For course manual updates and corrections, refer to `ni.com/info` and enter the Info Code `lvconn`.

C. What You Need to Get Started

Before you use this course manual, make sure you have the following items:

- ❑ Windows XP or later installed on your computer; this course is optimized for Windows XP
- ❑ LabVIEW Professional Development System 2010 or later
- ❑ Microsoft Excel
- ❑ *LabVIEW Connectivity* course CD, containing the following folders:

Directory	Description
`Exercises`	Contains all the VIs and support files needed to complete the exercises in this course
`Solutions`	Contains completed versions of the VIs you build in the exercises for this course

National Instruments
Not for Distribution

D. Installing the Course Software

Complete the following steps to install the course software.

1. Insert the course CD in your computer. The **LabVIEW Connectivity Course Material Setup** dialog box appears.
2. Click **Install LabVIEW Connectivity**.
3. Follow the onscreen instructions to complete installation and setup.

Exercise files are located in the `<Exercises>\LabVIEW Connectivity` folder.

Tip Folder names in angle brackets, such as `<Exercises>`, refer to folders in the root directory of your computer.

Repairing or Removing Course Material

You can repair or remove the course material using the **Add or Remove Programs** feature on the Windows **Control Panel**. Repair the course manual to overwrite existing course material with the original, unedited versions of the files. Remove the course material if you no longer need the files on your computer.

E. Course Goals

This course presents the following topics:

- Networking technologies
  - External procedure call model
  - Broadcast model
  - Client/server model
  - Publish/subscribe model
- Implementing the external procedure call model
  - Calling shared libraries from LabVIEW
  - Programmatically controlling VIs using the VI Server
    - Using the VI Server functions to programmatically load and operate VIs and LabVIEW itself
  - Using ActiveX objects in LabVIEW
    - Using LabVIEW as an ActiveX client
    - Using LabVIEW as an ActiveX server
    - ActiveX events

National Instruments
Not for Distribution

  - Using .NET Objects in LabVIEW
    - Using LabVIEW as a .NET client
    - .NET events
- Implementing the broadcast model
  - Using the UDP VI and functions to create a UDP multicast session
- Implementing the client/server model
  - Using the TCP/IP VI and functions to communicate with other applications locally and over a network
- Using LabVIEW Web services and HTTP Client VIs to create and deploy Web services

This course does not present any of the following topics:

- Basic principles of LabVIEW covered in the *LabVIEW Core 1* and *LabVIEW Core 2* courses
- Every built-in VI, function, or object; refer to the *LabVIEW Help* for more information about LabVIEW features not described in this course
- Developing a complete VI for any student in the class; refer to the NI Example Finder, available by selecting **Help»Find Examples**, for example VIs you can use and incorporate into VIs you create

F. Course Conventions

The following conventions are used in this course manual:

<> Angle brackets that contain numbers separated by an ellipsis represent a range of values associated with a bit or signal name—for example, AO <3..0>.

[ ] Square brackets enclose optional items—for example, `[response]`.

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

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

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

This icon denotes a caution, which advises you of precautions to take to avoid injury, data loss, or a system crash.

National Instruments
Not for Distribution

bold	Bold text denotes items that you must select or click 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. Italic text 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 enter from the keyboard, sections of code, programming examples, and syntax examples. This font also is used for the proper names of disk drives, paths, directories, programs, subprograms, subroutines, device names, functions, operations, variables, filenames, and extensions.
`monospace bold`	Bold text in this font denotes the messages and responses that the computer automatically prints to the screen. This font also emphasizes lines of code that are different from the other examples.
`monospace italic`	Italic text in this font denotes text that is a placeholder for a word or value that you must supply.
Platform	Text in this font denotes a specific platform and indicates that the text following it applies only to that platform.

National Instruments
Not for Distribution

National Instruments
Not for Distribution

1

Calling Shared Libraries in LabVIEW Exercises

Exercise 1-1 Computer Name

Goal

Call a DLL function from the Windows API.

Scenario

Programmatically determining the Windows computer name is necessary in a number of situations. For example, if you have several computers logging data to a central server, you should identify the computer name for each set of data in order to track the data. The computer name is also helpful for storing or retrieving data over a Windows File Share and is a useful piece of information to include when generating reports.

There is not a native function within LabVIEW to retrieve the computer name. However, the Windows operating system provides an API, in the form of DLLs, which allows you to interface with the operating system. One common use of this API is retrieving useful information about the operating system or computer, such as the computer name.

Call a Windows API DLL function to determine the name of the computer. You should also handle any errors returned by the function in an appropriate manner.

Note The Windows computer name is different from the network address of a computer. You can determine a computer's network address in LabVIEW by using the String to IP and IP to String functions located on the TCP/IP palette.

National Instruments
Not for Distribution

Design

Inputs and Outputs

Table 1-1. Computer Name Inputs and Outputs

Type	Name	Properties	Default Values
Numeric Control	Buffer Size	32-bit Unsigned Integer	256
String Indicator	Computer Name	—	Empty

Program Structure

When you call functions from a DLL, the DLL often defines a method for reporting errors. The Windows API returns a numeric value from each function call indicating whether the function was executed correctly. In the event of an incorrect execution, you can call additional Windows API functions to identify the error code and convert the code to an understandable string. To correctly identify the error, the error handling functions must be called immediately after the original function call, because the error information will be lost if any other Windows API calls are made from the same program. This error handling mechanism works differently than the normal error handling mechanism in LabVIEW, where each subVI reports its own errors and errors can be chained together and handled at the end of a program or section.

Get Computer Name

The Windows API provides a function called `GetComputerName` in `Kernel32.dll`. You must allocate a string buffer large enough to store the string returned by the function, fortunately, the Call Library Function Node in LabVIEW Version 8.20 or later can automatically allocate the buffer based on a size parameter passed to the function. If the buffer is too small to store the computer name, or if another error occurs, this function returns a value of zero. Use the return value from this function as well as the error out cluster from the Call Library Function Node to control the transition logic of the state machine.

Handle Errors

This program has the potential to generate normal LabVIEW errors, such as a missing DLL file in a Call Library Function Node, as well as Windows API errors, such as an insufficient buffer size for the `GetComputerName` function to store the computer name. The handle errors case should handle both types of errors.

National Instruments
Not for Distribution

Use the General Error Handler instead of the Simple Error Handler to allow for custom messages generated by the Windows API error reporting calls.

In order to provide a meaningful error message for Windows API calls, you must perform two steps. First, you must call the Windows API function `GetLastError`. This function returns a numeric value that represents any error encountered during the last function call, in this case, `GetComputerName`. In order to translate the numeric code into a meaningful message, you must call the `FormatMessage` Windows API function. You can use this function many different ways. Control the behavior of the function by setting one or more flags in a parameter called `dwFlags`. In order to return a message for a system error, set the `FORMAT_MESSAGE_FROM_SYSTEM` flag. Also set the `FORMAT_MESSAGE_IGNORE_INSERTS` flag because you do not need to include any additional parameters in your message. Setting this parameter also allows you to ignore the **arguments** parameter of the function, thereby simplifying the function call. For more information on setting flags, refer to the *Background: Windows API Reference* section.

Tip All of the functions called in this exercise are in `kernel32.dll`. However, Windows API functions are found in many other DLLs. Refer to the *Windows API Reference* to determine in which DLL a given function is located.

Additional Information

The following sections describe some issues particular to Windows API calls that you must address in order to implement a solution.

Background: Windows API Reference

You can find the Windows API definitions for each of the functions used in this exercise in the `<Exercises>\LabVIEW Connectivity\Computer Name` directory. Open each of the PDF files in this directory and browse the reference material in them. As you proceed through the exercise, refer to the references periodically to identify the meaning of each parameter you pass to the functions.

Note The function information in this manual is from the *Windows API Reference* in the *MSDN library.* You can access the full *Windows API Reference* at: `http://msdn.microsoft.com/en-us/library/Aa383749`.

Bitmasked Flags

Some functions in the API contain a flags parameter that allows you to enable one or more options by masking the bits in an integer. The **dwFlags** parameter of the `FormatMessage` function is an example of this technique.

National Instruments Not for Distribution

In order to set a flag in LabVIEW, pass an integer constant with the appropriate size and value to the Call Library Function Node. Remember that you can change the way a numeric constant is represented to make it easier to enter the flag value. For example, the specification for the `FormatMessage` function gives the flags in hexadecimal format. You can set the format of your numeric constant to hexadecimal and then enter the value directly.

If you need to set multiple flags in a flags parameter, you can use a Bitwise Or function to combine multiple flags. In LabVIEW, the Or function is a polymorphic function which automatically becomes a Bitwise Or if you wire two integers to it.

Data Types

Windows API and other DLL function specifications often refer to many data types which have different names in LabVIEW. Table 1-2 lists LabVIEW types for some of the Windows data types used in this exercise. For a more complete list, the Call DLL example in the NI Example Finder provides a complete data type conversion list, as well as examples for each data type. The Windows Data Type reference is also a useful reference. It can be found at: `http://msdn.microsoft.com/en-us/library/Aa383751`.

Table 1-2. Data Type Conversion

Windows Data Type	LabVIEW Data Type
LPTSTR	String (Pass as C String Pointer)
DWORD	U32
LPDWORD	U32 (Pass by pointer)
BOOL	I32
va_list*	Varies (Use Adapt to Type)
LPCVOID	Varies (Use Adapt to Type)

String Types

Many Windows API functions support two methods for representing strings. The first method is ASCII, in which each character in a string is represented by a single byte. Traditional ASCII has a 128 character set, which contains all of the upper and lowercase letters, as well as other common symbols and a set of control characters. LabVIEW typically uses ASCII to represent strings.

National Instruments Not for Distribution

The second method uses “wide” strings, which is another term for Unicode string representation (UTF-16 encoding). Unicode requires two bytes for each character, and allows for a much larger character set than traditional ASCII. Unicode is not natively supported in LabVIEW, but it is possible to use Unicode through add on libraries or calls to external functions.

Windows API functions which deal with strings often have two versions present in the DLL, marked with an `A` for ASCII and a `W` for wide. For example, there are two `GetComputerName` functions in `kernel32.dll`, `GetComputerNameA` and `GetComputerNameW`. In most cases, you should use the “A” version of the function in LabVIEW.

Implementation

1. Create the front panel shown in Figure 1-1.

Figure 1-1. Computer Name Front Panel

   - ❑ Create a blank VI and save the VI as `Computer Name.vi` in the `<Exercises>\LabVIEW Connectivity\Computer Name` directory.
   - ❑ Create the **Buffer Size** control as described in Table 1-1.
   - ❑ Create the **Computer Name** indicator as described in Table 1-1.

2. Call the `GetComputerName` function with the parameters shown in Table 1-3.

Table 1-3. GetComputerName Parameters

Parameter Name	Type	Format/Data type	Other
return type	Numeric	Signed 32-bit Integer	
lpBuffer	String	C String Pointer	Select **lpnSize** for **Minimum size**
lpnSize	Numeric	Unsigned 32-bit Integer	Select **Pointer to Value** for **Pass**

National Instruments Corporation
Warning Regarding Use of NI Software and NI Hardware
Not for Distribution

- ❑ Open the `GetComputerName` function reference from `<Exercises>\LabVIEW Connectivity\Computer Name\getcomputername.pdf` and identify the prototype and parameters for the function.
- ❑ Place a Call Library Function Node.

- ❑ Double-click the Call Library Function Node to open the **Call Library Function** dialog box.
- ❑ Click the **Function** tab and configure the settings to match Figure 1-2.
  - Click the **Browse** button and navigate to `\Windows\System32\kernel32.dll` or enter `kernel32.dll`.
  - Select **GetComputerNameA** from the **Function name** pull-down menu.
  - Select **Run in any thread** from the **Thread** section.

Note Because the Windows API functions are reentrant (multi-threaded), calling `GetComputerNameA` in UI thread functions correctly, except the error is not stored in the proper memory location for `GetLastError` to access it.

National Instruments
Not for Distribution

– Select **stdcall (WINAPI)** from the **Calling convention** section.

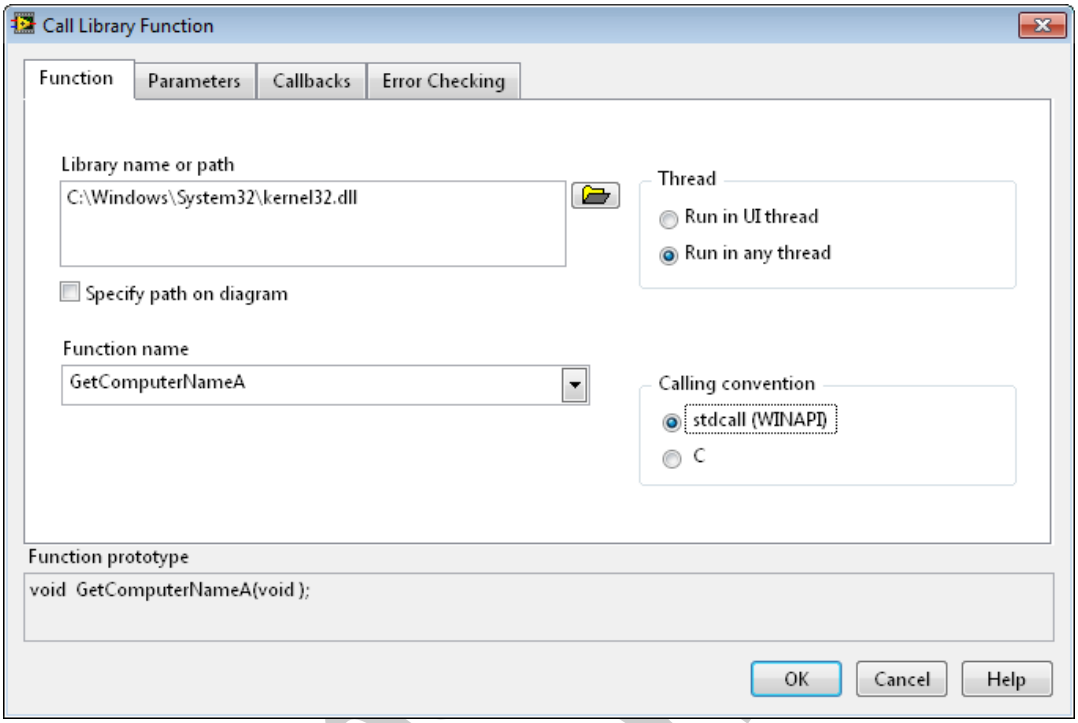


Figure 1-2. GetComputerNameA Call Library Function

❑ Click the **Parameters** tab.

– Ensure the **return type** parameter is selected.

– Select **Numeric** from the **Type** pull-down menu in the **Current parameter** section.

– Verify the **Data type** pull-down menu is set to **Signed 32-bit Integer**.

– Click the **+** button to add a parameter after the return type parameter.

– Enter `lpBuffer` in the **Name** text box.

– Select **String** from the **Type** pull-down menu.

– Verify the **String format** pull-down menu is set to **C String Pointer**.

Note You must declare an additional parameter before setting the minimum size for the lpBuffer parameter. Leave **Minimum size** blank.

National Instruments
Not for Distribution

- Click the + button to add a parameter.
- Enter `lpnSize` in the **Name** text box.
- Select **Numeric** from the **Type** pull-down menu.
- Select **Unsigned 32-bit Integer** from the **Data type** pull-down menu.
- Select **Pointer to Value** from the **Pass** pull-down menu.
- Select the **lpBuffer** parameter from the parameter list.
- Select **lpnSize** from the **Minimum size** pull-down menu.
- Confirm that the function prototype matches the following text.

```
int32_t GetComputerNameA(CStr lpBuffer,
uint32_t *lpnSize);
```

❑ Click the **OK** button.

3. Call the `GetLastError` function with the parameters shown in Table 1-4.

Table 1-4. GetLastError Parameters

Parameter Name	Type	Format/Data type	Other
return type	Numeric	Unsigned 32-bit Integer	—

❑ Open the `GetLastError` function reference from `<Exercises>\LabVIEW Connectivity\Computer Name\getlasterror.pdf` and identify the prototype and parameters for the function.

❑ Place another Call Library Function Node after the call to GetComputerName.

❑ Double-click the Call Library Function Node to open the **Call Library Function** dialog box.

❑ Click the **Function** tab.

- Click the **Browse** button and navigate to `\Windows\system32\kernel32.dll` or enter `kernel32.dll`.
- Select **GetLastError** from the **Function Name** pull-down menu.

National Instruments Not for Distribution

- Select **Run in any thread** from the **Thread** section.
- Select **stdcall (WINAPI)** from the **Calling convention** section.

❑ Click the **Parameters** tab.

- Ensure the return type parameter is selected.
- Select **Numeric** from the **Type** pull-down menu.
- Select **Unsigned 32-bit Integer** from the **Type** pull-down menu.
- Confirm that the function prototype matches the following text.

```
uint32_t GetLastError(void);
```

❑ Click **OK**.

4. Create the block diagram as shown in Figure 1-3 using the following items:

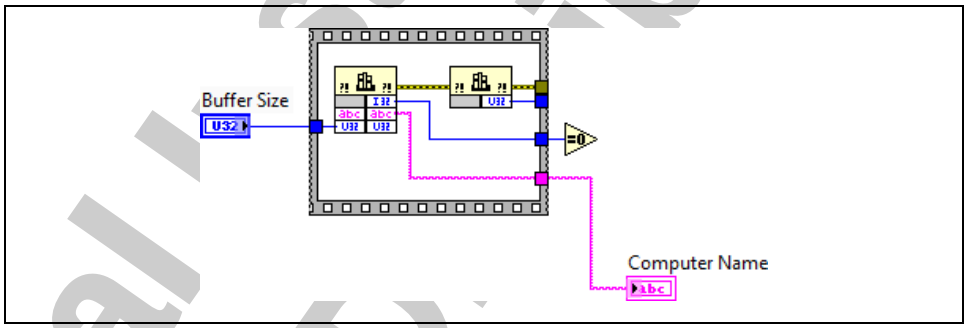


Figure 1-3. Get Computer Name

❑ Flat Sequence structure—Place around the two Call Library Function Nodes.

Note The Flat Sequence structure ensures that the VI checks for an error right after calling the `GetComputerNameA` function from the DLL and does so in the same thread.

Because of the way the LabVIEW execution system works, it is possible for something to run between the two DLL nodes. This could cause the GetLastError call to return an incorrect result. The Flat Sequence structure with nothing but the two DLL nodes in it reduces the likelihood that this would occur.

❑ Equal To 0?

❑ **Computer Name** indicator

❑ **Buffer Size** control

National Instruments Not for Distribution

5. Call the `FormatMessage` function with the parameters shown in Table 1-5.

Table 1-5. FormatMessage Parameters

Parameter Name	Type	Format/Data type	Other
return type	Numeric	Unsigned 32-bit Integer	—
dwFlags	Numeric	Unsigned 32-bit Integer	Select **Value** for **Pass**
lpSource	Adapt to Type	Pointers to Handles	—
dwMessageId	Numeric	Unsigned 32-bit Integer	Select **Value** for **Pass**
dwLanguageId	Numeric	Unsigned 32-bit Integer	Select **Value** for **Pass**
lpBuffer	String	C String Pointer	Select **nSize** for **Minimum size**
nSize	Numeric	Unsigned 32-bit Integer	Select **Value** for **Pass**
Arguments	Adapt to Type	Pointers to Handles	—

- ❑ Open the `FormatMessage` function reference from `<Exercises>\LabVIEW Connectivity\Computer Name\formatmessage.pdf` and identify the prototype and parameters for the function.
- ❑ Place a Call Library Function Node after the Flat Sequence structure.
- ❑ Double-click the Call Library Function Node to open the **Call Library Function** dialog box.
- ❑ Select the **Function** tab.
  - Click the **Browse** button and navigate to `\Windows\System32\kernel32.dll` or enter `kernel32.dll`.
  - Select **FormatMessageA** from the **Function Name** pull-down menu.
  - Select **Run in any thread** from the **Thread** section.
  - Select **stdcall (WINAPI)** from the **Calling convention** section.
- ❑ Click the **Parameters** tab.
  - Ensure the **return type** parameter is selected.

National Instruments
Not for Distribution

- Select **Numeric** from the **Type** pull-down menu.
- Select **Unsigned 32-bit Integer** from the **Type** pull-down menu.
- Click the **+** button to add a parameter.
- Enter `dwFlags` in the **Name** text box.
- Select **Numeric** from the **Type** pull-down menu.
- Select **Unsigned 32-bit Integer** from the **Data type** pull-down menu.
- Verify the **Pass** pull-down menu is set to **Value**.
- Click the **+** button to add a parameter.
- Enter `lpSource` in the **Name** text box.
- Select **Adapt to Type** from the **Type** pull-down menu.
- Select **Pointers to Handles** from the **Data Format** pull-down menu.
- Click the **+** button to add a parameter.
- Enter `dwMessageId` in the **Name** text box.
- Select **Numeric** from the **Type** pull-down menu.
- Select **Unsigned 32-bit Integer** from the **Data type** pull-down menu.
- Verify the **Pass** pull-down menu is set to **Value**.
- Click the **+** button to add a parameter.
- Enter `dwLanguageId` in the **Name** text box.
- Select **Numeric** from the **Type** pull-down menu.
- Select **Unsigned 32-bit Integer** from the **Data type** pull-down menu.
- Verify the **Pass** pull-down menu is set to **Value**.
- Click the **+** button to add a parameter.
- Enter `lpBuffer` in the **Name** text box.

National Instruments
Not for Distribution

- Select **String** from the **Type** pull-down menu.
- Verify the **String format** pull-down menu is set to **C String Pointer**.

Note You must declare an additional parameter before setting the Minimum Size pull-down menu. Leave Minimum Size set to **<None>** for now.

- Click the **+** button to add a parameter
- Enter `nSize` in the **Name** text box.
- Select **Numeric** from the **Type** pull-down menu.
- Select **Unsigned 32-bit Integer** from the **Data type** pull-down menu.
- Verify the **Pass** pull-down menu is set to **Value**.
- Click the **+** button to add a parameter
- Enter `Arguments` in the **Name** text box.
- Select **Adapt to Type** from the **Type** pull-down menu.
- Select **Pointers to Handles** from the **Data Format** pull-down menu.
- Select the **lpBuffer** parameter from the parameter list.
- Select **nSize** from the **Minimum size** pull-down menu.
- Confirm that the function prototype matches the following text.
- `uint32_t FormatMessageA(uint32_t dwFlags, void *lpSource, uint32_t dwMessageId, uint32_t dwLanguageId, CStr lpBuffer, uint32_t nSize, void *Arguments);`

Note The terminals corresponding to the arguments with type **Void** are blank until wired because void parameters accept any type of data.

❑ Click the **OK** button.

National Instruments Not for Distribution

6. Create the error handling code shown in Figure 1-4 using the following items.

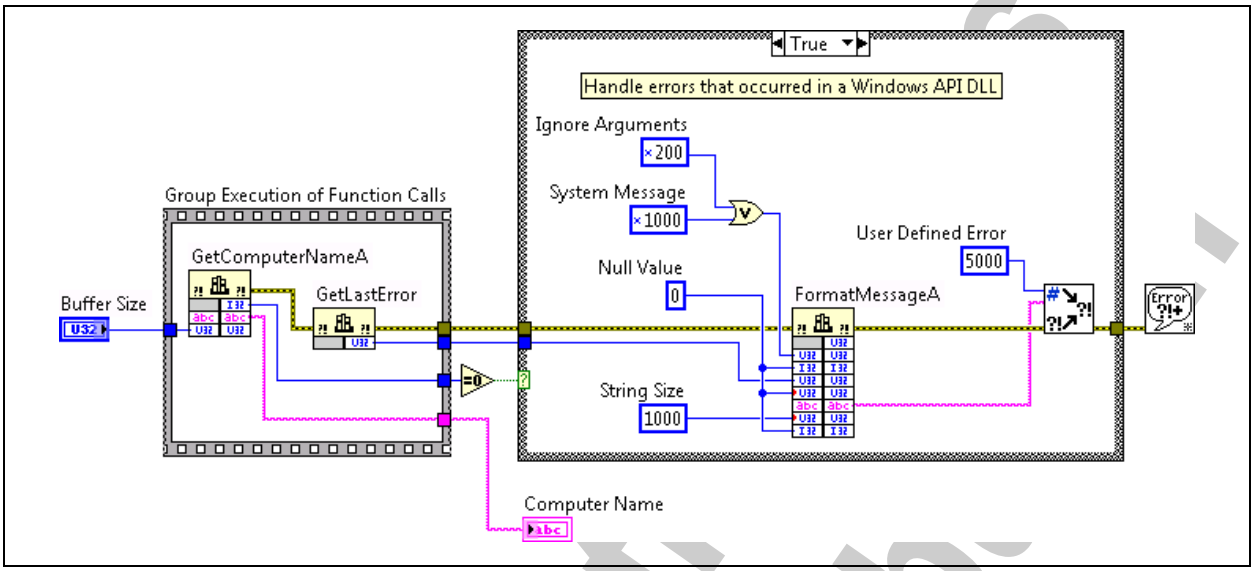


Figure 1-4. Error Handling Code

- ❑ Numeric constant
  - – Set the representation to U32.
  - – Right-click the numeric constant and select **Visible Items»Radix**.
  - – Click the radix and select **Hex**.
  - – Set the value of the constant to `200`.
  - – Label the constant `Ignore Arguments`.
- ❑ Numeric constant
  - – Create a copy of the **Ignore Arguments** constant.
  - – Change the label of the new constant to `System Message`.
  - – Set the value of the **System Message** constant to `1000`.
- ❑ Or
- ❑ Numeric constant
  - – Set the representation to I32.
  - – Label the constant `Null Value`.

National Instruments Not for Distribution

- ❑ Numeric constant
  - – Set the representation to U32.
  - – Label the constant `String Size`.
  - – Set the value of the **String Size** constant to `1000`.
- ❑ Error Cluster From Error Code VI
  - – Right-click the **error code** input terminal of the Error Cluster From Error Code VI and select **Create»Constant**.
  - – Set the value of the constant to `5000`.
  - – Label the constant `User Defined Error`.
- ❑ Place a Case structure around the error handling code.
- ❑ General Error Handler VI.

Testing

1. Test the VI with an appropriate buffer size.
   - ❑ Run the VI with the default **Buffer Size** (256).
   - ❑ The name of your computer should display in the **Computer Name** indicator.
2. Verify that the correct computer name displays.
   - ❑ Locate **My Computer** on the desktop of your computer or in Windows Explorer.
   - ❑ Right-click **My Computer** and select **Properties** from the shortcut menu.
   - ❑ Select the **Computer Name** tab and verify that the **Full Computer Name** matches the value the VI returns.

Note The case of the names does not need to match.

3. Test the error handling in the VI.
   - ❑ Set the **Buffer Size** control to `1` and run the VI.
   - ❑ Verify that the VI displays **`The file name is too long.`** as an error message.

National Instruments Not for Distribution

Tip The error message that displays for this VI is the description for the Windows error message `ERROR_BUFFER_OVERFLOW (System Error 111)`. Refer to `http://msdn.microsoft.com/library/en-us/debug/base/system_error_codes.asp` for more information about system error codes and their descriptions.

End of Exercise 1-1

National Instruments
Not for Distribution

Notes

National Instruments
Not for Distribution

2

VI Server Exercises

Exercise 2-1 VI Server Options

Goal

Observe and set the VI server configuration options.

Description

VI Server presents a potential security risk because other programs and/or computers can use it to call VIs, which can in turn be used to do almost anything with a computer. To protect your computer, VI server contains security options which allow you to select who can use VI server, which VIs users can access, how the VIs can be used, and what communication mechanisms can be used.

This exercise demonstrates the VI Server configuration options and sets them to a configuration you can use to run the remaining exercises in the course. The security configuration set in this exercise is very light. Therefore, if you are completing these exercises on a development machine or any other important machine, increase the security level by only allowing certain machines to access VI Server or return the settings to the default values when you finish the course.

Implementation

1. Configure VI Server.
   - ❑ Select **Tools»Options** and select **VI Server** from the **Category** list.
   - ❑ In the Protocols section, select **TCP/IP**. Note the **Port** number.
   - ❑ Verify **ActiveX** is selected.
   - ❑ Verify that all options under **Accessible Server Resources** are selected.
2. Configure Machine Access.
   - ❑ In the Machine Access section, enter * in the **Machine name/address** field.

National Instruments
Not for Distribution

Tip Entering * in the **Machine name/address** field opens VI Server Access to all computers. For security reasons, you should not do this on a production computer. Instead, add the machine name or IP address of each computer that needs access to VI server on this computer.

3. Configure Exported VIs.

   ❑ In the **Exported VIs** section**,** verify * is entered in the **Exported VIs** list.

   ❑ Click the **OK** button to exit the **Options** dialog box.

End of Exercise 2-1

National Instruments
Not for Distribution

Exercise 2-2 VI Statistics

Goal

Open and use VI server references to Application and VI objects to gather information about all open VIs.

Scenario

The Application VI Server object allows you to retrieve a list of all VIs in memory. Using this list, you can determine information about the VIs in memory. This information is useful for writing tools which track information about the VIs on your system. In this exercise you track VI usage statistics to identify the number of VIs running and the number of VIs in memory at any given time. Because VI Server has the ability to access application instances on remote machines, you can use this program to track the VI usage on any computer which allows you VI Server access.

Design

Inputs and Outputs

Table 2-1. VI Statistics Inputs and Outputs

Type	Name	Properties	Default Value
String Control	Machine Name	String	Empty (defaults to local machine)
Numeric Indicator	VIs Open	Signed 32-bit Integer	0
Numeric Indicator	VIs Running	Signed 32-bit Integer	0
Table	VI Report	Table, Column headers visible	Empty (Column headers only)

Program Flow

1. Acquire a reference to the LabVIEW Application object by using the Open Application Reference function.
2. Use this reference to access the ExportedVIs property, which gives you a list of each VI in memory.
3. Use a For Loop and the Open VI Reference function to get a reference to each VI in the list.
4. Using the VI reference, access the desired properties, in this case, Name, VIType and Exec.State.

National Instruments Not for Distribution

5. Close each VI reference.

6. Gather and display the data.

7. Close the application reference.

Property Descriptions

Use the following properties in this program:

ExportedVIs (Application Object)—This property returns an array of strings, which represent the name of all VIs in memory, if run on the local machine, or a list of all exported VIs in memory, if run on a remote machine. You must use this property instead of the AllVIs property to run the program on a remote machine. Notice that the strings contain only the names of the VIs, and not their paths. However, because the VIs are already guaranteed to be in memory, you only need the VI name to open a VI reference.

Name (VI Object)—This property accesses the name of the VI. You could use the names from the ExportedVIs property in place of this property. However, using the property provides a consistent technique for accessing the VI data and also simplifies wiring.

VIType (VI Object)—This property returns an enumeration containing the type of the VI. This property is useful because not all VIs are standard, executable VIs. Examples of other types of VIs include global variables, type definitions, and custom controls. In this program, this property provides information for the VI report. Certain VI properties are valid for only some VI types, and therefore, it may be necessary to check the value of this property before accessing it. For example, if the program uses any properties from the Execution group other than Exec.State, you would need to check this property before accessing the properties to ensure that the current VI reference is not a control or a global variable.

Exec.State (VI Object)—This property returns an enumeration containing the execution state of the VI. In this program, you increment the number of running VIs if this property is equal to Run top level or Running.

National Instruments
Not for Distribution

Implementation

1. Create a blank VI and save the VI as `VI Statistics.vi` in the `<Exercises>\LabVIEW Connectivity\VI Statistics` directory.

2. Create the front panel as shown in Figure 2-1.

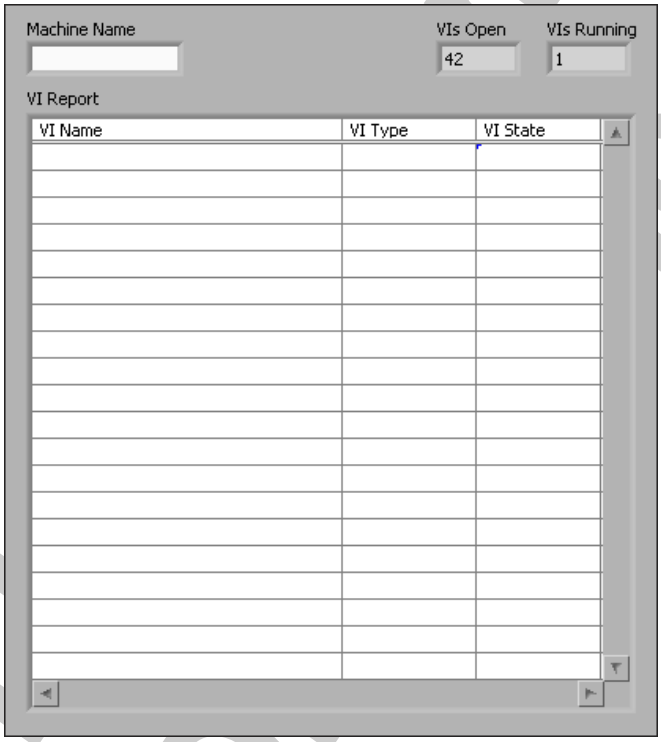


Figure 2-1. VI Statistics Front Panel

- ❑ Create the following items as described in Table 2-1.
  - – **Machine Name** control
  - – **VIs Open** indicator
  - – **VIs Running** indicator
- ❑ Place a Table control on the front panel.
  - – Label the table `VI Report`.
  - – Right-click the table and select **Change to Indicator** from the shortcut menu.
  - – Right-click the table and select **Visible Items»Column Headers** from the shortcut menu.

National Instruments Not for Distribution

– Enter `VI Name`, `VI Type`, and `VI State` as the first three column headers.

3. Acquire a reference to the LabVIEW Application object and access the ExportedVIs property. Create the block diagram as shown in Figure 2-2 using the following items.

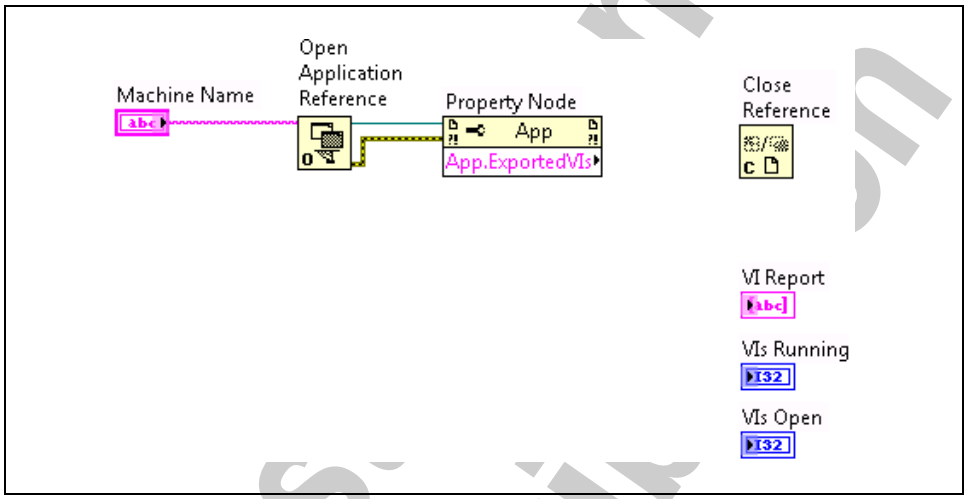


Figure 2-2. Application Properties

- ❑ Open Application Reference
- ❑ ExportedVIs Property Node—Right-click the **application reference** output of the **Open Application Reference** function and select **Create»Property for Application Class»Application» Exported VIs In Memory** from the shortcut menu.
- ❑ Close Reference

Tip Leave space between the Property Node and the Close Reference function so you can insert more code between them in later steps.

National Instruments Not for Distribution

4. Acquire a VI reference to each exported VI. Use the following items to modify the block diagram as shown in Figure 2-3.

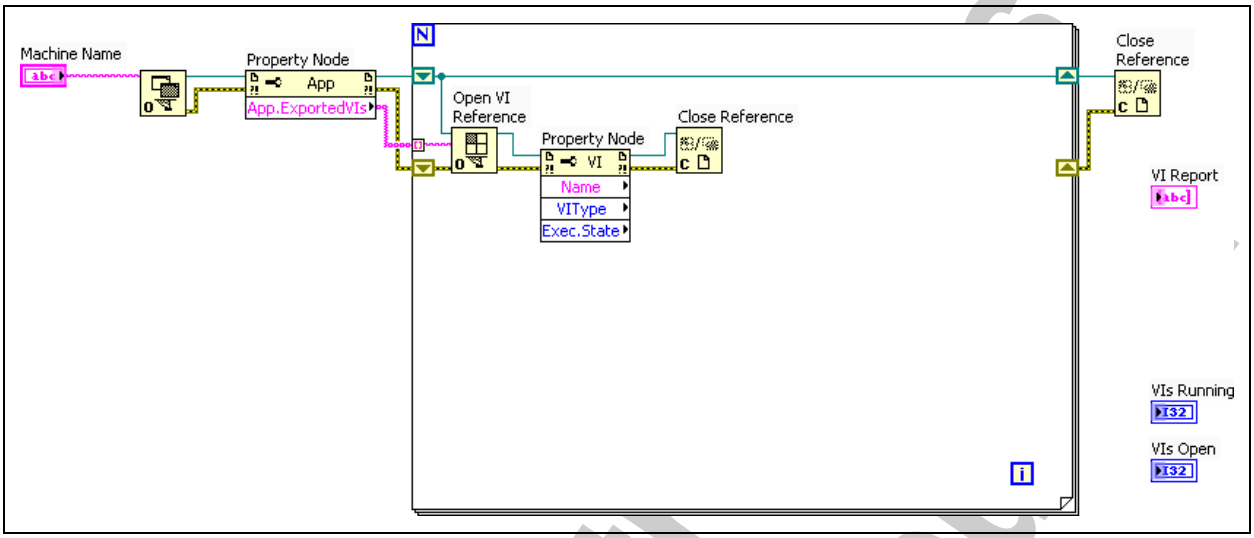


Figure 2-3. VI Properties

- ❑ For Loop
- ❑ Open VI Reference
- ❑ VI Name Property Node
  - – Right-click the **vi reference** output of the Open VI Reference function and select **Create»Property for VI Class»VI Name** from the shortcut menu to create the VI Name Property Node.
  - – Expand the VI Name Property Node so that three items are available.
  - – Click the second item in the Property Node and select **VI Type** from the list.
  - – Click the third item in the Property Node and select **Execution» State** from the list.
- ❑ Close Reference
- ❑ Shift Registers—Replace the application reference and error tunnels on the For Loop with Shift Registers.

National Instruments
Not for Distribution

5. Figure 2-4 shows the items and wiring you add in steps 5 and 6. In this step, add the following items to gather and display VI statistics.

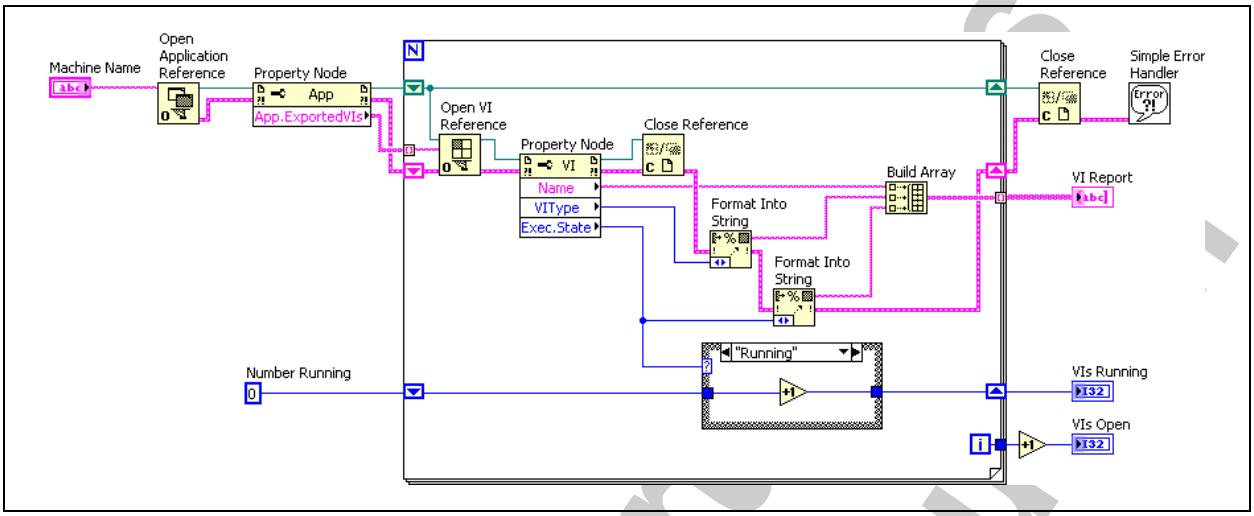


Figure 2-4. VI Statistics Block Diagram

- ❑ Numeric constant
  - – Set the representation to I32.
  - – Label the constant `Number Running`.
- ❑ Increment function—Wire the value from the **Number Running** constant through the Increment function to the **VIs Running** indicator. Use shift registers to wire through the For Loop.
- ❑ Case structure
  - – Place the Case structure around the Increment function.
  - – Wire the output of the Exec.State property to the case selector terminal.
  - – Right-click the Case structure and select **Add Case After** from the shortcut menu.
  - – Verify that the Run top level case of the Case structure is visible. Right-click the Case structure and select **Swap Diagram With Case»Idle** from the shortcut menu.
  - – Wire the numeric data through the Idle and Bad cases of the Case structure.
  - – Switch to the Run Top Level case, which should have the increment function in it, then right-click the Case structure and

National Instruments Not for Distribution

select **Duplicate Case** from the shortcut menu. This creates a Running case which contains an increment function.

The Case structure increments the number in the **VIs Running** indicator if the VI is in the Run Top Level or the Running state.

- ❑ Increment function
  - – Place the function to the right of the For Loop.
  - – Wire the iteration terminal of the For Loop to the second Increment function through the border of the For Loop. Disable indexing on the tunnel.
- ❑ Two Format Into String functions—The Format Into String function determines the string representation of an enumerated value.
- ❑ Build Array

6. In this step, add the following item to handle errors.
   - ❑ Simple Error Handler VI
7. Save the VI.

Testing

1. Run the VI.
   - ❑ Close all other open VIs.
   - ❑ Run the VI Statistics VI.
2. Run the VI with multiple VIs in memory.
   - ❑ Open the solution to the Word Processor project in the `<Solutions>\LabVIEW Connectivity\Exercise 3-2` directory.

Note If you have the LabVIEW Core 3 files installed, you also can use `<Solutions>\LabVIEW Core 3\Course Project\Exercise 7-8\TLC Main.vi`.

- ❑ Run the VI Statistics VI.
- ❑ Observe the results in the **VI Report**.

National Instruments Not for Distribution

3. Test the VI on a remote system.

   ❑ Determine the network address of a target computer near you and enter it in the **Machine Name** control.

Tip In most cases, you can use the Computer Name, which you found in Exercise 1-1, as a network address. If this name does not work, find the IP address of the computer by using the String to IP and IP to String functions.

   ❑ Verify that the VI Server settings on the target computer are configured as described in Exercise 2-1.

   ❑ Open one or more VIs on the target computer.

   ❑ Run the VI. All exported VIs in memory on the target computer should be displayed.

Challenge

Add statistical information for the VI priority, VI execution system, and/or state of the front panel to your table. Remember that not all properties are valid for all types of VIs. Use the context help to identify which types of VIs a property applies to and the VIType property to determine which VIs have the property, otherwise you receive an error.

End of Exercise 2-2

National Instruments
Not for Distribution

Exercise 2-3 Remote Run VI

Goal

Build a VI that programmatically opens and runs another VI on a remote computer.

Description

You have seen how the Application refnum runs transparently over a network. In this exercise, use VI Server to run a VI remotely. The techniques in this exercise show how to use VI Server to open and run a VI on a remote machine. VI Server operates the same whether it is on a network or a local machine.

Implementation

1. Open the Remote Run VI located in the `<Exercises>\LabVIEW Connectivity\Remote Run` directory. The front panel is built for you.

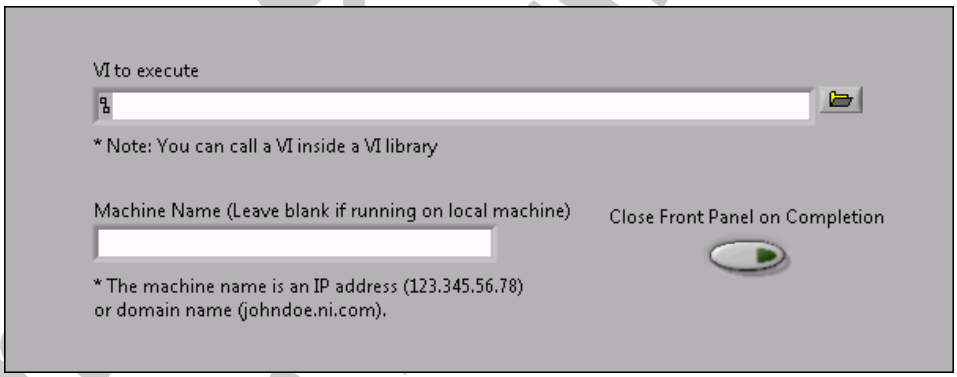


Figure 2-5. Remote Run VI Front Panel

2. In the **VI to execute** control, browse to `<Exercises>\LabVIEW Connectivity\Remote Run\Statistics.vi`. Right-click the control and select **Data Operations»Make Current Value Default** from the shortcut menu.

National Instruments Not for distribution

3. Build the block diagram shown in Figure 2-6 using the following items.

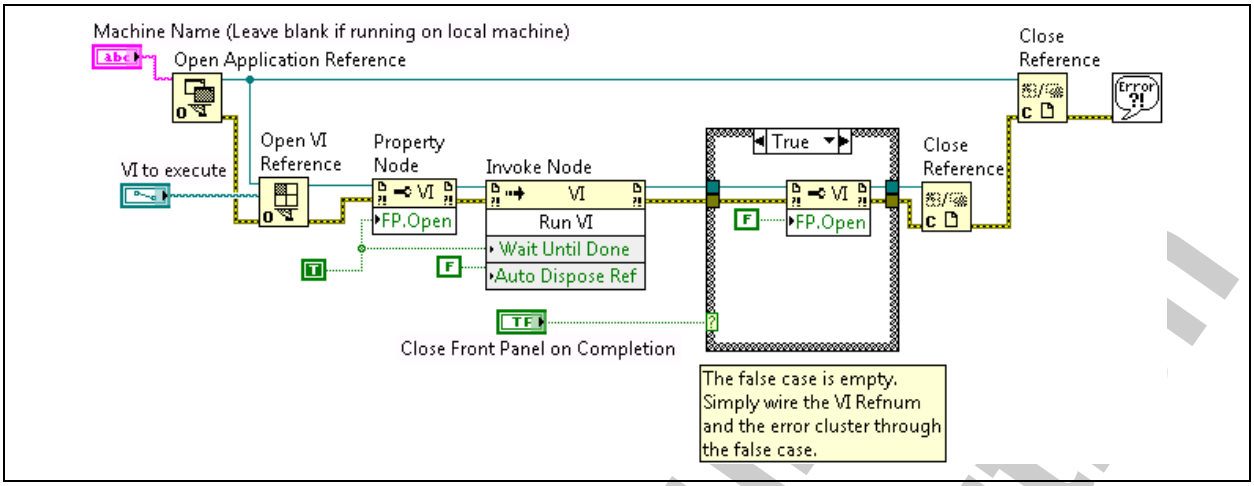


Figure 2-6. Remote Run VI Block Diagram

- ❑ Open Application Reference

- ❑ Open VI Reference
- ❑ Wire the **VI to execute** path control, which determines the VI to execute, to the **vi path** input of the Open VI Reference function.

- ❑ Two Close References

- ❑ Property Node
  - Wire the **vi reference** output of the Open VI Reference function to the **reference** input of the Property Node.
  - Click the **Property** terminal and select **Front Panel Window» Open**.
  - Right-click the Property Node and select **Change All to Write** from the shortcut menu.
  - Wire a TRUE Boolean constant to the **Front Panel Window Open** property terminal.
  - Create a copy of this Property Node.

National Instruments
Not for Distribution

- ❑ Invoke Node
  - – Wire the VI reference from the Property Node to the Invoke Node.
  - – Click the method terminal and select **Run VI** from the list.
  - – Wire the TRUE Boolean constant to the Wait Until Done property terminal and a FALSE Boolean constant to the Auto Dispose Ref property terminal.

- ❑ Simple Error Handler VI
- ❑ Case structure—Use the Case structure to select whether the front panel of the called VI remains open when the VI completes execution.
  - – Place the Case structure around the second Property Node.
  - – Wire a FALSE Boolean constant to the Front Panel Window Open property of the Property Node. Verify these items are in the True case. This case closes the front panel of the called VI if it is selected.
  - – Wire the VI refnum and the error cluster through the False case.
  - – Wire the case selector to the **Close Front Panel on Completion** control.

4. Save the VI.

Testing

Run the VI on the local computer.

This VI opens a reference to the Frequency Response VI located in the `<Exercises>\LabVIEW Connectivity\Remote Run` directory. The front panel of the VI is opened by accessing the Front Panel Open property of the VI. Then, the Run VI method runs the VI. Because the Wait Until Done property is TRUE, this VI waits for the Frequency Response VI to complete execution. After exiting the Frequency Response VI, the front panel remains open or closes depending on the position of the front panel switch. Finally, the Close Reference function closes the VI reference, freeing the resources.

If time permits, complete the following Optional and Challenge steps, otherwise close the VI.

National Instruments Not for Distribution

Optional

If your computer is connected through TCP/IP to another computer that has LabVIEW and each computer has a unique IP address, you can run the Remote Run VI on one computer and have it call the Frequency Response VI on the other computer.

1. Find a partner and exchange IP addresses. Decide which computer is the server. Complete the following steps on the server computer to set up the VI Server.

   - ❑ Select **Tools»Options** and select **VI Server** from the **Category** list to display the **VI Server** page. Verify that **TCP/IP** is selected and that a port number is entered.

   - ❑ In the **Machine Access** section, enter the IP address of the client computer. Select **Allow Access** and click **Add**.

   - ❑ In the **Exported VIs** section, confirm that a wildcard (*) is allowed access. This allows the client computer, or any computer allowed access in the **VI Server: Machine Access** section, to access any VIs on your computer. Click the **OK** button.

2. On the client computer, verify the path to the Frequency Response VI on the server computer. Enter the IP address of the server computer in the **Machine Name** control.

3. Run the Remote Run VI on the client computer. Does the VI behave as expected? Repeat steps 1 and 2, but reverse situations with your partner.

Challenge

Break into groups of three. Write a VI on the first computer that calls the Remote Run VI on the second computer, which then calls the Frequency Response VI on the third computer.

End of Exercise 2-3

National Instruments
Not for Distribution

Exercise 2-4 Dynamically Calling VIs

Goal

Observe two different methods for calling VIs dynamically and learn the difference between strictly and weakly typed refnums.

Description

This exercise demonstrates two ways to dynamically call a VI. The first method is to use a Weakly Typed VI Refnum. This technique is advantageous because it can be used to call any VI, regardless of the VI's connector pane. However, passing data to the VI using a Weakly Typed VI Refnum is difficult.

The second method uses a strictly typed VI refnum. The strictly typed refnum specifies the connector pane for the called VI, and allows you to use a Call By Reference Node, which simplifies the passing of data to the dynamically called VI. However, a Strictly Typed VI Refnum only allows you to call VIs with a matching connector pane. Therefore it is not as flexible as the weakly typed VI refnum.

Instructions

1. Open the Dynamically Calling VIs VI located in the `<Exercises>\LabVIEW Connectivity\Dynamically Calling VIs` directory.
2. Complete the VI front panel as shown in Figure 2-7 using the following items.
   - ❑ Place a VI Refnum to the left of the Boolean switch.
     - Label the refnum `Weakly Typed`.
     - Right-click the refnum and choose **Select VI Server Class**. Verify that **VI** is checked.
   - ❑ Place a VI Refnum to the right of the Boolean switch.
     - Label the refnum `Strictly Typed`.
     - Right-click the refnum and choose **Select VI Server Class» Browse**. Navigate to `<Exercises>\LabVIEW Connectivity\Dynamically Calling VIs` directory and select the Pop up VI. Click the **OK** button. The refnum adapts to the connector pane of the Pop up VI.

National Instruments Not for Distribution

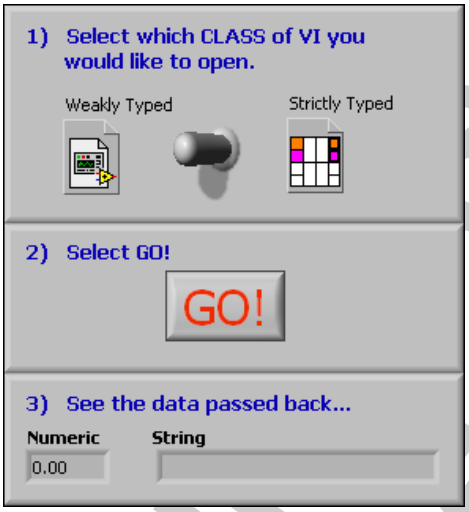


Figure 2-7. Weakly versus Strictly Typed Ref VI Front Panel

3. Complete the False case as shown in Figure 2-8.

   Wire the **Weakly Typed** VI refnum to the **type specifier** input of the Open VI Reference function in the False case as shown in Figure 2-8.

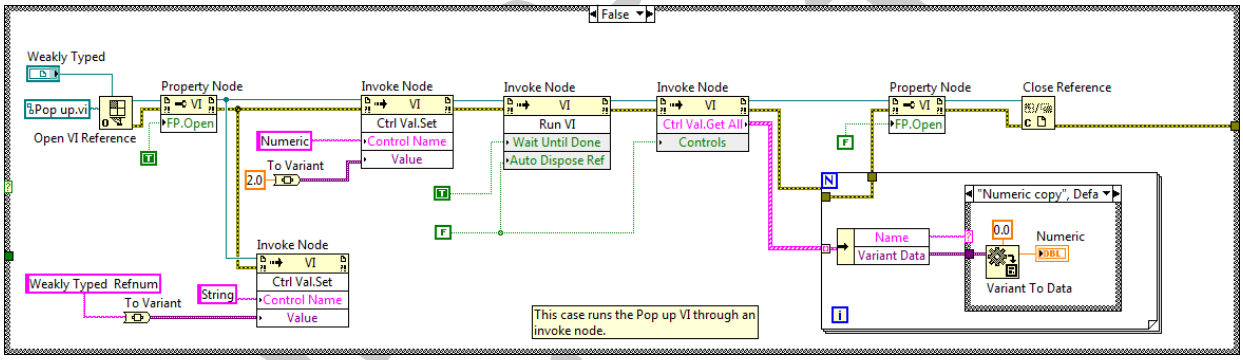


Figure 2-8. Weakly VS Strictly Typed Ref VI Block Diagram False Case

The False case contains a VI reference to the Pop up VI. This VI reference opens the front panel of the VI using the Front Panel Window:Open property. The Set Control Value method passes values to the **Numeric** and **String** controls of the Pop up VI.

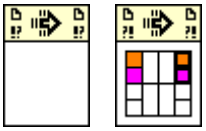
The Run VI method runs the VI and waits until it completes execution. The Get All Control Values method returns the values of the front panel indicators of the Pop up VI. These values display on the front panel of this VI. Finally, the Close VI Reference function closes the front panel of the Pop up VI and releases the VI Reference.

National Instruments Not for Distribution

4. Complete the True case on the block diagram as shown in Figure 2-9.

   ❑ Wire the **Strictly Typed** VI refnum to the **Type Specifier** input of the Open VI Reference function.

   ❑ Place a Call By Reference Node on the block diagram.



5. Wire the **VI reference** output of the Open VI Reference function to the **reference** input of the Call By Reference Node. The node adopts the connector pane of the Pop up VI, as shown at left.

Note When you wire the strictly typed VI refnum for the Pop up VI to the Open VI Reference function, a strictly typed VI reference is generated that you can wire to the Call By Reference Node.

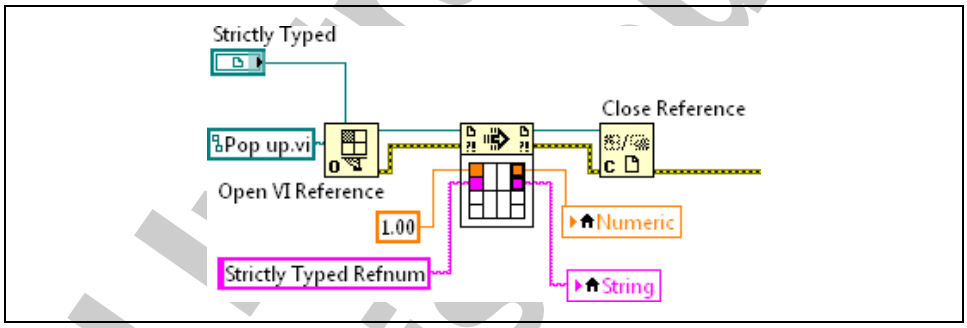


Figure 2-9. Block Diagram Code Inside the True Case

6. Save the VI.

Testing

1. Run the VI.

2. Select the strictly typed reference and click the **GO!** button.

   The Pop up VI appears. It returns the value it receives or allows you to change the data. When you finish with the Pop up VI and click the **DONE** button. The front panel of the Dynamically Calling VIs VI shows the values of the indicators from the Pop up VI.

3. Run the VI again and select the weakly typed reference. Notice that the behavior is the same as the behavior of the strictly typed reference.

   Although both calling methods produce the same result, the Run VI method provides more flexibility and allows you to call a VI asynchronously. If you call a VI asynchronously by passing a false value to the **Wait Until Done** parameter of the Run VI method, the

National Instruments Not for Distribution

dynamically called VI executes independently of the calling VI. The calling VI continues its dataflow progression without waiting for the called VI to complete.

The Call By Reference Node simplifies calling a VI dynamically, particularly when passing data to the subVI. The Call By Reference Node requires a strictly typed reference that eliminates the possibility of a run-time type mismatch error. If you do not need the additional flexibility of the Run VI method, use the Call By Reference Node to reduce the complexity of your code.

End of Exercise 2-4

National Instruments
Not for Distribution

Notes

National Instruments
Not for Distribution

Notes

National Instruments
Not for Distribution

3

Using .Net and ActiveX Objects in LabVIEW Exercises

Exercise 3-1 Font Dialog

Goal

Call a System .NET Assembly to display a Windows Common Dialog box.

Scenario

For many applications, you want to provide a familiar look and feel for your user. One technique for doing this is reusing Windows Common Dialogs whenever possible. For example, to have your user select a text font, you can call a Font dialog from the operating system. The Windows Common Font Dialog Box creates a font style and color selection dialog that is familiar to users of most Windows-based word processors, as shown in Figure 3-1.

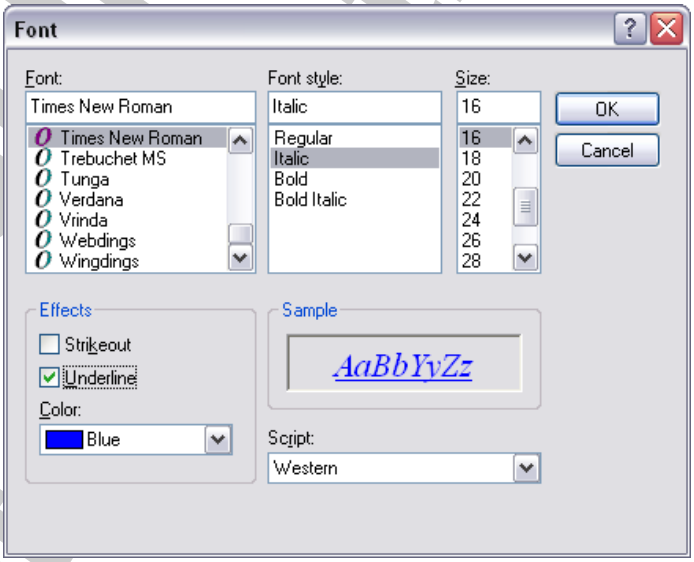


Figure 3-1. Windows Common Font Dialog Box

Create a subVI that calls a font dialog and returns references to the font and color selected by the user. The subVI should use proper error handling techniques and should return a value indicating if the user has canceled the dialog.

National Instruments Not for distribution

Design

There are multiple ways to display Windows Common Dialogs. The Microsoft Common Dialog Control ActiveX server and ActiveX control provide access to the Color, Font, Help, Open, Printer, and Save dialog boxes. However, these ActiveX components require special licensing to use, which can be acquired through Microsoft Visual Studio. Alternately, you can create a FontDialog object from the System.Windows.Forms .NET assembly. Using the .NET assembly requires the .NET Framework to be installed, but does not require any additional licensing. Furthermore, the .NET assembly provides the newest version of the dialog, which has additional features and improved integration with other Windows components.

FontDialog Inputs and Outputs

Table 3-1. FontDialog Inputs and Outputs

Type	Name	Properties	Default Value
Cluster Control	error in	Error Cluster	No Error
Constructor Node	Font	.NET Reference, System.Drawing.Font class	Not a refnum
Constructor Node	Color	.NET Reference, System.Drawing.Color class	Not a refnum
Enum Indicator	DialogResult	Values defined by FontDialog object	None
Cluster Indicator	error out	Error Cluster	No Error

The FontDialog subVI should perform the following steps:

1. Create a FontDialog object using a .NET Constructor Node.
2. Set the ShowColor property to TRUE so that the font dialog allows the user to select a color.
3. Call the ShowDialog method to show the dialog and return a result. Return the result to the calling VI.
4. Use the Font and Color properties to obtain references to the selected font and color and return these references to the calling VI.
5. Close the reference to the FontDialog object.

National Instruments Not for Distribution

FontDialog Object Description

The FontDialog object creates a common dialog box that displays a list of fonts that are currently installed on the system. You can create a FontDialog class by selecting the FontDialog object from the **System.Windows.Forms** assembly using a .NET Constructor Node. You use the following properties and methods of the FontDialog object in this exercise. Full documentation for the FontDialog and other objects included in the .NET Framework can be found on MSDN or in the documentation for Microsoft Visual Studio .NET.

ShowColor Property—Setting this to TRUE instructs the FontDialog to show a selector for the font color. This property should be set before showing the dialog.

ShowDialog Method—Displays the font dialog. This method returns an enumerated type which indicates the user's response to the dialog. Notice that the enumerated type returned from this function is shared among many dialogs, and therefore not all the values are actually possible from a FontDialog. A FontDialog typically returns OK or Cancel.

Two versions of the ShowDialog method exists. One takes no parameters and the other takes an IWin32Window object to designate the owning window for the dialog. For this exercise, use the version of the function with no parameters. This may occasionally cause the font dialog to show up behind the main application. You can solve this problem by using the IWin32Window version of the function. However, this requires getting a reference to the window handle of the LabVIEW front panel and converting it to an IWin32Window object, which is beyond the scope of this exercise.

Font Property—Returns a .NET reference to a System.Drawing.Font object. You can use this reference to get information about the font, such as the font name and size, or you can pass this reference to other objects that take .NET Font references, such as a .NET RichTextBox control.

Color Property—Returns a .NET reference to a System.Drawing.Color object. You can convert this color into a LabVIEW color by using the reference to get the **R**, **G** and **B** properties of the color and then using the RGB to Color VI. Alternately, you can pass this reference to any .NET object which uses colors.

National Instruments
Not for Distribution

Implementation

1. Create the VI.

   ❑ Create a blank VI and save it as `Font Dialog.vi` in the `<Exercises>\LabVIEW Connectivity\Font Dialog` directory.

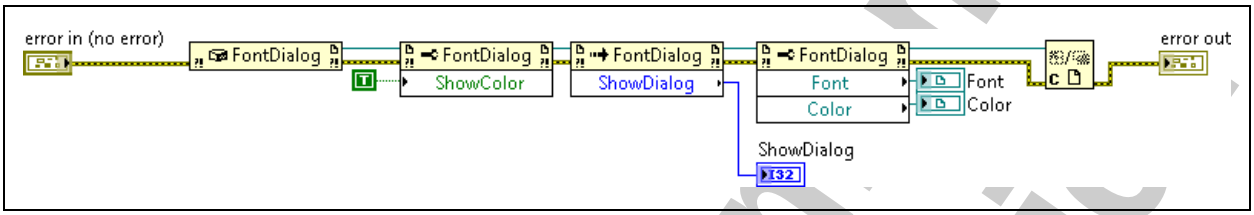


Figure 3-2. Completed Font Dialog Block Diagram

2. Add the following item to the block diagram as shown in Figure 3-2 to open a .NET reference to the FontDialog object.

   ❑ Place a Constructor Node on the block diagram to display the **Select .NET Constructor** dialog box.

   ❑ Select **System.Windows.Forms** from the **Assembly** pull-down menu.

Note If more than one version of **System.Windows.Forms** is listed, select the latest one.

   – Double-click the + to the left of the **System.Windows.Forms** item in the **Objects** list. Scroll down and select **FontDialog** to add it to the **Constructors** list.

   – Click **OK**.

3. Add the following items to the block diagram as shown in Figure 3-2 to show the font dialog.

   ❑ FontDialog Property Node.

   – Right-click the **new reference** output of the Constructor Node and select **Create»Property for System.Windows.Forms.FontDialog Class»ShowColor** to create the FontDialog Property Node.

   – Right-click the FontDialog Property Node and select **Change All To Write** from the shortcut menu.

National Instruments Not for Distribution

- ❑ **TRUE constant**—Right-click the **ShowColor** input of the FontDialog Property Node and select **Create»Constant**. Set the value of the constant to TRUE.

- ❑ **FontDialog Invoke Node**—Right-click the **reference** output of the FontDialog Property Node and select **Create»Method for System.Windows.Forms.FontDialog Class»ShowDialog()** from the shortcut menu to create a FontDialog Invoke Node.

- ❑ **Ring Indicator**—Right-click the **ShowDialog** output of the FontDialog Invoke Node and select **Create»Indicator** from the shortcut menu to create a ring indicator described in Table 3-1.

4. Add the following items to the block diagram as shown in Figure 3-2 to get .NET references to font and color.

   - ❑ FontDialog Property Node

     – Right-click the **reference** output of the FontDialog Invoke Node and select **Create»Property for System.Windows.Forms.FontDialog Class»Font** to create another FontDialog Property Node.

     – Expand the second FontDialog Property Node to show two elements. Select **Color** as the second element.

   - ❑ Right-click the **Font** output of the FontDialog Property Node and select **Create»Indicator** from the shortcut menu to create the **Font** output described in Table 3-1.

   - ❑ Right-click the **Color** output of the FontDialog Property Node and select **Create»Indicator** from the shortcut menu to create the **Color** output described in Table 3-1.

5. Add the following items to the block diagram as shown in Figure 3-2 to close the reference and handle errors.

   - ❑ Place a Close Reference function on the block diagram.

   - ❑ Right-click the **error in** input of the Constructor Node and select **Create»Control** from the shortcut menu to create the **error in** input described in Table 3-1.

   - ❑ Right-click the **error out** output of the Close Reference function and select **Create»Indicator** from the shortcut menu to create the **error out** output described in Table 3-1.

National Instruments Not for Distribution

❑ Wire the error wire through the Error case of the Case structure to the **error out** indicator.

6. Create the icon and connector pane.

❑ Switch to the VI front panel.

❑ Organize the controls in a logical manner.

❑ Right-click the **ShowDialog** indicator and select **Replace» Modern»Ring & Enum»Enum** from the shortcut menu.

Note Converting the ring indicator into an enumerated type indicator allows you to better control Case structures with the result of the dialog.

❑ Create an icon and connector pane similar to Figure 3-3.

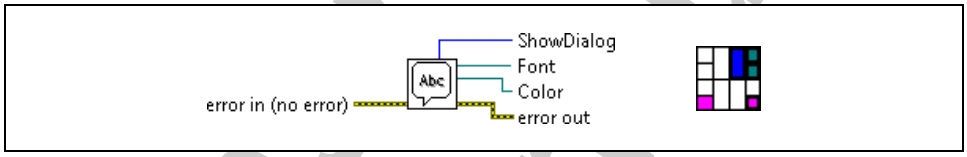


Figure 3-3. Font Dialog icon and Connector Pane

7. Save the VI.

Testing

Test the VI as a top-level VI.

❑ Run the VI. A font dialog should display. Notice that the font dialog may be behind the front panel. Minimize the front panel or press <Alt-Tab> to find the font dialog window.

❑ Click **OK** in the font dialog to finish the VI.

National Instruments Not for Distribution

Challenge

Test the VI as a subVI.

- ❑ Create a VI that calls the FontDialog VI.
- ❑ Check the **ShowDialog** to determine if the user clicked the **OK** button.
- ❑ Use the Font reference to display the selected Font Name.
- ❑ Use the Color reference to display the selected color in a LabVIEW **Color Box** indicator.

Tip Refer to the *Design* section for a suggestion on how to convert a .NET Color reference to a LabVIEW color.

End of Exercise 3-1

National Instruments
Not for Distribution

Exercise 3-2 Word Processor

Goal

Use a Windows Forms .NET Control.

Scenario

Develop a Word Processor in LabVIEW. The Word Processor should have the following features and should have a look and feel similar to other Windows word processors.

Word Processor Features

- Contains a standard, multiline area for typing text.
- Displays scrollbars when necessary.
- Provides a graphical toolbar to perform all functions.
- Allows the user to create a new, blank file.
- Allows the user to save the current text as a Rich Text File.
- Allows the user to change the font, color, and effects for selected text or for new text at the cursor position.
- Allows the user to cut text to the clipboard.
- Allows the user to copy text to the clipboard.
- Allows the user to paste text from the clipboard.
- Ends the application when the user clicks the close button on the title bar.

Design

You could implement all of this functionality in LabVIEW by using a text box control. However, manually implementing each of these features would be a considerable undertaking. A better approach is to utilize an existing component which has most of these features built in.

The .NET Framework contains common windows controls which you can use in a user interface. These controls behave very similarly to ActiveX controls in LabVIEW. One of the available controls is RichTextBox. This control is a text box, much like a LabVIEW string control, except that the control contains many built-in word processor functions. Most of the features specified in the *Scenario* are already included in RichTextBox.

National Instruments
Not for Distribution

However, you still need to provide an interface to activate the various functions.

Word Processor Inputs and Outputs

Table 3-2. Word Processor Inputs and Outputs

Type	Name	Properties	Default Value
.NET Container	TextBox	.NET Container with a System.Windows.Forms.RichTextBox object inserted	—
Boolean button	New	Strict Type Definition Custom Control	False
Boolean button	Save	Strict Type Definition Custom Control	False
Boolean button	Font	Strict Type Definition Custom Control	False
Boolean button	Cut	Strict Type Definition Custom Control	False
Boolean button	Copy	Strict Type Definition Custom Control	False
Boolean button	Paste	Strict Type Definition Custom Control	False

Event Handling Design

The user activates various functions of the word processor by pressing buttons on a toolbar. A common user interface event handler is the best design for this application because it allows the application to efficiently monitor and react to these buttons. The program consists of an initialization section before the event loop, a number of different events, and a cleanup section after the event loop. With the exception of the Close event, all of the events are Value Change events for button controls.

Initialization Section—Sets any properties which need to be set only once at the beginning of the program. For example, the scrollbars on the text box should be configured in this section by using the ScrollBars property of the RichTextBox.

New Event—Calls the Clear method of the RichTextBox to erase all text and create a blank slate for the user to type.

Save Event—Prompts the user to enter a filename and location by using a standard file dialog. If the user does not cancel the dialog, the event case should call the SaveFile method of the RichTextBox.

Font Event—Calls the Font Dialog VI you created in Exercise 3-1 to allow the user to select the font, color and other effects. If the user does not cancel

National Instruments Not for Distribution

the dialog, the references for the font and color should be applied to the RichTextBox by using the SelectionFont and SelectionColor properties.

Cut Event—Calls the Cut method of the RichTextBox.

Copy Event—Calls the Copy method of the RichTextBox.

Paste Event—Calls the Paste method of the RichTextBox.

Close Event—Exits the event loop and enters the cleanup section. This Event structure case should handle the Panel Close? filter event.

Cleanup Section—Closes all open references, including the RichTextBox reference, and handle errors with a simple error handler.

RichTextBox Control Description

The RichTextBox control is an enhanced text box with the ability to change text characteristics and a number of built-in word processor features. You can create a RichTextBox control by inserting the RichTextBox control from the System.Windows.Forms assembly into a .NET Container. You use the following properties and methods of the RichTextBox control in this exercise. Full documentation for the RichTextBox and other controls included in the .NET Framework can be found on MSDN or in the documentation for Microsoft Visual Studio .NET.

ScrollBars Property—Controls the scrollbars on the RichTextBox. Use this property to show or hide the horizontal and vertical scrollbars or to set the scrollbars so that they display only when necessary. For this exercise, set the property to Vertical to cause the vertical scrollbar to appear only when necessary.

Clear Method—Removes all text from the text box.

LoadFile Method—Loads the contents of a file into the RichTextBox. The default version of this method accepts a path and loads a Rich Text File into the text box. Another version of the method allows you to open file types other than Rich Text Files, such as Plain Text files or Unicode text files. For this exercise, you only need to load Rich Text Files.

SaveFile Method—Saves the contents of the RichTextBox into a file. Like the LoadFile method, alternate versions of this method can save in different file types. However you only need to save Rich Text Files for this exercise.

SelectionFont Property—Applies a Font reference to text in the RichTextBox. This property affects the font type, size, and effects, but does not set the font color.

National Instruments Not for Distribution

If no text is selected when this property is set, the font specified in this property is applied to the current insertion point and to all text that is typed into the control after the insertion point. The font setting applies until the property is changed to a different font or until the insertion point is moved to a different section within the control.

If text is selected within the control, the selected text and any text entered after the text selection have the value of this property applied to it.

SelectionColor Property—Applies a Color reference to text in the RichTextBox. This property behaves in the same way as SelectionFont with regards to selected text and insertion points.

Focus Method—Sets the input focus to the control. Use this property to return the cursor to the text box after displaying a dialog such as the font dialog or the dialog to search for a string.

Cut Method—Copies selected text to the Windows clipboard and then deletes it.

Copy Method—Copies selected text to the Windows clipboard.

Paste Method—Pastes text that has been previously placed on the Windows clipboard at the current cursor location.

Implementation

1. Open the Word Processor Project located in the `<Exercises>\LabVIEW Connectivity\Word Processor` directory.
2. Open `Word Processor.vi` from the **Project Explorer** window.

National Instruments
Not for Distribution

3. Build the VI front panel as shown in Figure 3-4.

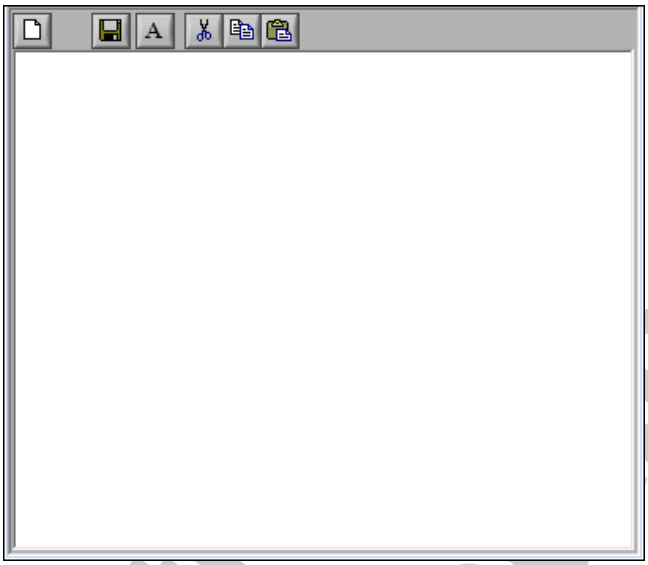


Figure 3-4. Word Processor Front Panel

- ❑ Place a .NET Container on the front panel of the VI. Label the container `TextBox`.
- ❑ Place a RichTextBox inside the container.
  - – Right-click the .NET Container and select **Insert .NET Control**.
  - – Select **System.Windows.Forms(2.0.0.0)** from the **Assembly** pull-down menu.
  - – Select **RichTextBox** from the **Controls** list.
  - – Click the **OK** button.
- ❑ Hide the label for the .NET Container.

4. Set window properties.
   - ❑ Select **File»VI Properties** and choose **Window Appearance** from the **Category** pull-down menu.
   - ❑ Click the **Customize** button and set the **Window Appearance** as shown in Figure 3-5.
   - ❑ Click the **OK** button. Click the **OK** button.

National Instruments
Not for Distribution

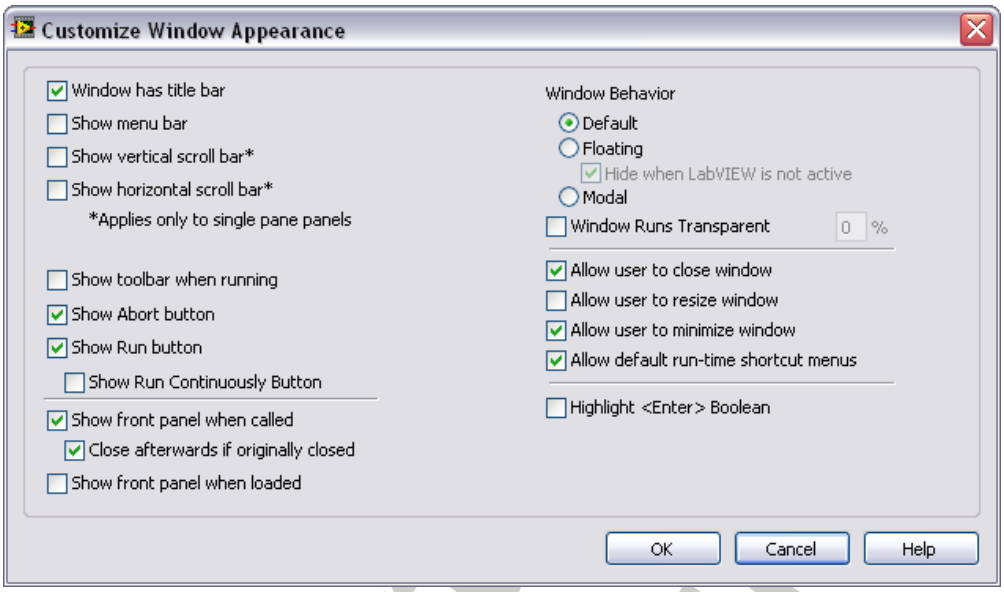


Figure 3-5. Word Processor Window Appearance Options

5. Create the initialization section, cleanup section, close event, and main program structure of the block diagram. Use the following items to create the block diagram as shown in Figure 3-6.

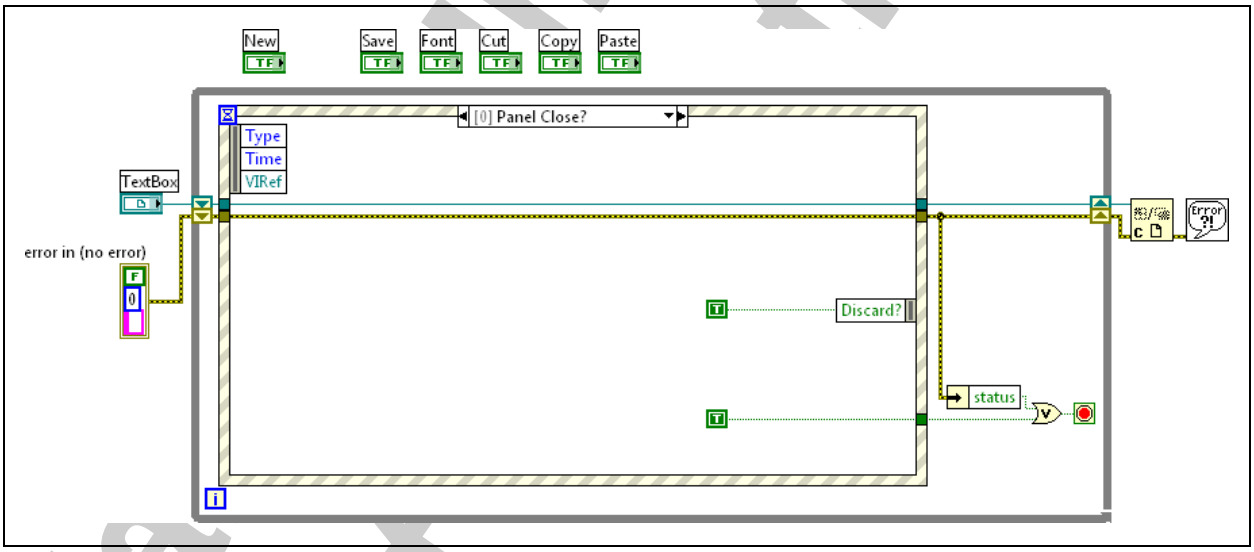


Figure 3-6. Word Processor Block Diagram

- While Loop
- Event structure
  - Right-click the border of the Event structure and select **Edit Events Handled by This Case**.

National Instruments Not for distribution

– Select **<This VI>** from the **Event Sources** list.

– Select **Panel Close?** from the **Events** list.

Note LabVIEW generates two events when you close a panel. The Panel Close event is a notify event that occurs after the panel closes. The Panel Close? event is a filter event that occurs before the panel closes and allows you to prevent the panel from closing. To use the toolbar to stop the application, you must use the filter event. Otherwise the VI closes each time you stop it.

– Click **OK**.

❑ Unbundle By Name

❑ Or

❑ Close Reference

❑ Simple Error Handler

❑ Two True constants

6. Create the New event as shown in Figure 3-7.

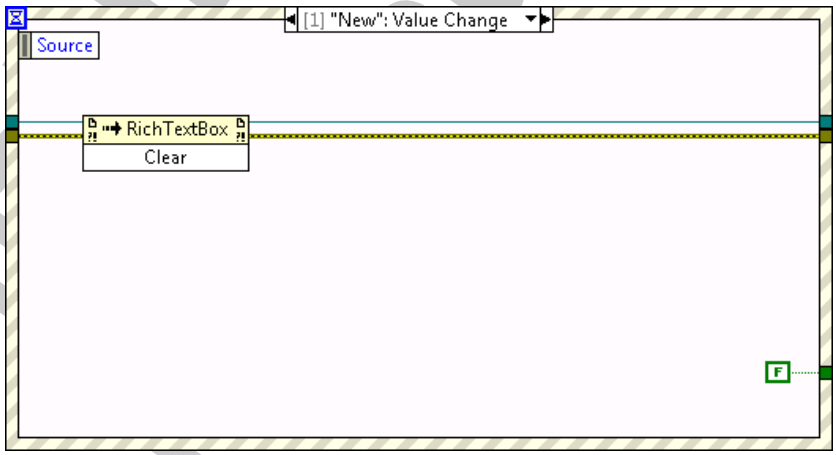


Figure 3-7. New Event

❑ Add the New case.

– Right-click the border of the **Event Structure** and select **Add Event Case** from the shortcut menu.

– Select **Controls»New** from the **Event Sources** list.

National Instruments Not for Distribution

– Select **Value Change** from the **Events** list.

– Click the **OK** button.

❑ Right-click the .NET Refnum wire and select **Create»Method for System.Windows.Forms.RichTextBox Class»Clear()** to create a Clear Invoke Node.

❑ Right-click the stop tunnel on the Event structure and select **Create» Constant** to create a False constant.

Note You may want to resize the Event Data Node to make space for your error and refnum wires. You should do this for each subsequent event.

7. Create the Cut event as shown in Figure 3-8.

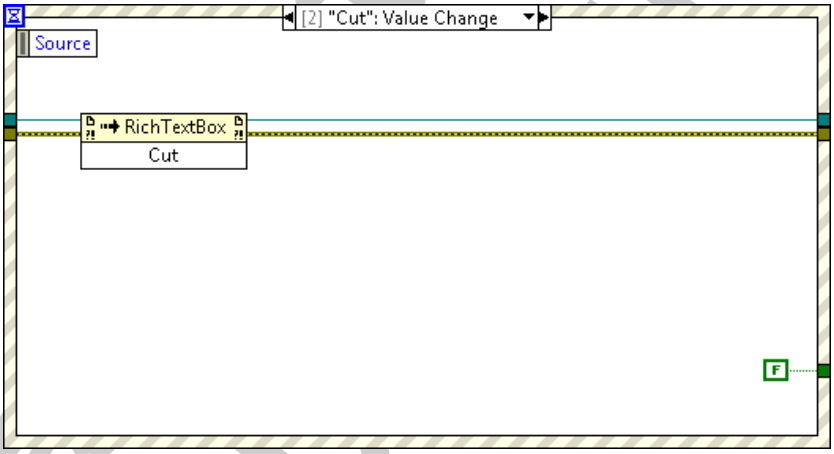


Figure 3-8. Cut Event

Note The order the events are created in makes no difference, so it makes sense to first create the events which you can easily duplicate from the New event case.

❑ Add the Cut event.

– Right-click the border of the **Event Structure** and select **Duplicate Event Case**.

– Select **Controls»Cut** from the **Event Sources** list.

– Select **Value Change** from the **Events** list.

– Click the **OK** button.

❑ Click the method in the RichTextBox Invoke Node and change it to **Cut()**.

National Instruments Not for Distribution

8. Create the Copy event.

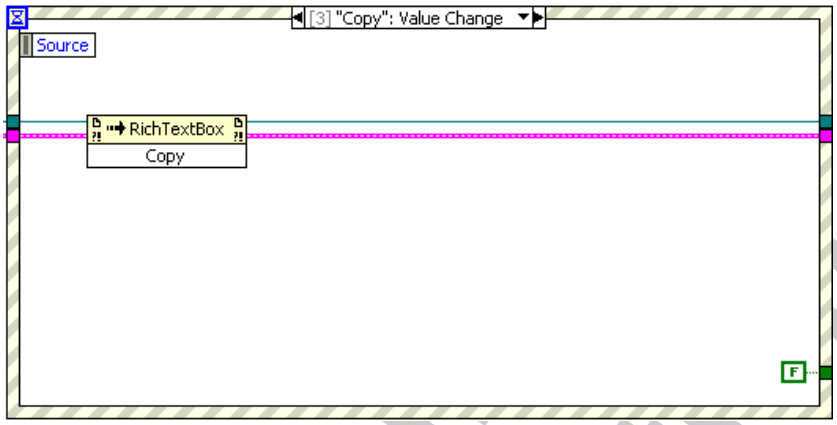


Figure 3-9. Copy Event

❑ Add the Copy event.

– Right-click the border of the **Event Structure** and select **Duplicate Event Case** from the shortcut menu.

– Select **Controls»Copy** from the **Event Sources** list.

– Select **Value Change** from the **Events** list.

– Click the **OK** button.

❑ Click the method in the RichTextBox Invoke Node and change it to **Copy**().

9. Create the Paste event.

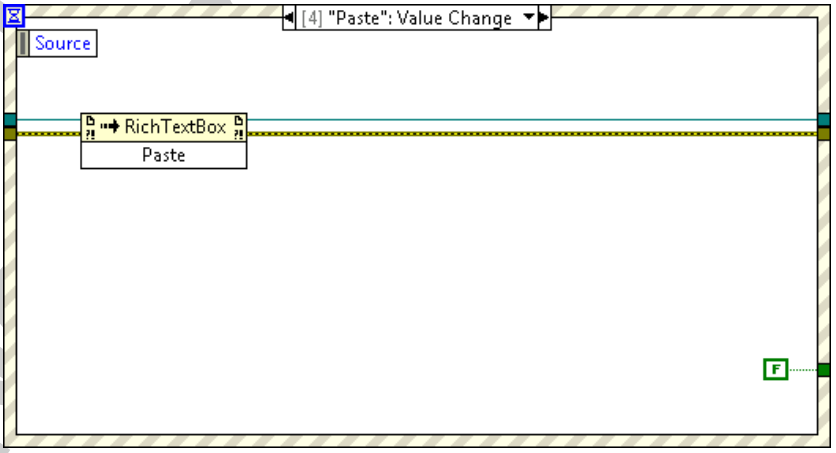


Figure 3-10. Paste Event

National Instruments Not for Distribution

❑ Add the Paste event.

– Right-click the border of the Event structure and select **Duplicate Event Case** from the shortcut menu.

– Select **Controls»Paste** from the **Event Sources** list.

– Select **Value Change** from the **Events** list.

– Click the **OK** button.

❑ Click the method in the RichTextBox Invoke Node and change it to **Paste**().

10. Create the Save event.

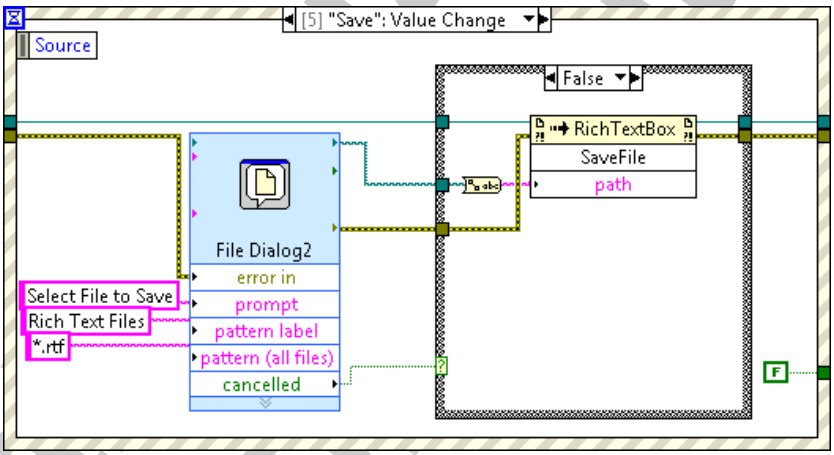


Figure 3-11. Save Event

❑ Add the Save event.

– Right-click the border of the Event structure and select **Add Event Case** from the shortcut menu.

– Select **Controls»Save** from the **Event Sources** list.

– Select **Value Change** from the **Events** list.

– Click **OK**.

❑ Place a **File Dialog** Express VI in the event case.

– Select **File** and **New or Existing** in the **Configure File Dialog** dialog box.

❑ Click **OK**.

National Instruments Not for Distribution

Tip Expand the File Dialog Express VI and select the terminals shown in Figure 3-11 to make the VI easier to wire.

- Right-click the **prompt** input of the File Dialog Express VI and select **Create»Constant** from the shortcut menu.
- Enter `Select File to Save` in the string constant.
- Right-click the **pattern label** input of the File Dialog Express VI and select **Create»Constant** from the shortcut menu.
- Enter `Rich Text Files` in the string constant.
- Right-click the **pattern(allfiles)** input of the File Dialog Express VI and select **Create»Constant** from the shortcut menu.
- Enter `*.rtf` in the string constant.

❑ Place a Case structure in the event case.

❑ Wire the **cancelled** output of the File Dialog Express VI to the Not function and wire the output of the Not function to the case selector terminal.

❑ Place a Path to String function inside the False case of the Case structure.

❑ Right-click the .NET Refnum wire and select **Create»Method for System.Windows.Forms.RichTexdBox Class»SaveFile(String path)** from the shortcut menu to create a SaveFile Invoke Node. Place the Invoke Node in the False case.

❑ Place the Clear Specific Error VI located in the Word Processor Project in the True case of the Case structure.

❑ Right-click the Code input of the Clear Specific Error VI and select **Create»Constant** from the shortcut menu to create a Numeric Constant.

❑ Enter `43` for the value of the Numeric Constant.

Note The Font Dialog Express VI returns error 43 when the user clicks the cancel button. The word Processor VI handles the cancel error explicitly by ignoring the open command and returning control to the user interface. Therefore, you can safely discard this error to prevent it from stopping the program and displaying an error message.

National Instruments Not for Distribution

- ❑ Right-click the stop tunnel on the Event structure and select **Create» Constant** from the shortcut menu to create a FALSE constant.
- ❑ Wire the diagram as shown in Figure 3-11 and Figure 3-12.

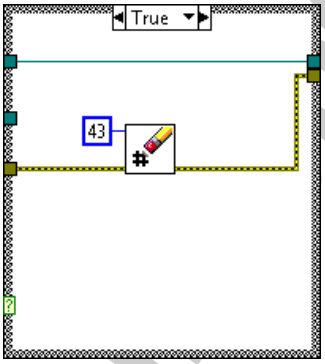


Figure 3-12. Save Event True Case

11. Create the Font event.

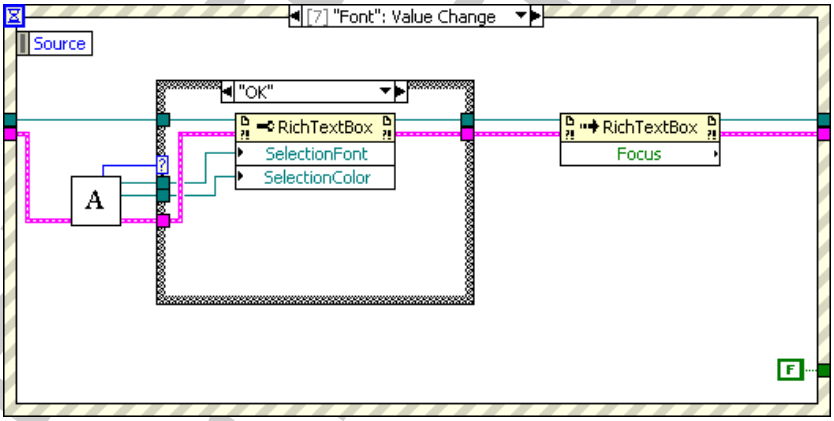


Figure 3-13. Font Event

- ❑ Add the Font event.
  - – Right-click the border of the Event structure and select **Add Event Case** from the shortcut menu.
  - – Select **Controls»Font** from the **Event Sources** list.
  - – Select **Value Change** from the **Events** list.
  - – Click **OK**.
- ❑ Place the Font Dialog VI that you created in Exercise 3-1 inside the event case. The VI is located in the `<Exercises>\LabVIEW Connectivity\Font Dialog` directory.

National Instruments Not for Distribution

- ❑ Place a Case structure in the event case.
- ❑ Wire the **ShowDialog** output of the Font Dialog VI to the case selector terminal.
- ❑ Right-click the .NET Refnum wire and select **Create»Property for System.Windows.Forms.RichTextBox Class»SelectionFont** to create a SelectionFont Property Node. Place the Property Node in the OK case of the case structure.
- ❑ Resize the Property Node to accept another property and set the second property to SelectionColor.
- ❑ Right-click the Property Node and select **Change All to Write**.
- ❑ Change to the None case of the Case structure.
- ❑ Change the case name to `"Cancel", Default`.
- ❑ Wire the error and refnum wires through the Cancel case of the case structure.
- ❑ Right-click the .NET Refnum wire and select **Create»Method for System.Windows.Forms.RichTextBox Class»Focus()** to create a Focus Invoke Node. Place the Invoke Node to the right of the Case structure.
- ❑ Right-click the stop terminal on the Event structure and select **Create»Constant** to create a False constant.
- ❑ Wire the diagram as shown in Figure 3-13.

12. Place each button terminal in the corresponding event case.

Note You place the button terminals in the event cases because the buttons use a latch Boolean mechanism. If the buttons are not in the event cases, they are not read when clicked, and remain pressed. Wait until the end of implementation to place the buttons in the cases to prevent creating duplicate buttons when you duplicate event cases.

13. Save the VI.

National Instruments
Not for Distribution

Testing

1. Test basic functionality.
   - ❑ Run the VI.
   - ❑ Enter text in the text box.
   - ❑ Ensure that word wrap works by typing to the end of a line.
2. Test the text box scrollbar.
   - ❑ Press the <Enter> key repeatedly until there is enough text for the vertical scrollbar to appear.
   - ❑ Use the scrollbar to scroll to the top of the document.
3. Test the Save function.
   - ❑ Click the **Save** button.
   - ❑ Click the **Cancel** button. The program should not stop or display an error.
   - ❑ Click the **Save** button again.
   - ❑ Navigate to the `<Exercises>\LabVIEW Connectivity\Word Processor` directory and enter `test.rtf` for the file name.
   - ❑ Click the **OK** button.
4. Test the New function.
   - ❑ Click the **New** button. The text field should clear and the scrollbar should disappear.
5. Test the Font function.
   - ❑ Close the block diagram of the VI if it is still open, also close any other unused windows. This makes it easier to find the font dialog if it does not appear as the front window.
   - ❑ Click the **New** button.
   - ❑ Click the **Font** button and select the font dialog. Remember that the font dialog may appear behind other windows.
   - ❑ Select a font, a style, a size and a color different from the default values.

National Instruments
Not for Distribution

- ❑ Click the **Cancel** button. The cursor should return to the text box.
- ❑ Enter a few words of text. Because you clicked the Cancel button, the text should have the default font, not the font you selected in the font dialog.
- ❑ Click the **Font** button again and select the font dialog.
- ❑ Select a font, a style, a size and a color different from the default values.
- ❑ Click the **OK** button. The cursor should return to the text box.
- ❑ Enter a few more words of text. The original text should be unchanged, but the new text should use the font, size, style, and color you selected.
- ❑ Use the mouse to select an area of text you have already entered. Try selecting an area around the change in fonts.
- ❑ Click the **Font** button and select the Font Dialog.
- ❑ Select a font, a style, a size, and a color different from both the default values and the values you selected before.
- ❑ Click the **OK** button. The selected text should now have the new text characteristics.

6. Test the Cut function.
   - ❑ Ensure that a word is still selected in the text box and click the **Cut** button. The selected word should disappear.
7. Test the Paste function.
   - ❑ Place the cursor at the end of the text you have entered.
   - ❑ Click the **Paste** button. The word you cut should appear at the cursor location.
8. Test the Copy function.
   - ❑ Select an area of text.
   - ❑ Click the **Copy** button. The text should remain in place.
   - ❑ Move the cursor to a different location.
   - ❑ Click the **Paste** button. The copied text should appear.

National Instruments Not for Distribution

9. Test the Close function.

   ❑ Click the **Close** button at the upper right hand corner of the window. The program should stop, but the VI should not close.

Challenge

There are many more features you can add to your word processor if time allows. Some suggestions include:

- Add an Open function.
- Add an Undo function.
- Add a Redo function.
- Add a display to the bottom of the window to display the number of lines and number of characters.
- Add Run-Time menus to perform the functions on the toolbar.
- Allow the find to search for instances of a string beyond the first.
- Add an options page to your program which allows the user to select things such as the background color and whether word wrap or horizontal scroll bars are used.
- Allow the user to resize the window. Make the text box resize, but prevent the resize from affecting the toolbar.
- Allow the text box to open and close plain text files.

End of Exercise 3-2

National Instruments
Not for Distribution

Exercise 3-3 Auto Save

Goal

Use .NET events to create a timer.

Scenario

Word processors commonly contain an Auto Save feature that periodically saves the work in them. This features helps users avoid losing data in the event of a crash or power failure.

Add an auto save function to the word processor. This function should save the document once every 30 seconds.

Design

There are a number of ways to implement an auto save feature. You could use the Timeout event of the LabVIEW Event structure to save the contents of the text box. However, the Timeout event occurs only after no events have fired for the specified time period. Therefore, if you configured the Timeout to 30 seconds, the program would only auto save if the user had not used any other functions for 30 seconds. You could use a smaller timeout, but you would need to track and check the amount of time since the last auto save. This would increase the complexity and decrease the performance of the program.

You could use a second LabVIEW loop to implement the auto save. The second loop could call the save function directly by branching the .NET refnum. Unfortunately, this could lead to errors if the first loop finished and closed the reference before the second loop finished. The second loop could also communicate with the main loop by firing a User Event every 30 seconds. While this implementation is fairly elegant, it significantly increases the complexity of the program. Furthermore, any implementation with a second loop must deal with a tradeoff between timing accuracy and increased processor usage from regularly checking the elapsed time.

A better solution would be to asynchronously register an event which occurs at a specified period. You can do this by using the .NET Timer Object from the System.Timers assembly. You can create a Timer object and specify a time interval so that the Timer periodically fires .NET events at the interval specified. You can handle the .NET event in LabVIEW by specifying a subVI to use as an event handler. By passing a reference to the RichTextBox to this subVI, you can asynchronously call the Save method every 30 seconds without having to program the timing mechanism in LabVIEW.

National Instruments
Not for Distribution

Auto Save Inputs and Outputs

Table 3-3. Auto Save Inputs and Outputs

Type	Name	Properties	Default Value
Cluster Control	error in	Error Cluster	No Error
.NET Refnum Control	TextBox reference	.NET Reference, System.Windows.Forms.RichTextBox class	Not a refnum
Numeric Control	Interval (ms)	Double precision numeric	30000
.NET Refnum Indicator	Timer reference	.NET Reference, System.Timers.Timer class	Not a refnum
.NET Refnum Indicator	TextBox reference out	.NET Reference, System.Windows.Forms.RichTextBox class	Not a refnum
Cluster Indicator	error out	Error Cluster	No Error

Timer Object Description

The Timer class generates recurring .NET events at a frequency that you specify. You can create a Timer object by selecting the Timer object from the System.Timers assembly using a .NET Constructor Node. One of the constructors for the Timer takes a numeric value to specify the interval. Calling this constructor is the easiest way to initialize a Timer. Use the following event and method from the Timer class in this exercise. Full documentation for the Timer and other objects included in the .NET Framework can be found on MSDN or in the documentation for Microsoft Visual Studio .NET.

Start method—Calling this method instructs the Timer to begin generating events. You can stop the event generation by calling the Stop method or by closing the Timer reference using a Close Reference function.

Elapsed event—After a Timer starts, it repeatedly generates the Elapsed event. The time between events is equal to the Interval property, which can be set directly, or initialized through the class Constructor.

National Instruments Not for Distribution

Implementation

1. Open the Word Processor Project located in the `<Exercises>\LabVIEW Connectivity\Word Processor` directory.
2. Add a new VI to the Word Processor Project and save it as `Auto Save.vi` in the `<Exercises>\LabVIEW Connectivity\Word Processor` directory.
3. Create the front panel as shown in Figure 3-14.

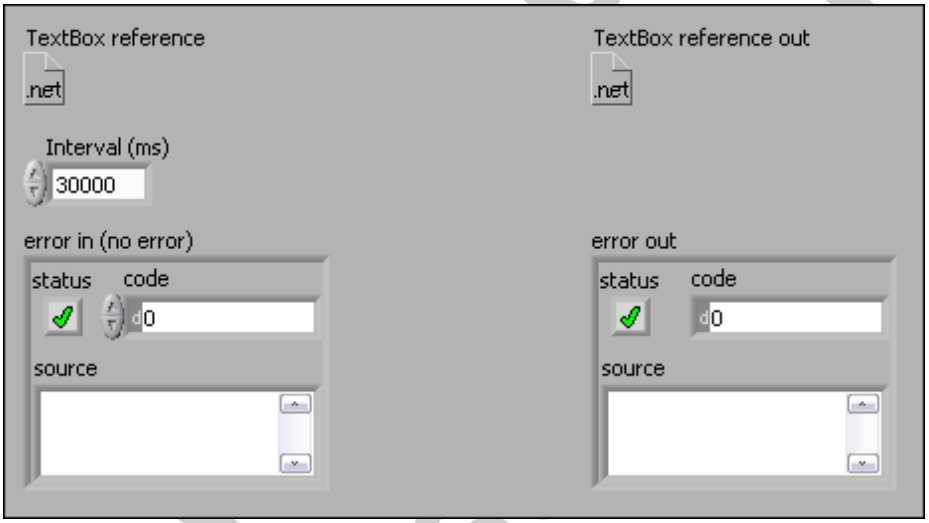


Figure 3-14. Auto Save VI Front Panel

- ❑ Create the **Interval** control as described in Table 3-3.
- ❑ Create the **Error In** control as described in Table 3-3.
- ❑ Create the **Error Out** control as described in Table 3-3.
- ❑ Place a .NET Refnum on the front panel.
  - – Right-click the .NET Refnum and choose **Select .NET Class» Browse** from the shortcut menu.
  - – Select **System.Windows.Forms** from the **Assembly** pull-down menu.
  - – Double-click the `System.Windows.Forms` item in the **Objects** list and select **RichTextBox**.
  - – Click **OK**.
  - – Label the control `TextBox Reference`.

National Instruments Not for Distribution

- ❑ Create a copy of the **TextBox reference** control. Name the copy `TextBox Reference out`.
- ❑ Right-click the **TextBox reference out** control and select **Change to Indicator** from the shortcut menu.

4. Add the following items and wiring as shown in Figure 3-15 to create a Timer object.

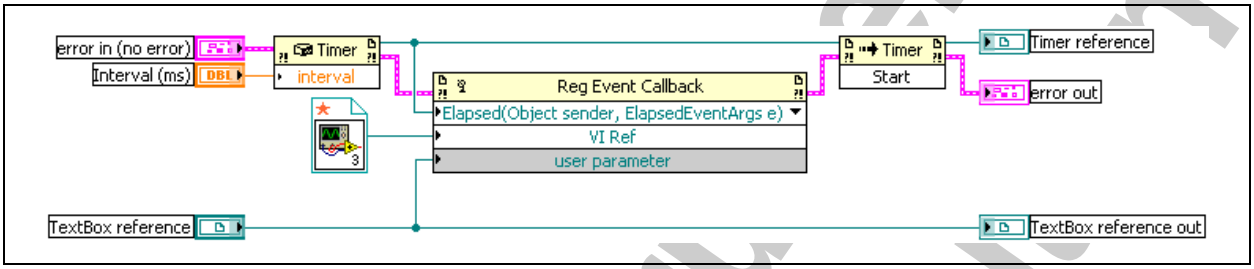


Figure 3-15. Completed Auto Save Block Diagram

- ❑ Place a Constructor Node on the block diagram as shown in Figure 3-15. The **Select .NET Constructor** dialog box should appear.
  - Select **System** from the **Assembly** pull-down menu.
  - Double-click the `System.Timers` item in the **Objects** list and select **Timer** to add Timer constructors to the **Constructors** list.
  - Select **Timer(Double interval)** from the **Constructors** list.
  - Click **OK**.
- ❑ Connect the **Interval** terminal to the **interval** input of the Constructor Node as shown in Figure 3-15.

5. Add the following items and wiring as shown in Figure 3-15 to register a handler for the Elapsed event.

- ❑ Place a Register Event Callback function on the block diagram.
- ❑ Wire the **reference** output of the Constructor Node to the **Event** input of the Register Event Callback function as shown in Figure 3-15.
- ❑ Click the **Event** input of the Register Event Callback function and select **Elapsed(Object sender, ElapsedEventArgs e)**.

National Instruments Not for Distribution

- ❑ Wire the TextBox Reference to the **user parameter** input of the Register Event Callback function as shown in Figure 3-15.

6. Add the following items and wiring as shown in Figure 3-15 to start the Timer.

   - ❑ Right-click the **reference** output of the Constructor Node and select **Create»Method for System.Timers.Timer Class»Start()** from the shortcut menu.

   - ❑ Right-click the reference output of the Start Invoke Node and select **Create»Indicator** from the shortcut menu. Name the indicator `Timer Reference`.

7. Create the Callback VI.

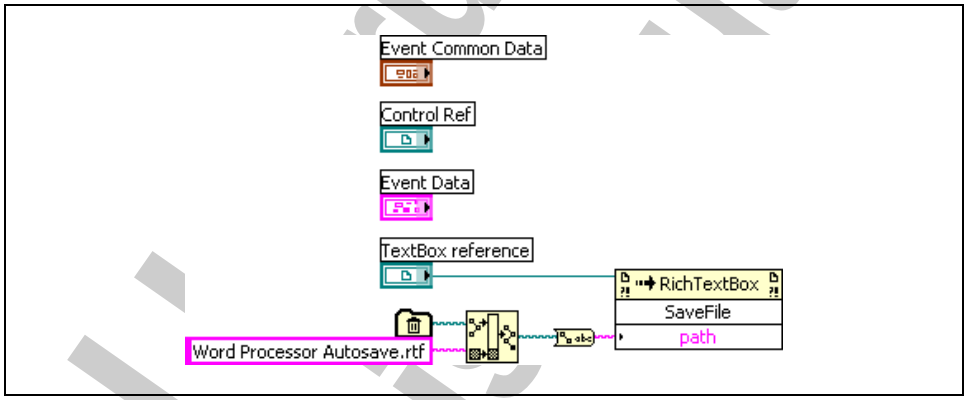


Figure 3-16. Auto Save Callback Block Diagram

- ❑ Right-click the **VI Ref** input of the Register Event Callback node and select **Create Callback VI** from the shortcut menu. This opens a new VI with the appropriate connector pane for this event callback.

- ❑ Save the callback VI as `Auto Save Callback.vi` in the `<Exercises>\LabVIEW Connectivity\Word Processor` directory.

- ❑ Open the block diagram of the callback VI.

- ❑ Right-click the TextBox reference terminal and select **Create» Method for System.Windows.Forms.RichTextBox Class» SaveFile(String path)** from the shortcut menu to create a SaveFile Invoke Node.

- ❑ Place a Temporary Directory constant on the block diagram.

- ❑ Place a Build Path function on the block diagram.

National Instruments Not for Distribution

- ❑ Right-click the **name or relative path** input of the Build Path function and select **Create»Constant** from the shortcut menu. Enter `Word Processor Autosave.rtf` for the value of the constant.
- ❑ Place a Path to String function on the block diagram.
- ❑ Wire the block diagram as shown in Figure 3-16.
- ❑ Save and close the Auto Save Callback VI.

8. Build the icon and connector pane for the Auto Save VI.
   - ❑ Return to the front panel of the Auto Save VI.
   - ❑ Right-click the VI icon and select **Edit Icon** from the shortcut menu.
   - ❑ Build the connector pane similar to the one shown in Figure 3-17.

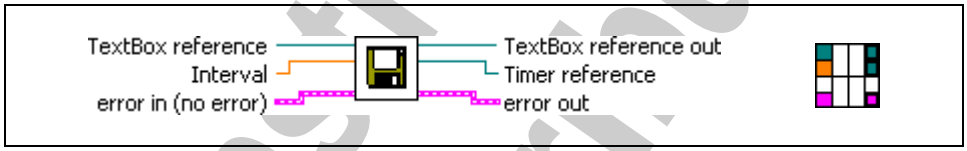


Figure 3-17. Auto Save Icon and Connector Pane

9. Save the VI.
10. Add auto save functionality to the Word Processor.

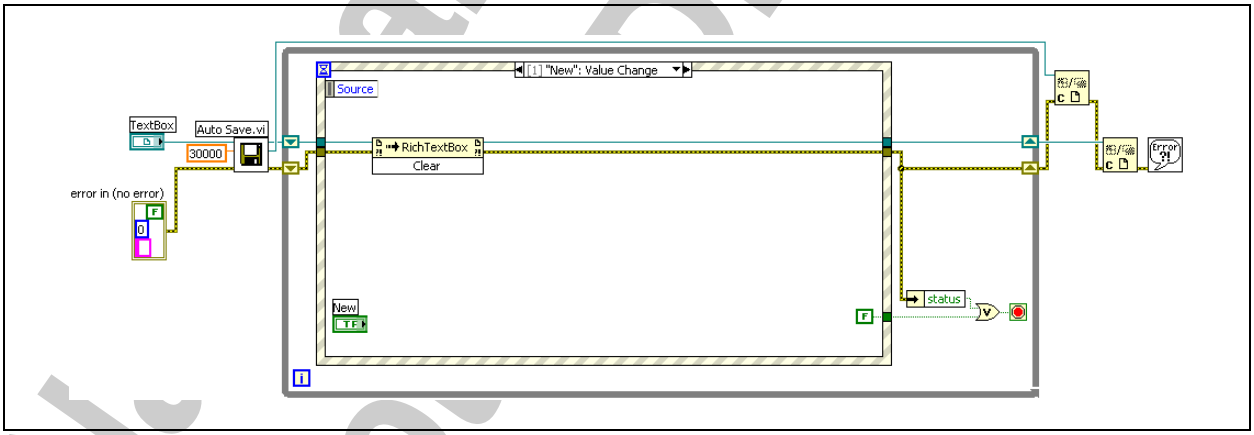


Figure 3-18. Word Processor with Auto Save Block Diagram

- ❑ Open the Word Processor VI located in the `<Exercises>\LabVIEW Connectivity\Word Processor` directory.
- ❑ Place the Auto Save VI on the block diagram of the Word Processor VI.

National Instruments Not for Distribution

- ❑ Place a Close Reference function on the block diagram of the Word Processor VI.
- ❑ Right-click the **Interval** input of the Auto Save VI and select **Create»Constant** from the shortcut menu. The constant should default to `30000`.
- ❑ Wire the diagram as shown in Figure 3-18.

11. Save the VI.

Testing

1. Navigate to the `Windows\Temp` directory.
   - ❑ Minimize all programs to get to the desktop of the computer.

Tip Press <Windows Key-D> to display the desktop.

- ❑ **(Windows XP)** Right-click the **My Computer** icon and select **Properties** from the shortcut menu to open the **System Properties** dialog box.
- ❑ **(Windows XP)** Select the **Advanced** tab and click the **Environment Variables** button.
- ❑ **(Windows 7/Vista)** Click the Start button, right-click **Computer** and select **Properties**.
- ❑ **(Windows 7/Vista)** Click **Advanced System Settings** and click the **Environment Variables** button.
- ❑ Select **TEMP** under **User variables**. If **TEMP** does not exist under **User variables**, locate it under **System variables**.
- ❑ Click the **Edit** button.
- ❑ Select all the text in the Variable value field then right-click the text and select **Copy** from the shortcut menu.
- ❑ Click the **Cancel** button. Do not change the value of the variable.
- ❑ Click the **Cancel** button twice to exit the **Environment Variables** and **System Properties** windows.

Tip This process shows you how to check the value of any system variable. However, an easier way to get the value of the TEMP variable would be to simply attach an indicator to a **Temporary Directory** constant in LabVIEW.

National Instruments Not for Distribution

- ❑ Open Windows Explorer.
- ❑ Paste the temporary directory path into the **Address** field and press <Enter>. Leave the Windows Explorer window open.

Note If you try to browse to the temporary directory instead of pasting the path, you may need to show hidden files from the Windows Explorer folder view options.

2. Test the Auto Save function.
   - ❑ Run the Word Processor VI.
   - ❑ Enter text into the text box.
   - ❑ Wait 30 seconds and then switch to the Windows Explorer window.
   - ❑ Double-click the `Word Processor Autosave.rtf` and verify that your text shows up in the file.
3. Save and close the Word Processor VI.

End of Exercise 3-3

National Instruments
Not for Distribution

Exercise 3-4 Browse to URL and Display VI Statistics Report

Goal

Use ActiveX automation to control Internet Explorer and then integrate that automation into an application.

Scenario

This exercise is divided into two parts, browsing to a URL and then using that code to display a VI Statistics report.

Part A: Browse to URL

Displaying a Web page or HTML report is a common task for measurement and test applications. Develop a reusable component that allows you to open a Web browser and navigate to a specified address.

Part B: Display VI Statistics Report

In Exercise 2-4 you created a VI to gather information about the VIs running on a computer. Currently, the exercise only displays this data to the front panel of the VI. In order to make the data more accessible, publish the data into an HTML report and use the reusable Web browser component to display the data.

Design

Part A: Browse to URL Inputs and Outputs

Table 3-4. Browse to URL Inputs and Outputs

Type	Name	Properties	Default Value
String Control	URL	—	Empty
Cluster Control	Error In	Error Cluster	No Error
Cluster Indicator	Error Out	Error Cluster	No Error

Microsoft provides two ActiveX interfaces for interacting with Internet Explorer. Internet Explorer can be embedded within another application as an ActiveX control, or Internet Explorer can be accessed as a separate entity through an automation server. Embedding Internet Explorer within a LabVIEW front panel is a powerful tool that allows your user to view web pages or other documents without changing programs. However, an embedded control requires an ActiveX container on the front panel, which is less reusable than a call to the automation server.

National Instruments Not for Distribution

To create a reusable web browser component, write a subVI that does the following:

1. Opens a reference to the Internet Explorer automation server.
2. Shows an Internet Explorer window.
3. Navigates to a web site specified by the URL control.
4. Closes the ActiveX reference.

IWebBrowser2 Interface Description

To open a reference to the Internet Explorer automation server, create an InternetExplorer.IWebBrowser2 object from the Microsoft Internet Controls Type Library. The Microsoft Internet Controls Type Library contains more than one IWebBrowser2 object, so be certain to select the object in the InternetExplorer folder. In this exercise, you use the following properties and methods of the IWebBrowser2 object:

- Visible property—When you open an Internet Explorer automation reference, the Internet Explorer window is hidden by default. Set the **Visible** property to TRUE to display the Internet Explorer window.
- Navigate method—The Navigate method instructs Internet Explorer to open a document or URL. Use the URL control to pass a string to this method. Notice that the IWebBrowser2 object also contains a Navigate2 method. In addition to standard URLs, the second method allows you to browse to non-standard folders such as the Printers folder or the Recycle Bin. Although the Navigate2 method also works for this exercise, use the Navigate method because you do not need the extra functionality.

Tip The LabVIEW context help provides descriptions of many ActiveX properties and methods.

Part B: Display VI Statistics Report Inputs and Outputs

Table 3-5. VI Statistics Report Inputs and Outputs

Type	Name	Properties	Default Value
String Control	Machine Name	—	Empty (defaults to local machine)
File Path Control	Report File Path	Browse button, **New or Existing Files** Selection Mode	`<Exercises>\LabVIEW Connectivity\VI Statistics Report\VI Statistics Report.html`

National Instruments Not for Distribution

In order to display the VI statistics data in a web browser, you must first format the data into HTML format. While you could do this manually using string manipulation, the Report Generation VIs in LabVIEW simplify the task considerably. A subVI with the appropriate report generation code is provided in the `<Exercises>\LabVIEW Connectivity\VI Statistics Report` directory.

Use the Generate VI Statistics HTML VI provided along with the Browse to URL VI that you created in Exercise 3-1 to display the results of the VI Statistics VI in a report. You can remove the current indicators to simplify wiring and avoid duplicating the displayed information.

Part A: Browse to URL Implementation

1. Create a blank VI and save it as `Browse to URL.vi` in the `<Exercises>\LabVIEW Connectivity\Browse to URL` directory.

2. Create the front panel as shown in Figure 3-19.

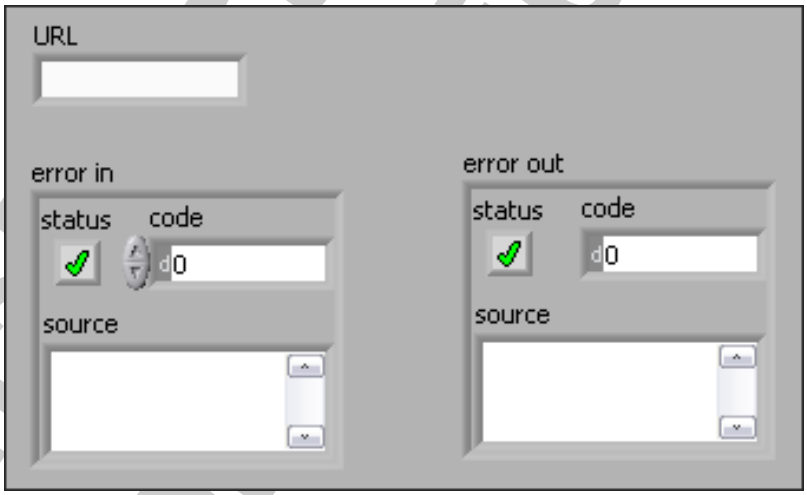


Figure 3-19. Browse to URL Front Panel

3. Create the following items as described in Table 3-4.

   - ❑ **URL** control
   - ❑ **Error In** control
   - ❑ **Error Out** indicator

4. Add the following items to the block diagram as shown in Figure 3-20 to open the automation reference.

National Instruments Not for Distribution

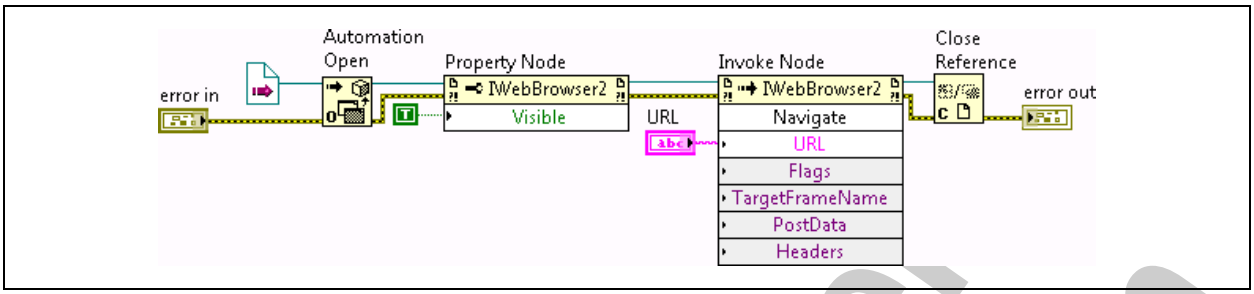


Figure 3-20. Completed Browse To URL Block Diagram

- ❑ Automation Open function
- ❑ Automation Refnum constant
  - Right-click the **automation refnum** input of the Automation Open function and select **Create»Constant** to create the Automation Refnum constant.
  - Right-click the Automation Refnum constant and choose **Select ActiveX Class»Browse.**
  - Select **Microsoft Internet Controls** from the **Type Library** pull-down menu.
  - Place a checkmark in the **Show Creatable Objects Only** checkbox.

5. In the **Objects** list, double-click the `InternetExplorer` item and select **IWebBrowser2**, as shown in Figure 3-21. Click **OK**.

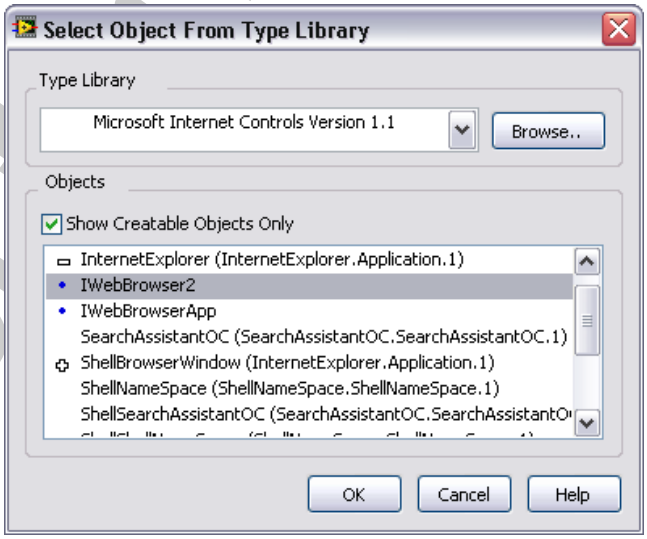


Figure 3-21. Select Automation Object

National Instruments
Not for Distribution

6. Add the following items and wiring to the block diagram to show the Internet Explorer window.

   - ❑ Visible Property Node
     - – Right-click the automation refnum output of the Automation Open function and select C**reate»Property for SHDocVw.IWebBrowser2 Class»Visible** to create the Visible Property Node.
     - – Right-click the Property Node and select **Change all to Write** from the shortcut menu.
   - ❑ True constant—Place a true constant on the block diagram and wire it to the Property Node.
   - ❑ Wire the error and refnum wires of the Automation Open function to the Visible Property Node.

7. Add the following items and wiring to the block diagram to browse to a URL.

   - ❑ Navigate Invoke Node—Right-click the **automation refnum** output of the Automation open function and select **Create»Method for SHDocVw.IWebBrowser2 Class»Navigate** from the shortcut menu to create a Navigate Invoke Node.
   - ❑ Wire the **URL** control to the Navigate Invoke Node.
   - ❑ Wire the error and refnum wires of the Visible Property Node to the Navigate Invoke Node.

8. Add the following item to the block diagram to close the automation reference.

   - ❑ Close Reference function

9. Create an icon and connector pane for the VI.

   - ❑ Right-click the VI icon and select **Edit Icon** from the shortcut menu.
   - ❑ In the Icon Editor select **Edit»Import Glyph from File**. Select `<Exercises>\LabVIEW Connectivity\Browse to URL\Images\ieiwn.bmp` as the file to import.
   - ❑ Double-click the **Rectangle** tool to create a black box around the icon.

National Instruments Not for Distribution

10. Click the **OK** button to exit the Icon Editor. The icon should resemble Figure 3-22.

11. Configure the connector pane for the VI similar to the example shown in Figure 3-22.

Figure 3-22. Browse to URL Icon and Connector Pane

12. Save the VI.

Testing

1. Browse to a file.

   ❑ Enter `C:\Program Files\National Instruments\LabVIEW X.X\readme\readme.html` in the **URL** control, where *X.X* is your LabVIEW version.

Note The preceding path may be different if you have installed LabVIEW to a directory other than the default installation directory.

   ❑ Run the VI. An Internet Explorer window should display the LabVIEW `readme` file.

2. Browse to a Web site.

   ❑ Enter `http://www.ni.com` in the **URL** control.

   ❑ Run the VI. An Internet Explorer window should display the National Instruments Web site.

Note You can browse to a Web site only if your computer has internet access.

National Instruments Not for Distribution

Part B: VI Statistics Report Implementation

Modify the front panel of the original VI Statistics VI so that it matches Figure 3-23.

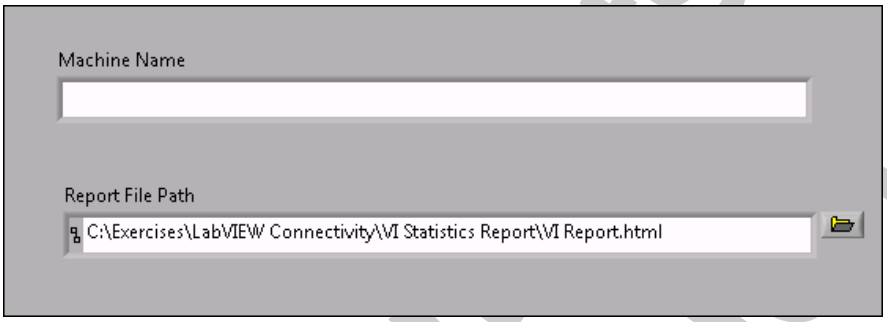


Figure 3-23. VI Statistics Report Front Panel

1. Open the VI Statistics VI that you created in Exercise 2-2 located in the `<Exercises>\LabVIEW Connectivity\VI Statistics` directory.

   - ❑ Save the VI as `VI Statistics Report.vi` in the `<Exercises>\LabVIEW Connectivity\VI Statistics Report` directory. Choose the **Substitute copy for original** option in the **Save As** dialog box.
   - ❑ Delete the **VIs Open**, **VIs Running**, and **VIs Report** indicators.
   - ❑ Create the **Report File Path** control as described in Table 3-5.

Tip To set the default value of the path control as described in Table 3-5, click the browse button and navigate to the appropriate folder. Enter `VI Statistics Report.html` in **File Name** and click **OK**. Right-click the path control and select **Data Operations»Make Current Value Default**.

2. Add the following items and wiring to the block diagram as shown in Figure 3-24 to generate an HTML report for the data.

National Instruments
Not for Distribution

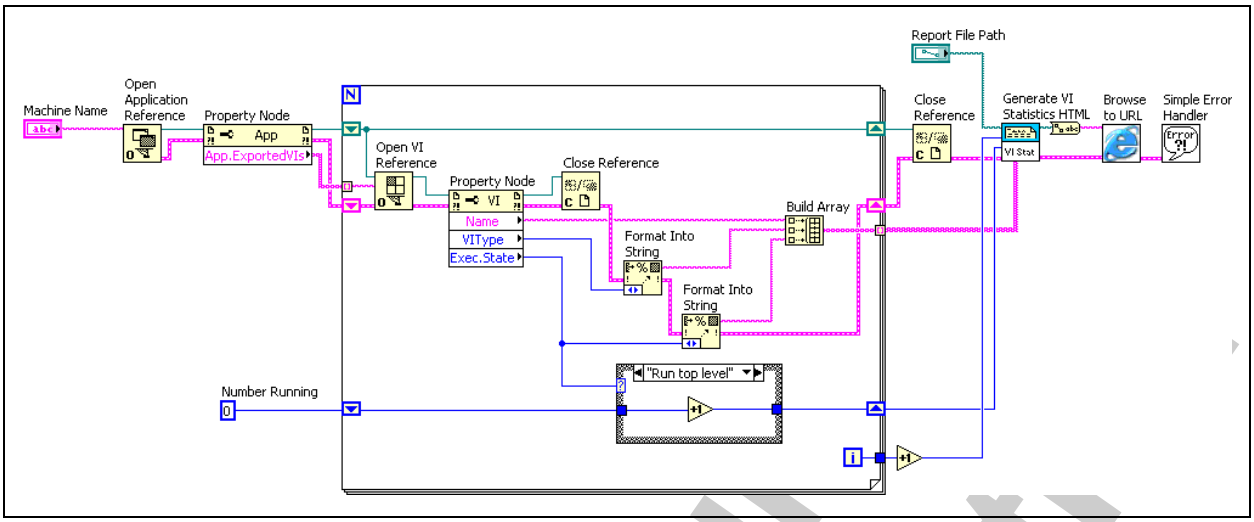


Figure 3-24. Completed VI Statistics Report Block Diagram

- ❑ Place `Generate VI Statistics HTML.vi`, located in `<Exercises>\LabVIEW Connectivity\VI Statistics Report` directory, on the block diagram.
- ❑ Open the subVI and inspect the code to generate the report. This subVI uses the Report Generation VIs in LabVIEW.

3. Add the following items and wiring to the block diagram to display the report in Internet Explorer.
   - ❑ Place `Browse to URL.vi`, located in the `<Exercises>\LabVIEW Connectivity\Browse to URL` directory on the block diagram. You created this subVI in Exercise 3-1.
   - ❑ Place a Path to String function on the block diagram.
   - ❑ Simple Error Handler
4. Save the VI.

National Instruments
Not for Distribution

Testing

1. Run the VI.
   - ❑ Switch to the front panel of the VI.
   - ❑ Close all other open VIs.
   - ❑ Run the VI. Notice that report generation loads a number of VIs into memory.
2. (Optional) Test the VI under other conditions.
   - ❑ Open one or more other VIs and run the VI. Observe the difference in states between the running and non-running VIs.
   - ❑ Enter the name of another machine in the **Machine Name** control and run the VI remotely.

End of Exercise 3-4

National Instruments
Not for Distribution

Notes

National Instruments
Not for Distribution

Notes

National Instruments
Not for Distribution

4

Connecting to Databases Exercises

Exercise 4-1 Viewing a Database

Goal

Examine the contents of a database using Microsoft Excel.

Description

Applications such as LabVIEW and Microsoft Excel can use the OLE DB standard to retrieve information from a database. In this exercise, you will use Microsoft Excel to view the records and fields in the three tables contained in the Theatre database.

The Theatre database contains the following three tables: Event_Table, Production_Table, and Cue_Information_Table. The Event table contains administrative information for every event that occurs. The Production table assigns a Production ID to each production. The Cue Information table contains timing and lighting data for the cues in each production.

- Event_Table consists of the following five fields: Event_ID, Production_Id, Date, Location, and Seats_Sold.
- Production_Table consists of the following two fields: Production_ID and Production_Name.
- Cue_Information_Table consists of the following eight fields: Cue_Id, Production_ID, Cue_Name, Wait_Time, Fade_Time, Follow_Time, Intensity, and Color.

Implementation

1. Launch Excel.
2. Import data from the database.
   - ❑ Select the **Data** tab.
   - ❑ In the **Get External Data** section, click **From Other Sources** and select **From Data Connection Wizard**. This launches the Data Connection Wizard that uses the OLE DB standard to communicate with the database.
   - ❑ Select **Other/Advanced** and click **Next**.

National Instruments Not for Distribution

- ❑ Select **Microsoft Jet 4.0 OLE DB Provider** and click **Next**.

Note The Microsoft Jet OLE DB Provider uses the Microsoft Jet database engine to expose data stored in Microsoft Access databases (.mdb) and other databases.

- ❑ Click the Browse button and navigate to `<Exercises>\LabVIEW Connectivity\Theatre Database\`. Select `TheatreDatabase.mdb`. Click **Open**.
- ❑ Click the **Test Connection** button to verify that there is a successful connection. In the Microsoft Datalink dialog box, click **OK**.
- ❑ In the **Datalink Properties** dialog box, click **OK**.
- ❑ In the **Data Connection Wizard**, select `Cue_Information_Table` and click **Next**. Click **Finish**.
- ❑ In the **Import Data** dialog box, click **OK**. Excel imports the Cue Information data from the database and displays it in a spreadsheet.

3. Examine the contents of the table. Save the table in its own worksheet.

Note In Microsoft Excel 2003 or 2007, to select a worksheet, click the tabs in the lower left. The default names for worksheets in Excel are Sheet1, Sheet2 and Sheet3.

4. Repeat steps 2 and 3 for the following tables:
   - `Event_Table`
   - `Production_Table`
5. Save the file as `TheatreDatabase tables.xlsx` in the `<Exercises>\LabVIEW Connectivity\Theatre Database` directory.

End of Exercise 4-1

National Instruments
Not for Distribution

Exercise 4-2 Connect to a Theatre Database Using LabVIEW

Goal

Configure and open a connection to a database using LabVIEW.

Scenario

Using LabVIEW to communicate with a database allows you to perform many database operations programmatically. The first step to communicate with a database using LabVIEW is to successfully open a connection to the database. In this exercise, you will connect to a database with LabVIEW using the OLE DB method.

Design

The VI you build will connect to a database using the OLE DB standard. You will create and configure a Universal Data Link (UDL) file which the VI will use to establish a connection with the Theatre database.

You will use the Database Connectivity Toolkit VIs.

The completed VI will perform the following tasks:

1. Open a connection to the Theatre database.
2. Close the connection to the Theatre database.
3. Check for errors.

Implementation

Connect to Database Using OLE DB

1. Create and configure a UDL file.
   - ❑ Launch LabVIEW.
   - ❑ Open a blank VI.
   - ❑ Select **Tools»Create Data Link**.
   - ❑ Select **Microsoft Jet 4.0 OLE DB Provider** and click **Next**.
   - ❑ Click the Browse button and navigate to `<Exercises>\LabVIEW Connectivity\Theatre Database\TheatreDatabase.mdb`.
   - ❑ Click the **Test Connection** button to verify that there is a successful connection and click **OK**.

National Instruments Not for Distribution

- ❑ Click **OK**.
- ❑ Navigate to `<Exercises>\LabVIEW Connectivity\Theatre Database\` and enter `MyUDL` for **File name**. Click **OK**.
- ❑ A dialog box informs you the data link has been successfully created. Click **OK**.

2. Create the front panel shown in Figure 4-1.

Figure 4-1. Open DB Connection Using OLE DB VI Front Panel

- ❑ Create a blank VI and save the VI as `Open DB Connection Using OLE DB.vi` in the `<Exercises>\LabVIEW Connectivity\Theatre Database\` directory.
- ❑ Place a File Path control on the front panel and label the control `UDL File Path`.

3. Create a VI to open and close a connection to a database using OLE DB. Build the block diagram shown in Figure 4-2 using the following items:

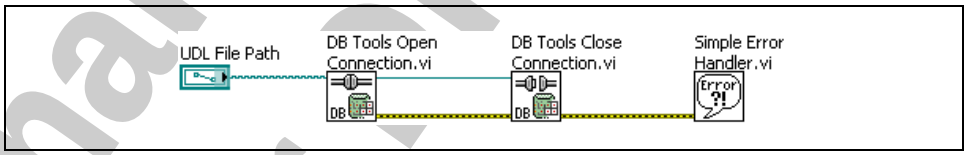


Figure 4-2. Open DB Connection Using OLE DB VI Block Diagram

- ❑ DB Tools Open Connection VI
- ❑ DB Tools Close Connection VI
- ❑ Simple Error Handler VI

National Instruments
Not for Distribution

Testing

1. On the front panel, set the **UDL File Path** control to `WrongUDL` and run the VI.

   This triggers error messages in a dialog box and in the error out indicator because `WrongUDL` is not a valid UDL.

2. Click the **UDL File Path** control browse button and navigate to `<Exercises>\LabVIEW Connectivity\Theatre Database\MyUDL.udl`.

3. Run the VI.

   If no errors occur, then the VI successfully opened and closed the connection to the database.

4. Save and close the VI.

End of Exercise 4-2

National Instruments
Not for Distribution

Exercise 4-3 Select Data from a Table

Goal

Retrieve data from the cue information table in the Theatre database.

Scenario

One of the most common database operations is retrieving data from the database. In this exercise, you will retrieve the cue name, wait time, fade time, follow time, light intensity, and light color for all the records in the Cue Information table in the Theatre database.

Design

You will use the Database Connectivity Toolkit VIs to retrieve data from the Theatre database. You will first create a VI that retrieves and displays the data as a variant data type. Then you will add code to convert the variant data type into specific LabVIEW data types and display the data.

Your final VI will perform the following tasks:

1. Open a connection to the Theatre database using a UDL file.
2. Retrieve data from the Cue Information table.
3. Convert the retrieved data from a variant data type to specific LabVIEW data types.
4. Close the connection to the Theatre database.
5. Check for errors.

National Instruments
Not for Distribution

Implementation

1. Create a blank VI and save it as `Select Data.vi` in the `<Exercises>\LabVIEW Connectivity\Theatre Database\` directory.

2. Create the first part of a VI to select data from the cue information table in the Theatre database. Create the block diagram shown in Figure 4-3 using the following items:

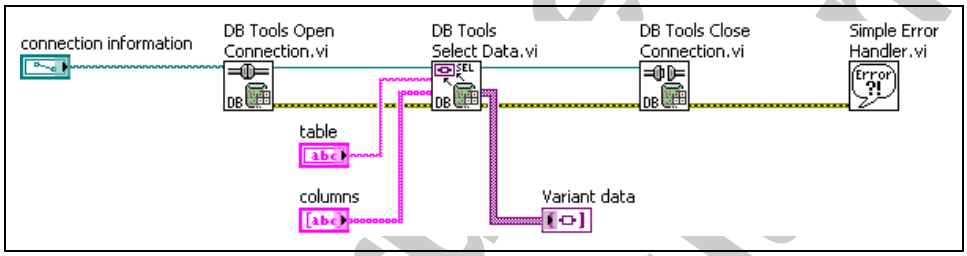


Figure 4-3. Select Data VI Block Diagram.

- ❑ DB Tools Open Connection VI—Right-click the connection information input terminal and select **Create»Control**.
- ❑ DB Tools Select Data VI
  - – Right-click the **table** input terminal and select **Create»Control**.
  - – Right-click the **columns** input terminal and select **Create» Control**.
  - – Right-click the **data** output terminal and select **Create» Indicator**. Change the label of the indicator to `Variant data`.
- ❑ DB Tools Close Connection VI
- ❑ Simple Error Handler VI

National Instruments
Not for Distribution

3. Create the front panel shown in Figure 4-4.

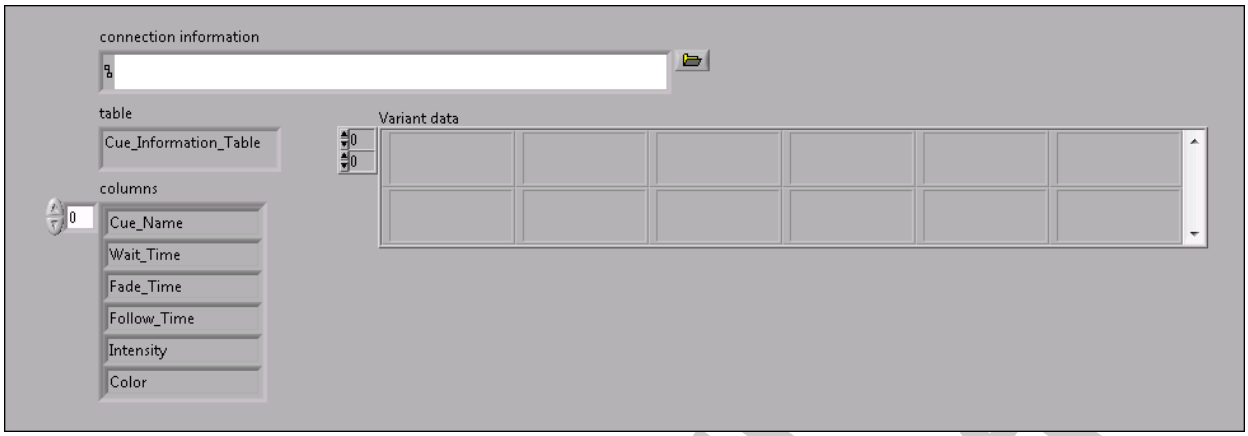


Figure 4-4. Select Data VI Front Panel

- ❑ On the front panel, click and drag the bottom edge of the **columns** string array indicator to resize it to six elements as shown in Figure 4-4.

Note Dragging one of the resize handles on the inside edge of the indicator increases the size of each cell. Dragging a handle on the outside edge increases the number of the cells.

- ❑ Right-click the blank area in the middle of the **Variant data** indicator and select **Visible Items»Scrollbar**. This removes the unnecessary scrollbar and declutters the interface.
- ❑ Click and drag the lower right corner of the **Variant data** indicator to resize it to two rows and six columns as shown in Figure 4-4.
- ❑ Right-click the outer edge of the **Variant data** indicator and select **Visible Items»Vertical scrollbar**.
- ❑ Arrange the controls and indicators as shown in Figure 4-4.

4. Get and display information from the database.

- ❑ Set the **connection information** control to `<Exercises>\LabVIEW Connectivity\Theatre Database\MyUDL.udl`.
- ❑ Set the **table** string control to `Cue_Information_Table`. This specifies the table in the database from which to access data.
- ❑ Right-click the **table** string control and select **Data Operations» Make Current Value Default**.

National Instruments
Not for Distribution

- ❑ Set the **columns** string array control to the values shown in the following figure. This specifies the fields in the database from which to access data.

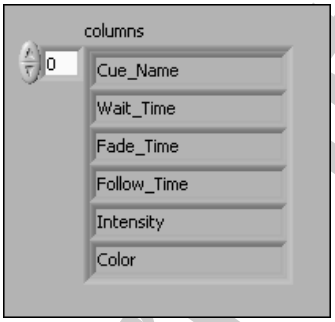


Figure 4-5. Columns Control on the Select Data VI Front Panel

- ❑ Right-click the border of the **columns** string array control and select **Data Operations»Make Current Value Default**.
- ❑ Run the VI. This should populate the **Variant data** indicator with values from the database.
- ❑ Scroll through the data by using the vertical scrollbar of the V**ariant data** indicator.

5. Save the VI.

   Although the data is displayed, the data is currently a variant datatype. To operate on the data, you must first convert it into its corresponding LabVIEW datatype.

National Instruments
Not for Distribution

6. In the second part of the exercise, convert the variant datatype into more usable LabVIEW datatypes by expanding the block diagram as shown in Figure 4-6 using the following items.

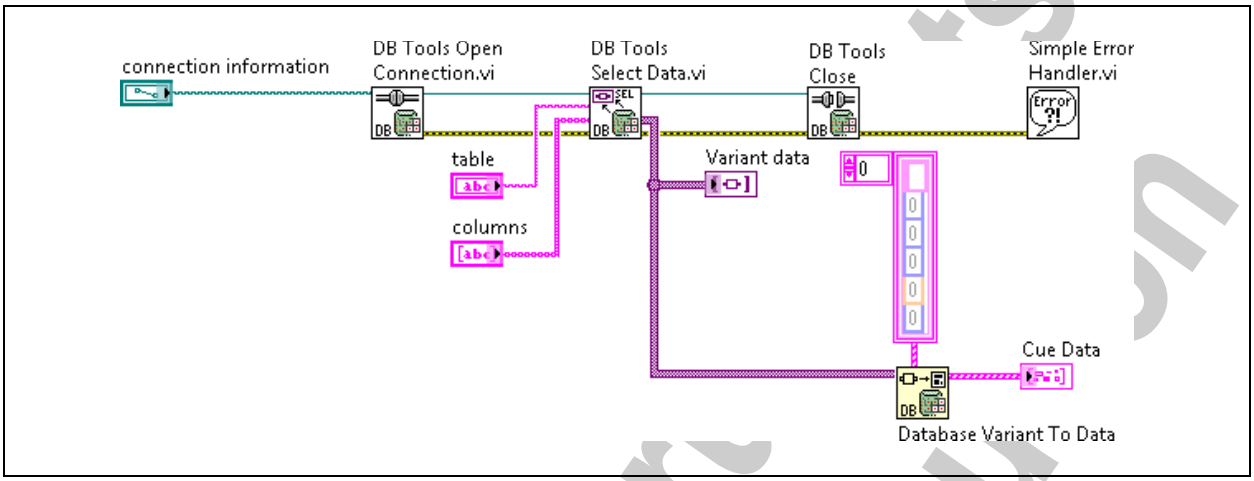


Figure 4-6. Completed Select Data VI Block Diagram

- ❑ Database to Variant Data function
- ❑ Create an array of clusters with the following elements.

Label	Data Type
Cue Name	String
Wait Time	U32
Fade Time	U32
Follow Time	U32
Intensity	DBL
Color	U32

National Instruments Not to be distributed

7. On the front panel, label the elements inside the Cue Data indicator to match Figure 4-7.

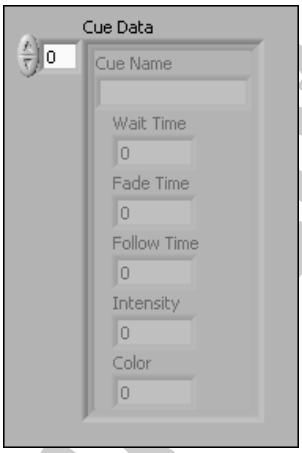


Figure 4-7. Cue Data elements on Select Data VI Front Panel

8. Right-click the **Color** indicator and select **Replace»Classic»Classic Numeric»Framed Color Box**.

9. Right-click the border of the **Cue Data** cluster indicator and select **Visible Items»Horizontal Scrollbar**. The completed front panel should be similar to Figure 4-8.

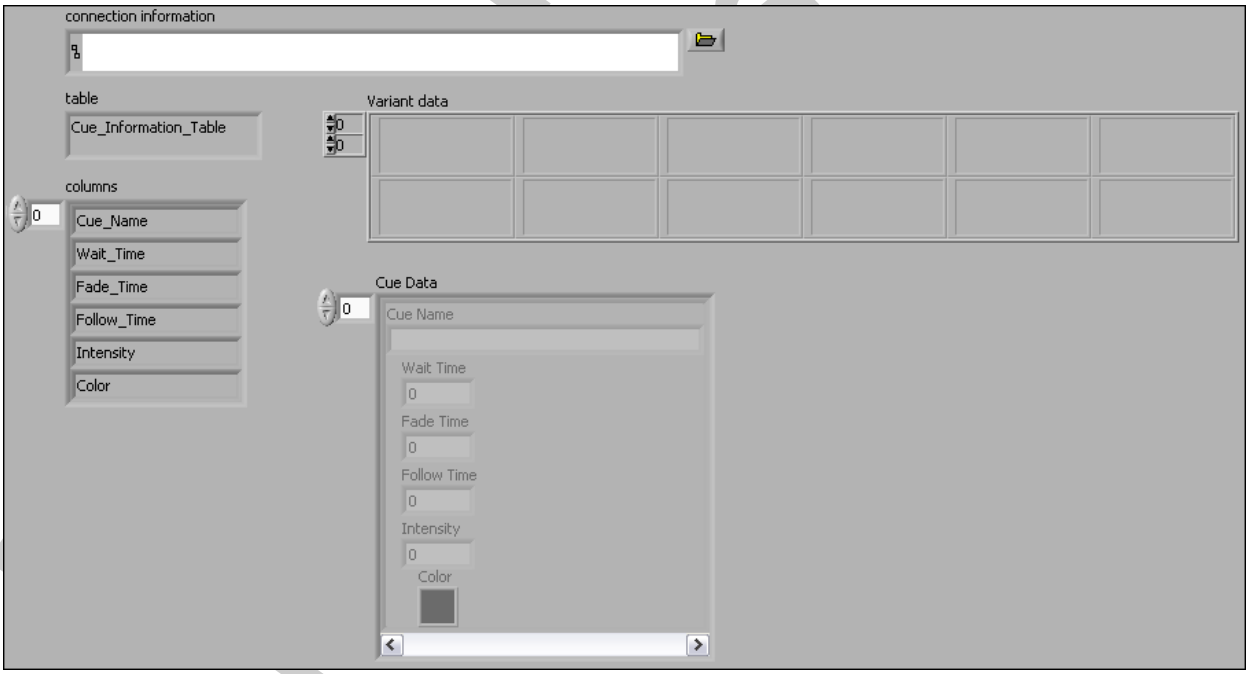


Figure 4-8. Completed Select Data VI Front Panel

National Instruments Confidential Distribution

10. Get cue data from the database and display the data as converted LabVIEW datatypes.

    - ❑ Run the VI. This populates the **Cue Data** indicator with values from the database.
    - ❑ Scroll through the data by using the index display or horizontal scrollbar of the **Cue Data** indicator. Notice that the data in the **Cue Data** indicator is more usable in LabVIEW than the data in the **Variant data** indicator.

11. Save the VI.

End of Exercise 4-3

National Instruments
Not for Distribution

Exercise 4-4 Insert New Record

Goal

Insert new records of cue data into the cue information table in the Theatre database.

Scenario

Another common database operation is inserting new records into a table in a database. In this exercise, you will insert new records of cue data into the cue information table in the Theatre database.

Design

You will use the Database Connectivity Toolkit VIs to insert new records into the Theatre database. You will also use the Record Cue Dialog Box subVI to launch a dialog box where the user can enter the cue data to insert into the cue information table in the Theater database.

Your completed VI will perform the following tasks:

1. Launch a dialog box that allows the user to enter cue data.
2. Open a connection to the Theatre database.
3. Insert a new record into the Cue Information table.
4. Close the connection to the Theatre database.
5. Check for errors.

Implementation

1. Open the Insert New Record VI in the `<Exercises>\LabVIEW Connectivity\Theatre Database\Insert New Record` directory.
2. Modify the block diagram to insert new records into a database table as shown in Figure 4-9 using the following items.

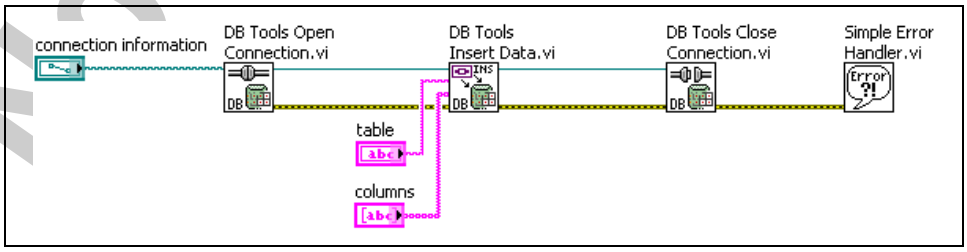


Figure 4-9. Insert New Record VI Partial Block Diagram

National Instruments Not for Distribution

- ❑ DB Tools Open Connection VI
- ❑ DB Tools Insert Data VI
- ❑ DB Tools Close Connection VI
- ❑ Simple Error Handler VI

3. Organize new cue data into a cluster datatype to insert a new record into the Cue Information table. Use the following items to expand the block diagram as shown in Figure 4-10.

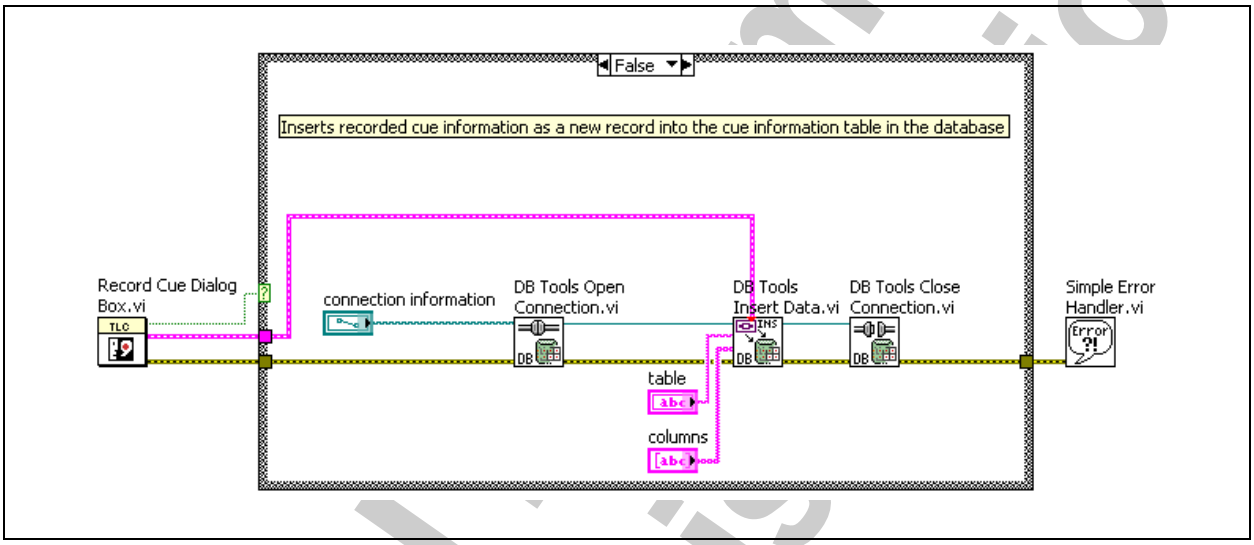


Figure 4-10. Insert New Record VI False Case

- ❑ Record Cue Dialog Box subVI—This subVI is located in the `<Exercises>\LabVIEW Connectivity\Theatre Database\Insert New Record` directory.

Note The Record Cue Dialog Box subVI displays a **Record** dialog box in which the user enters cue values. The VI then outputs the data through the **Output Cue** indicator.

- ❑ Case structure
  - Right-click the border of the Case structure and select **Make This Case False**.
  - Wire the **Cancelled?** output terminal of the Record Cue Dialog Box subVI to the Case Selector input terminal of the Case structure.
- ❑ Simple Error Handler

National Instruments Not for Distribution

4. Press <Ctrl-H> to open the Context Help window. Hover your mouse cursor over the wire that connects the Record Cue Dialog Box subVI to the **data** input of the DB Tools Insert Data VI. The Context Help window shows that the data type of the wire is a cluster that contains seven elements.

Note The DB Tools Insert Data VI inserts each item in the cluster into a table corresponding to each element in the **columns** input of the DB Tools Insert Data VI. The order is determined by the order of the cluster such that the 0th item in the cluster is inserted into the column name given in the 0th element of the columns input.

5. Create the block diagram for the **True** case of the Case structure as shown in Figure 4-11 using the following items.

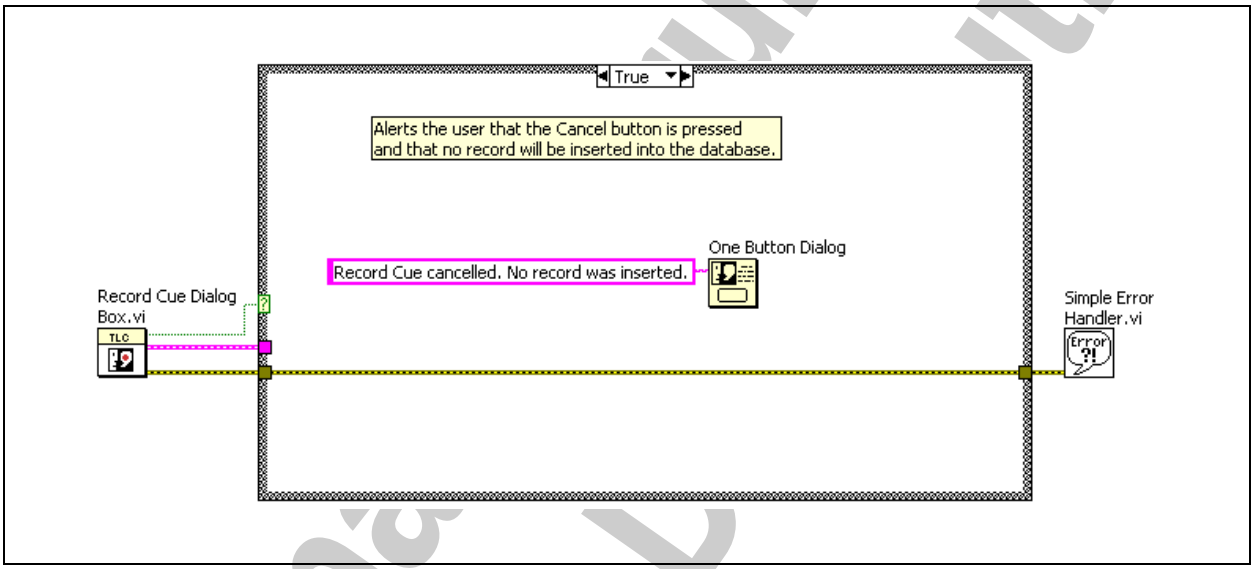


Figure 4-11. Insert New Record VI True Case

- ❑ One Button Dialog
  - – Right-click the **message** input terminal and select **Create» Constant**.
  - – Set the constant to `Record Cue cancelled. No record was inserted.`

National Instruments
Not for Distribution

Testing

1. On the front panel, set the **connection information** file path control to `<Exercises>\LabVIEW Connectivity\Theatre Database\MyUDL.udl`.

2. The **table** string control should be set to `Cue_Information_Table`. This specifies the table in the database in which to insert data.

3. The **columns** string array control should be set to the values shown in Figure 4-12. This specifies the fields in the table in which to insert data.

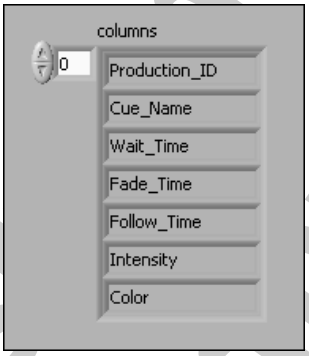


Figure 4-12. Columns Control on the Insert New Record VI Front Panel

4. Insert new records into the cue information table.

   ❑ Run the VI.

   ❑ In the **Cue Record** dialog box, set all the cue values to match the cue values shown in Figure 4-13. Select your own channel colors.

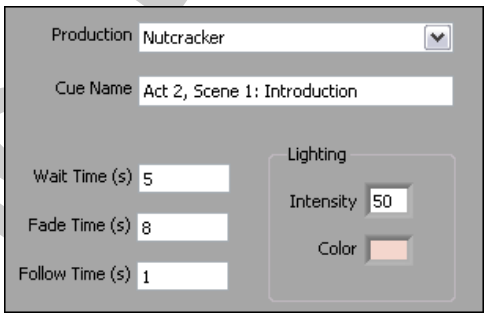


Figure 4-13. First Cue to Record

   ❑ Run the VI two more times to record the remaining cues shown in Table 4-1. Select your own lighting colors.

National Instruments
Not for Distribution

Table 4-1. Remaining Cues to Record

Cue Name	Wait Time	Fade Time	Follow Time	Intensity
Act 2, Scene 2: Clara and the Prince	1	5	8	95
Act 2, Scene 3: Various dances	2	5	9	100

5. View the records that you have inserted into the database.

   - ❑ Open the `Select Data.vi` you created in Exercise 4-3 from the `<Exercises>\LabVIEW Connectivity\Theatre Database` directory.
   - ❑ Set the **connection information** control to `<Exercises>\LabVIEW Connectivity\Theatre Database\MyUDL.udl`.
   - ❑ Set the **table** string control to `Cue_Information_Table`. This specifies the table in the database from which to access data.
   - ❑ Set the **columns** string array control to the values shown in the following figure. This specifies which fields in the database from which to access data.

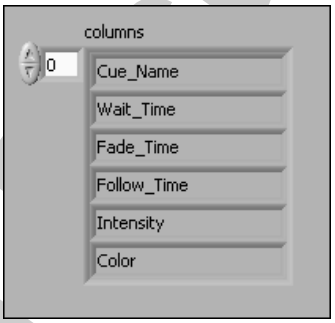


Figure 4-14. Columns Control on the Select Data VI Front Panel

   - ❑ Run the VI.
   - ❑ Scroll through the data using the index display or horizontal scrollbar of the **Cue Data** indicator. You can see the records that you have inserted at the end of the array.

6. Save and close the Insert New Records VI.

End of Exercise 4-4

National Instruments Not for Distribution

Exercise 4-5 SQL Query

Goal

Execute an SQL query statement to retrieve data from the Theatre database.

Scenario

The Structured Query Language (SQL) consists of a set of character string commands and is a widely supported standard for database access. You can use SQL commands to describe, store, retrieve, and manipulate the rows and columns in database tables. You will use the DB Tools Execute Query VI to send an SQL string to a database to query data from the Theatre database.

Design

You will modify the Select Data VI that you created in an earlier exercise to query data from the database using a SQL command. You will add the DB Tools Execute Query VI, DB Tools Fetch Recordset Data VI, and DB Tools Free Object VI to the application.

Your completed VI will perform the following tasks:

1. Open a connection to the Theatre database.
2. Send an SQL command to the Theatre database.
3. Retrieve the recordset returned from the SQL command.
4. Convert the retrieved data from a variant data type to specific LabVIEW data types.
5. Close the recordset reference.
6. Close the connection to the Theatre database.
7. Check for errors.

Implementation

1. Open the Select Data VI in the `<Exercises>\LabVIEW Connectivity\Theatre Database` directory.
2. Save a copy of the VI as `SQL Query.vi` in the `<Exercises>\LabVIEW Connectivity\Theatre Database` directory.

National Instruments Not for Distribution

3. Edit the block diagram to execute a SQL statement as shown in Figure 4-15.

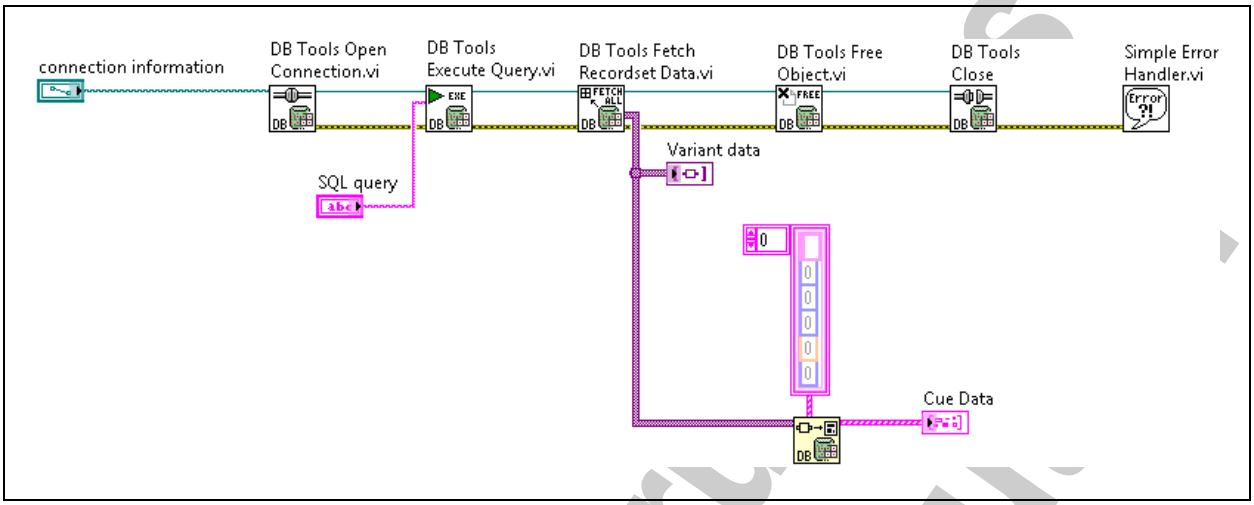


Figure 4-15. SQL Query VI Block Diagram

- ❑ Delete the following items:
  - Table control
  - Columns control
  - Error and connection reference wires connecting the Database Connectivity Toolkit VIs
- ❑ Add the following items:
  - DB Tools Execute Query VI
  - DB Tools Free Object VI
- ❑ Right-click the DB Tools Select Data VI and select **Replace» Database Palette»Advanced»DB Tools Fetch Recordset Data.vi**.

National Instruments
Not for Distribution

Testing

1. Set the **connection information** file path control to `<Exercises>\LabVIEW Connectivity\Theatre Database\MyUDL.udl`.
2. Set the **SQL Query** string control to `SELECT Cue_Name, Wait_Time, Fade_Time, Follow_Time, Intensity, Color FROM Cue_Information_Table WHERE Production_ID=4`.
3. Run the VI. The VI should return only **Cue Data** associated with the Nutcracker production.

End of Exercise 4-5

National Instruments
Not for Distribution

Notes

National Instruments
Not for Distribution

Notes

National Instruments
Not for Distribution

5

TCP/IP and UDP Exercises

Exercise 5-1 Simple Data Client VI and Simple Data Server VI

Goal

Open and examine some of the TCP/IP example VIs that show how you can use LabVIEW as a TCP client and a TCP server or how to send and receive UDP datagrams.

Scenario

Transferring data of TCP/IP is easy with LabVIEW. It is necessary to define the method that the data is transferred. The TCP and UDP functions in LabVIEW only send string data. So, all data types in LabVIEW need to be converted to a string before they are passed to a TCP or UDP function. Also, it is often necessary to send the size of the data with the packet.

In this exercise, notice how the data is passed from the TCP server to the TCP client. In particular, notice how the size of the data packet is determined and then sent to the server, before the data packet. Also in the exercise, notice how data is sent using UDP from a UDP sender VI to a UDP client VI.

Implementation

TCP Client

1. Open the NI Example Finder by clicking **Help»Find Examples**.
2. Select the **Search** tab and enter `tcp/ip` in the **Enter keyword(s)** text box and click **Search**.
3. Double-click **Simple Data Client.vi** in the search results. Do not close NI Example Finder.

National Instruments
Not for Distribution

4. Open and examine the block diagram as shown in Figure 5-1.

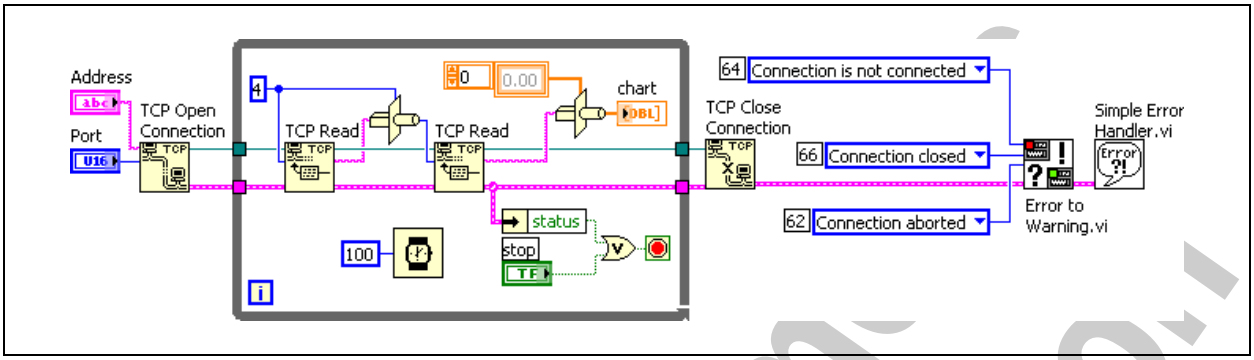


Figure 5-1. Simple Data Client VI Block Diagram

The TCP Open Connection function specifies which address and port you should use to open the connection.

The TCP Read function is configured to read four bytes of information. These four bytes are the size of the data and are type cast to a 32-bit signed integer (I32) and used for the **bytes to read** input on the second TCP Read function. The resulting data is type cast to an array of double-precision, floating-point scalar numbers (DBL) and displayed in a chart. The TCP Read functions are continuously called in a loop until you click the **STOP** button or an error occurs.

The Error to Warning VI converts non-critical connection errors to warnings.

The TCP Close Connection function is wired to stop the connection when you click the **STOP** button or an error occurs, and errors are reported by the Simple Error Handler VI.

TCP Server

1. Return to NI Example Finder, search for `tcp/ip` and double-click **Simple Data Server.vi** in the search results.

2. Open and examine the block diagram as shown in Figure 5-2.

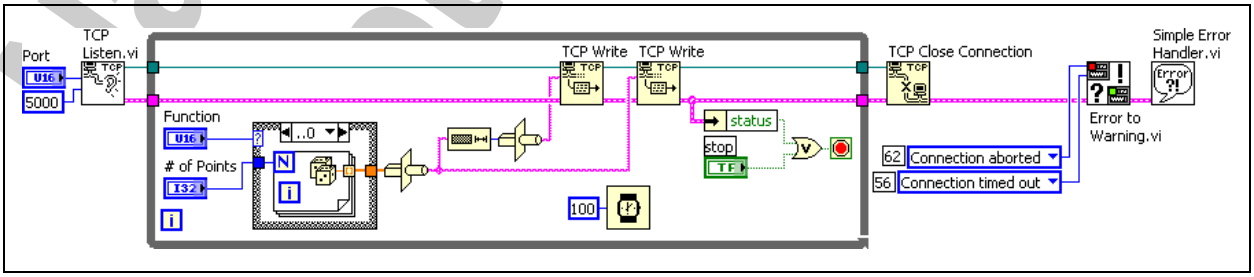


Figure 5-2. Simple Data Server VI Block Diagram

National Instruments Property for Distribution

The TCP Listen VI waits for a connection on the specified port for five seconds. The default **timeout ms** of `-1` causes the VI to wait indefinitely. The TCP Listen VI contains a TCP Create Listener function and a function to determine if a listener has already been created.

After a connection is made, a random or sine waveform is generated, based on the value of the **Function** control, and type cast to a string. The length of that string is type cast to a string and sent to the client with the TCP Write function. The first TCP Write function sets the amount of data to send. The second TCP Write function sends the waveform string. If a connection error occurs, error checking in the loop stops the loop.

The Error to Warning VI converts non-critical connection errors to warnings.

3. If the computer is connected through TCP/IP to another computer that has LabVIEW, and each computer has a unique IP address, you can run the Simple Data Client VI on one computer and the Simple Data Server VI on the other computer.

   - ❑ Find a partner and exchange IP addresses. Decide which computer is the server. Run the Simple Data Server VI on that computer.
   - ❑ Run the Simple Data Client VI with the IP address or hostname of the server on the other computer. You should see the random or sine waveform on the chart in the client VI as you change the **Function** control on the server VI.

Note To end the connection, click the **STOP** button on the client VI. You can run the client and server VIs on the same computer if the computer is not networked.

4. Close the Simple Data Client VI and Simple Data Server VIs. Do not save changes.

National Instruments
Not for Distribution

UDP Receiver

1. Return to NI Example Finder and search for UDP.
2. Open **UDP Receiver.vi** and examine the block diagram as shown in Figure 5-3.

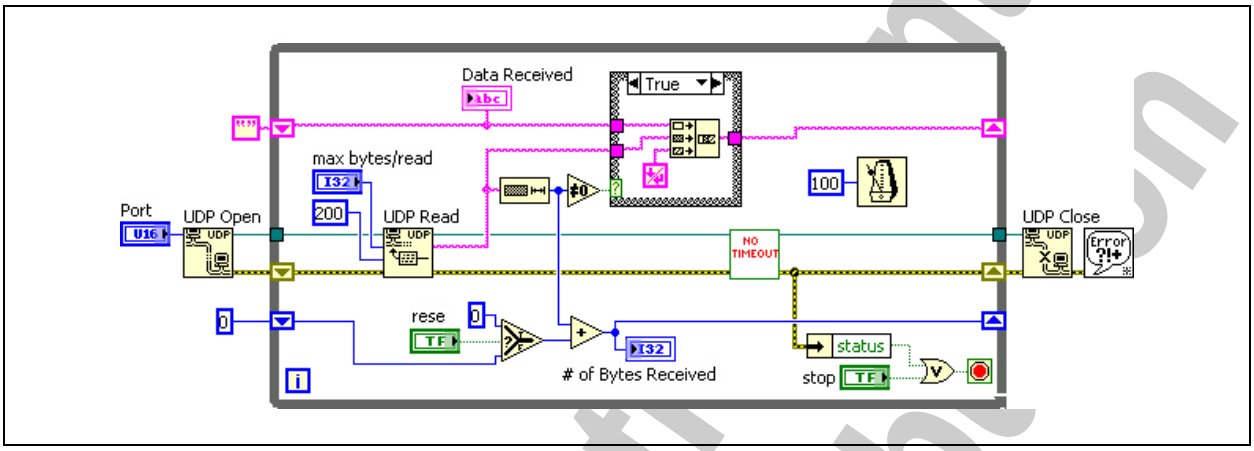


Figure 5-3. UDP Receiver

The UDP Open function opens a UDP socket on the specified port.

The UDP Read function reads a datagram from a UDP socket.

The UDP Close function closes a UDP socket.

National Instruments Not for Distribution

UPD Sender

1. Return to NI Example Finder and search for UDP.
2. Open **UDP Sender.vi** and examine the block diagram as shown in Figure 5-4.

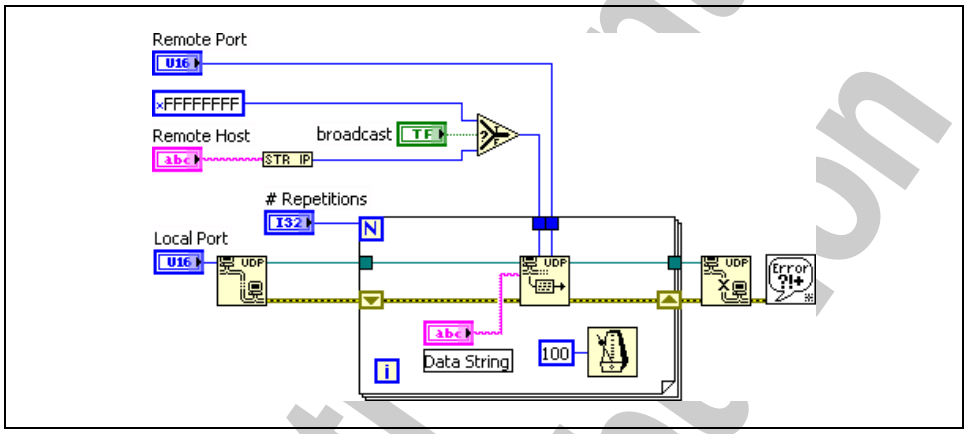


Figure 5-4. UDP Sender

The UDP Open function opens a UDP socket on the specified port.

The UDP Write function writes to a remote UDP socket specified by the Remote Host and Remote Port.

The UDP Close function closes a UDP socket.

3. Run **UDP Receiver.vi** and then run **UDP Sender.vi**. Notice data is sent to the receiver.
4. Click the **Stop** button on the UDP Receiver.
5. Close the UDP Receiver and UDP Sender VIs. Do not save changes.
6. Close the Example Finder.

End of Exercise 5-1

National Instruments Not for Distribution

Exercise 5-2 TCP Signal Data Transfer

Goal

Create a TCP program that transfers a signal from a server to a client

Scenario

You typically use TCP communication in situations where you need to reliably transfer data with no loss of data. For example, when transferring signal data, reliability is very important.

Write an application which uses TCP to transfer a simple signal (sine wave, sawtooth, and so on) from one computer to another. The application should be designed to have an extensible message protocol.

Design

When creating a TCP program, it is important to create a well defined protocol that the client and server use to communicate.

A common design pattern for the protocol is to send a message header followed by the data for that specific message. It is common for the header to contain a message ID and the message length.

The protocol header can be easily defined in LabVIEW by way of a cluster containing the protocol elements. When the client or server sends a message, it will first flatten and send the protocol cluster followed by more data for that message.

Client Server Architecture

Since a TCP client-server connection is bi-directional it does not matter whether the server sends or receives the signal. For this program the server will send the signal and the client will receive the signal.

For the file transfer program, transmit the number of chunks at the beginning of the program and then transmit the size of each chunk immediately before the chunk itself.

TCP Client—Signal Receiver Inputs and Outputs

Table 5-1. TCP File Sender Inputs and Outputs

Type	Name	Properties	Default Value
String Control	Address	—	localhost
Numeric Control	Port	Unsigned 16-bit Integer	6341

National Instruments Not for Distribution

Table 5-1. TCP File Sender Inputs and Outputs (Continued)

Type	Name	Properties	Default Value
Enum Typedef	Signal Type	Unsigned 32-bit enum	Sine
Waveform Graph	Waveform Graph	—	—
Boolean Control	Stop	Latch when released	False

TCP Client—Signal Receiver Program Flow

For this application, the TCP client will receive the signal. The TCP signal receiver should perform the following tasks:

1. Open a TCP connection to the server using the machine address and port number.
2. Send a message to the server requesting a specific signal (sine wave, sawtooth, etc.).
   - ❑ Create and send a message header to the server requesting a signal.
   - ❑ Send the desired signal type to the server as the message data.
3. Handle the server response message containing the requested signal, using the following steps:
   - ❑ Read the message header sent from the server.
   - ❑ Read the signal sent from the server.
   - ❑ Unflatten the signal string and then graph the signal.
4. When an error occurs or the user stops the client, send a **`Goodbye`** message to the server and close the TCP connection.

National Instruments
Not for Distribution

TCP Server—Signal Sender Program Flow

For this application, the TCP server will generate a signal and send the signal to the client. The TCP server should perform the following tasks:

1. Wait for incoming TCP connections. When a connection comes in, pass that connection to the reentrant connection handler VI. The TCP server can then forget about the connection since the reentrant instance of the connection handler VI now owns the specific TCP connection.
2. The connection handler will handle the request message header from the client using the following steps:
   - ❑ Read the message header sent from the client.
   - ❑ Decide how to handle the received message based on the Message ID from the header.
3. When a signal request message is received, perform the following steps:
   - ❑ Read the desired signal type from the TCP connection.
   - ❑ Generate the desired signal data and flatten it to a string.
   - ❑ Create and send a signal data message header to the client.
   - ❑ Send the flattened signal data to the client.
4. When an error occurs or a **`Goodbye`** message is received, close the TCP connection.

Implementation

Part A: TCP Client—Signal Receive

1. Open the TCP Signal Data Transfer project in the `<Exercises>\LabVIEW Connectivity\TCP Signal Data Transfer` directory.
2. Open `TCP Client.vi` from the **Project Explorer** window.

National Instruments Not for Distribution

3. Create the block diagram shown in Figure 5-5.

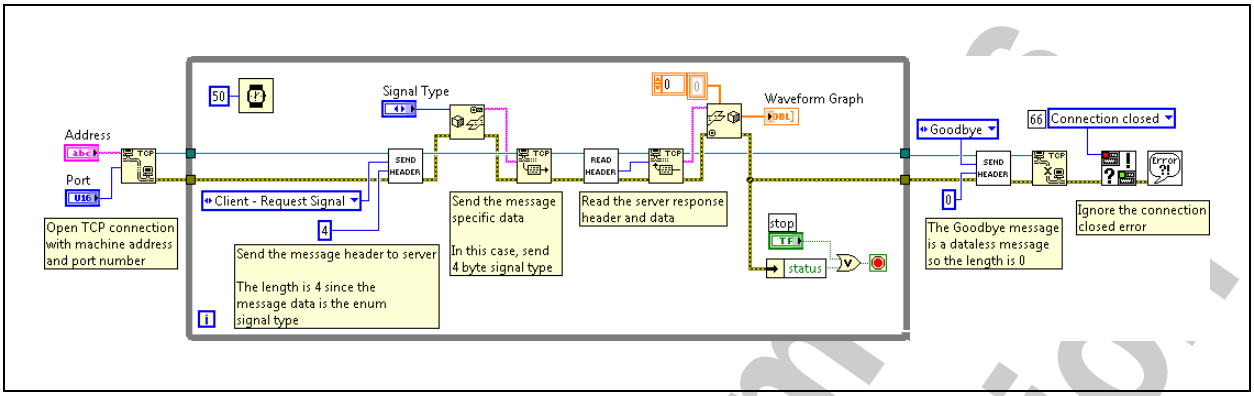


Figure 5-5. TCP Client Block Diagram

4. Create and wire the TCP Open Connection. Add the following items and wiring to the block diagram to open the TCP connection and file reference.

   - ❑ TCP Open Connection
   - ❑ Wire the address and port inputs
   - ❑ Wire the outputs **connection ID** and **error out**

5. Create a new message ID for the client to request a specific signal.

   - ❑ Open `TCP Message ID.ctl` located in the `Controls` directory of the TCP Signal Data Transfer project.
   - ❑ Add a `Client - Request Signal` item to the **Message ID** enum
   - ❑ Save and apply changes

6. Write code to send the Client - Request Signal message header. Add the following items to the block diagram:

   - ❑ Right-click the **Message ID** input of the Send Header VI and select **Create»Constant** from the shortcut menu. Set the value of the constant to `Client - Request Signal`.
   - ❑ Right-click the **Message Length** input of the Send Header VI and select **Create»Constant** from the shortcut menu. Set the value of the constant to `4`.

National Instruments Not for Distribution

7. Send the **Signal Type** as the message data. In this step, add the following items to the block diagram:
   - ❑ Flatten To String—Flatten the **Signal Type** enum to a string
   - ❑ TCP Write
   - ❑ Wire the string returned from the Flatten To String function into the TCP Write function
   - ❑ Wire the error and TCP connection ID
8. Read the signal sent by the server. In this step, add the following items to the block diagram:
   - ❑ TCP Read
   - ❑ Unflatten From String
   - ❑ Array of doubles constant for the Unflatten From String **type** input
     - – Right-click the **Waveform Graph** input and select **Create» Constant**.
   - ❑ Wire the Read Header VI **Message Length** output to the TCP Read **bytes to read** input.

Note In this example, the Message ID returned by the Read Header VI is ignored. When creating a more complicated TCP protocol, it may be desirable to verify the Message ID was received was the expected message.

Part B: TCP Server—Signal Sender

1. Open and examine `TCP Server.vi` in the `<Exercises>\LabVIEW Connectivity\TCP Signal Data Transfer` directory. Notice the following:
   - ❑ TCP Create Listener outside the While Loop
   - ❑ TCP Wait On Listener inside the While Loop
   - ❑ Static VI Reference to `TCP Connection Handler.vi`
   - ❑ Open VI Reference with option 0x8
   - ❑ Set Control Value method setting the Input Connection
   - ❑ Run VI method using FLASE for **Wait Until Done** and TRUE for **Auto Dispose Ref**

National Instruments Not for Distribution

2. Create a new message ID for the server to send a signal to the client.

   ❑ Open `TCP Message ID.ctl` in the `<Exercises>\LabVIEW Connectivity\TCP Signal Data Transfer` directory

   ❑ Add a **Server - Send Signal** item to the Message ID enum

   ❑ Save and apply changes

3. Open `TCP Connection Handler.vi` in the `<Exercises>\LabVIEW Connectivity\TCP Signal Data Transfer` directory.

4. Modify the TCP Connection Handler VI as shown in Figure 5-6.

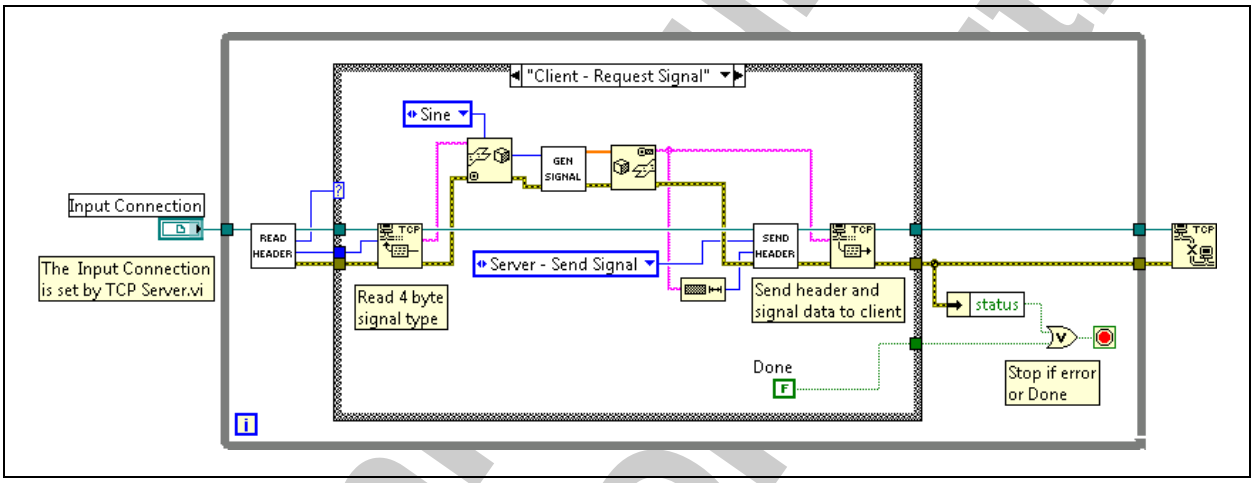


Figure 5-6. TCP Connection Handler Block Diagram

5. Add the Client - Request Signal message case.

   ❑ Right-click the border of the Case structure and select **Add Case After**.

6. Read the signal type from the TCP connection. In this step, add the following items to the Server - Send Signal case diagram:

   ❑ TCP Read

   ❑ Unflatten From String

   ❑ Signal Type constant from `Signal Type.ctl` from the **Controls** directory of the TCP Signal Data Transfer project and wire it to the Unflatten From String **type** input

National Instruments Not for Distribution

7. Generate the signal to send to the client. Add the following items to the Server - Send Signal case diagram.
   - ❑ `Generate Signal.vi` from the `SubVIs` directory of the TCP Signal Data Transfer project
   - ❑ Flatten To String
8. Send the Server - Send Signal message header. Add the following items to the block diagram:
   - ❑ `Send Header.vi` from the `SubVIs` directory of the TCP Signal Data Transfer project
   - ❑ Right-click the **Message ID** input of the Send Header VI and select **Create»Constant** from the shortcut menu. Set the value of the constant to `Server - Send Signal`.
   - ❑ String Length
   - ❑ Wire the Flatten to String to the String Length function. Wire the String Length function output to the Send Header VI **Message Length** input.
9. Create and wire the TCP Write node.
10. Create and wire a FALSE constant for Done.

Note The TCP Connection Handler VI you created does not report errors. It would be inappropriate to use the General Error Handler on the server because it is likely the server will not have a user present while running. If you wanted to record errors, you might want to create a server log file.

National Instruments Not for Distribution

Testing

Note The TCP Connection Handler VI is reentrant. To debug this VI, you must deal with the reentrant instances created by the TCP Server VI. To do this, save a breakpoint into in TCP Connection Handler VI. This breakpoint will be hit when the server receives a connection and spawns a handler.

1. Run the client and server on the same machine.
   - ❑ Run `TCP Server.vi` first and `TCP Client.vi` second.
   - ❑ While running, change the value of **Signal Type** on the TCP Client VI. You should see the appropriate signal on the graph.
   - ❑ Click **Stop** on the TCP Client VI.
2. (Optional) Run with the client and server running on separate machines.
   - ❑ Copy the TCP VIs to another machine, or interact with your neighbor's VIs.
   - ❑ Set **Address** on the TCP Client VI to be the machine name of the LabVIEW with TCP Server VI.
   - ❑ Repeat the steps from the first test.
3. (Optional) Run with multiple clients targeting one server.
   - ❑ Make a copy of the TCP Client VI and target the same TCP Server VI.
     - – Optionally, use the TCP Client VI from a second machine.
   - ❑ Repeat the steps from the first test where both the TCP Client VI and the copy are run.

End of Exercise 5-2

National Instruments
Not for Distribution

Notes

National Instruments
Not for Distribution

6

Web Services Exercises

Exercise 6-1 Create a LabVIEW Web Service to Add Two Numbers

Goal

Use a build specification to convert a simple VI that adds two numbers and shows the sum in a LabVIEW Web service, then use a Web browser to invoke the Web service, displaying the output terminal as text or XML.

Scenario

You have a subVI that accepts two numbers, adds them together, and returns the sum. You want to convert it into a Web service.

Design

You will be using the Application Builder to create a build specification for a RESTful Web service. You will configure the build specification and build the a Web service out of your VI. You will then enable the LabVIEW Web Application Server on your computer and deploy the Web service to it. Finally, you will use a Web browser to invoke your deployed Web service and display the sum of any two umbers

Implementation

1. Create a new LabVIEW Project called `Math Web Service.lvproj` and save it to the `<Exercises>\LabVIEW Connectivity\Web Services` directory
2. Create a new VI in the project and save it to `<Exercises>\LabVIEW Connectivity\Web Services\add.vi`.

National Instruments
Not for Distribution

3. Create a front panel similar to the one shown in Figure 6-1 using the following items:

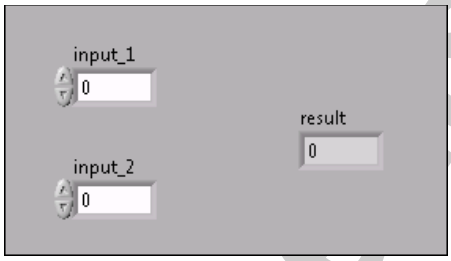


Figure 6-1. Add VI Front Panel

   - ❑ Two Numeric Controls—label one `input_1` and label the other `input_2`
   - ❑ Numeric Indicator labeled `result`

4. Configure the connector pane with two input terminals, one for the **input_1** control and one for **input_2** control, and one output terminal connected to the **result** indicator as shown in Figure 6-2.

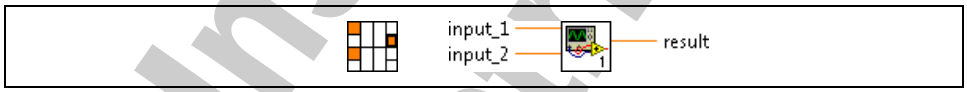


Figure 6-2. Add VI Connector Pane

5. Place an Add function on the block diagram and complete the block diagram shown in Figure 6-3.

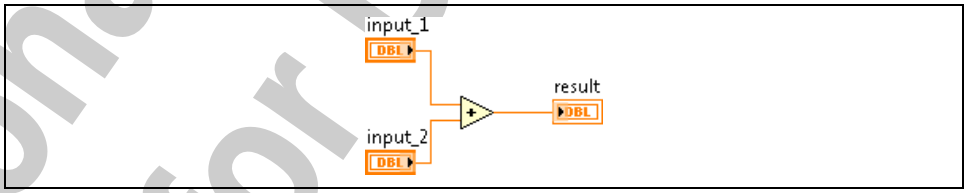


Figure 6-3. Add VI Block Diagram

6. Save and close the VI.
7. Create the build specification for your Web Service.
   - ❑ In the Project Explorer window, right-click **Build Specifications** and select **New»Web Service (RESTful)**.

National Instruments
Not for Distribution

8. Configure the settings for each of the following categories.

   ❑ Select the **Information** category and configure the following settings as shown in Figure 6-4.

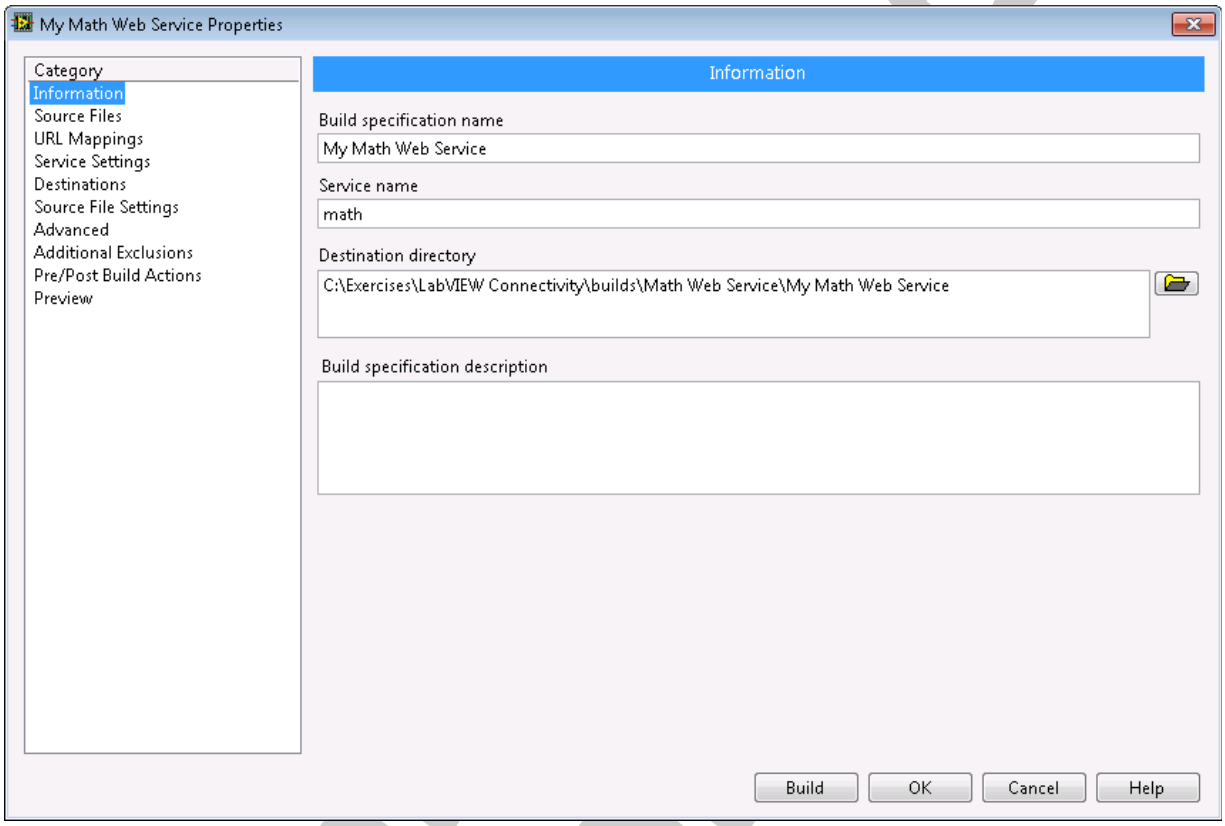


Figure 6-4. Information Category Settings for Web Service Build Specification

   – Enter `My Math Web Service` in the **Build specification name** field. This is the name that will be displayed in the project.

   – Enter `math` in the **Service name** field. This is used as the name of the service in URLs.

   – Accept the default **Destination directory**.

National Instruments
Not for Distribution

- ❑ Select the **Source Files** category.
  - Select **add.vi** in the Project Files section and click the arrow to move it to the Service VIs list.
  - In the **Configure RESTful VI** dialog box, change the **Output format** to **Text**, as shown in Figure 6-5, and then click the **OK** button.

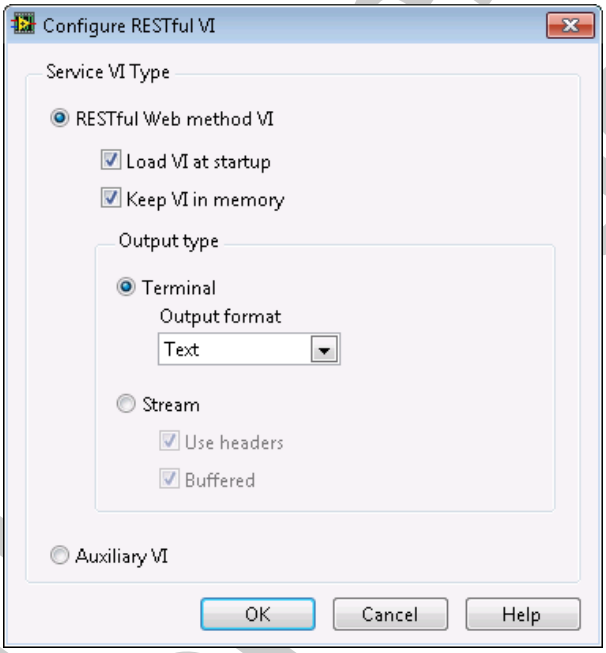


Figure 6-5. Configure RESTful VI Dialog Box

National Instruments Not for Distribution

❑ Select the **URL Mappings** category and notice that an entry has been created for your `add.vi` as shown in Figure 6-6.

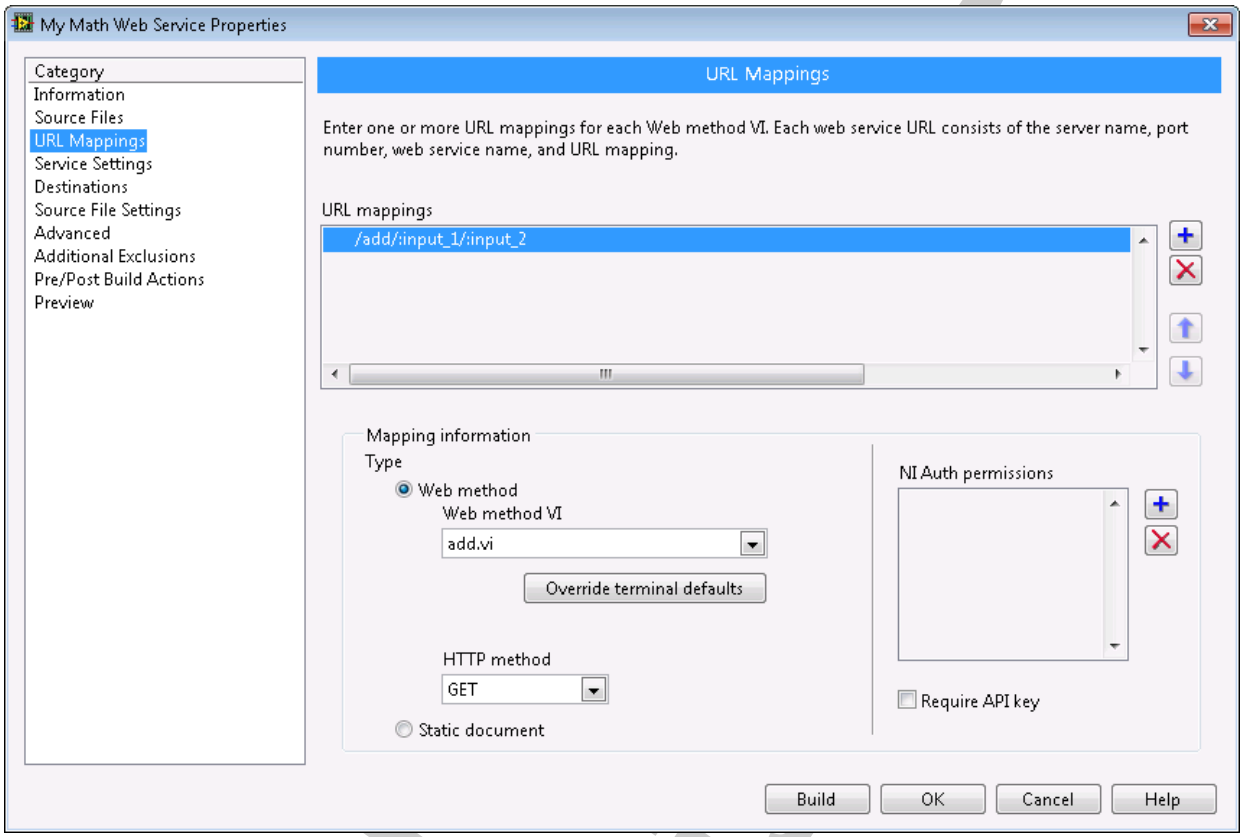


Figure 6-6. URL Mappings Configuration for Add VI

Note `:input_1` and `:input_2` are placeholders in the URL for values that will be passed to the controls of the same name in your `add.vi`.

– Set the order of the inputs to be `:input_1` and then `:input_2`. If they are not in that order in your URL mapping, triple-click the text, make the change, and then press <Enter>.

9. Click the **Build** button to build your Web service.

❑ When the build is complete, click the **Done** button in the **Build status** dialog box.

10. Ensure that the LabVIEW Web Application Server is running.

❑ Select **Tools»Options** in the **Project Explorer** window.

❑ Select the **Web Server** category and click the **Configure Web Application Server** button.

National Instruments Not for Distribution

This launches a Web browser and displays the Web Server Configuration page. You can also browse to `http://localhost:3580` and select the Web Server Configuration Extension from the icons on the left-hand side.

❑ Ensure that the port for the Application Web Server is set to `8080` and the **Enabled** box is checked as shown in Figure 6-7.

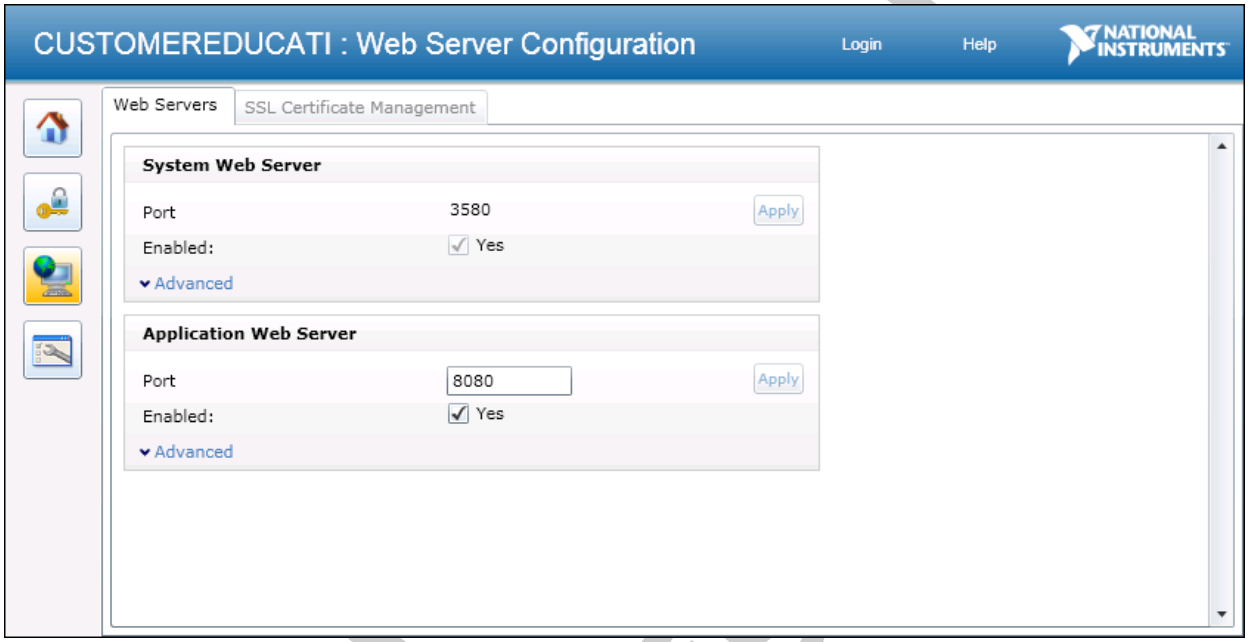


Figure 6-7. Web Server Configuration

❑ Close the Web browser and click the **OK** button in the **Options** dialog box.

National Instruments Not for Distribution

11. Deploy your Web service.

❑ In the LabVIEW Project Explorer, right-click the **My Math Web Service** build specification and select **Deploy** from the shortcut menu, as shown in Figure 6-8.

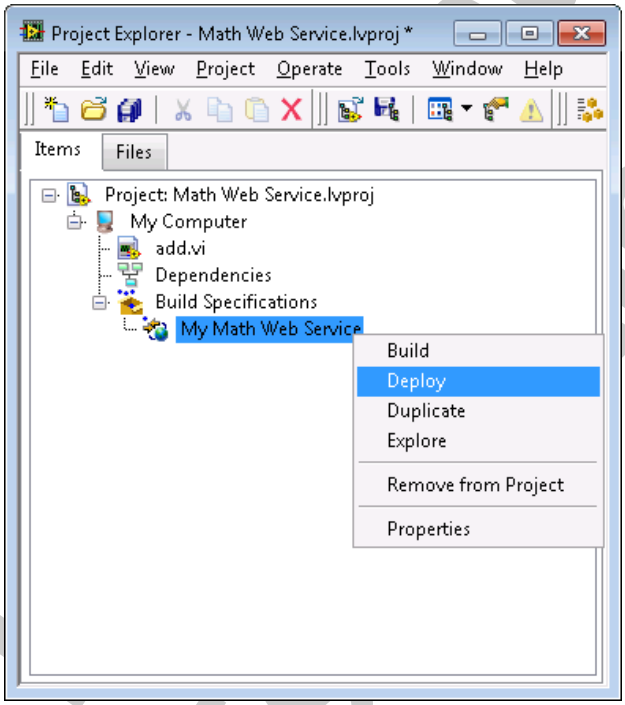


Figure 6-8. Deploy Web Service

❑ When the **Deployment Progress** dialog box shows that deployment completed successfully, click the **Close** button.

National Instruments
Not for Distribution

Testing

1. Open a Web browser and point it to your Web service at the following URL:

   `http://localhost:8080/math/add/3.25/4.5`

   ❑ This invokes the Web service, which runs your `add.vi`, passing the values of 3.25 and 4.5. The sum, 7.75 should display as shown in Figure 6-9.

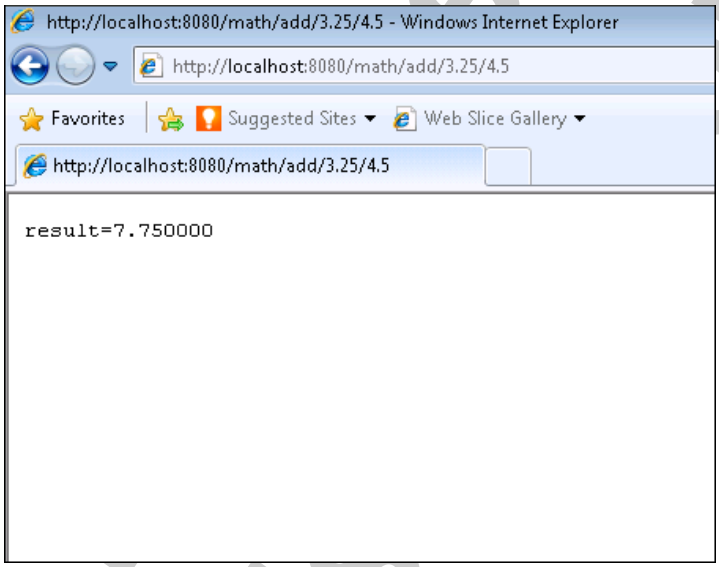


Figure 6-9. Result of Add Method Invoked by Math Web Service

2. Try passing different numbers, both integers and floating point numbers.
3. Try passing a different number of arguments: one, zero, or more than two.
4. Close your Web browser.

End of Exercise 6-1

National Instruments Not for Distribution

Exercise 6-2 Accept POST Data from an HTML Form

Goal

Add a new version of the `/math/add` method that allows the user to fill out an HTML form with the two numbers to be summed. You will learn to add static documents to a Web service.

Scenario

You want to add an HTML form to your service and invoke the `/math/add` method from the form.

Design

You will edit an HTML form using Windows Notepad or another text editor and add this to a new Virtual Folder in your LabVIEW Project.

You will configure the Web Service Build Specification to include the folder and its contents (the HTML file) when building the Web service.

You will write a new VI called `add-by-post.vi` that uses VIs from the Web Service palette to read variables from the HTML form to get the two numbers that are to be summed, then update the Build Specification with a new version of the `/math/add` method that is invoked in response to a POST instead of the GET that you used in Exercise 6-1.

Finally, you will load the form in a Web browser and test that the new service works as expected.

Implementation

1. Open `<Exercises>\LabVIEW Connectivity\Web Services\Math Web Service.lvproj`.
2. Add a virtual folder to the project for your html documents.
   - ❑ Right-click **My Computer** and select **New»Virtual Folder** from the shortcut menu. Name the folder `html` to refer to the static HTML documents it is intended to contain.
   - ❑ Right-click the `html` folder and select **Add»File** from the shortcut menu.
   - ❑ Browse to `<Exercises>\LabVIEW Connectivity\Web Services` and select `test_form.html` and click the **Add File** button.

National Instruments Not for Distribution

3. View the HTML form that will invoke the `/math/add` method.

   ❑ Open `test_form.html` in Internet Explorer or another Web browser.

   ❑ Select **View»Source**.

Tip If the menu bar is not displayed in Internet Explorer, press the <Alt> key.

   ❑ Look at the `<form action="/math/add-by-post">` tag.

      – `"/math/add-by-post"` is a relative path that includes the Web service name (`math`) and Web method (`add-by-post`).

      – The Web `method` is set to POST.

   ❑ Look at the `<input type="text" name="a">` tag.

      – `"a"` is a variable name and the variable is text. In LabVIEW, text is a string type.

   ❑ Verify that your project resembles Figure 6-10.

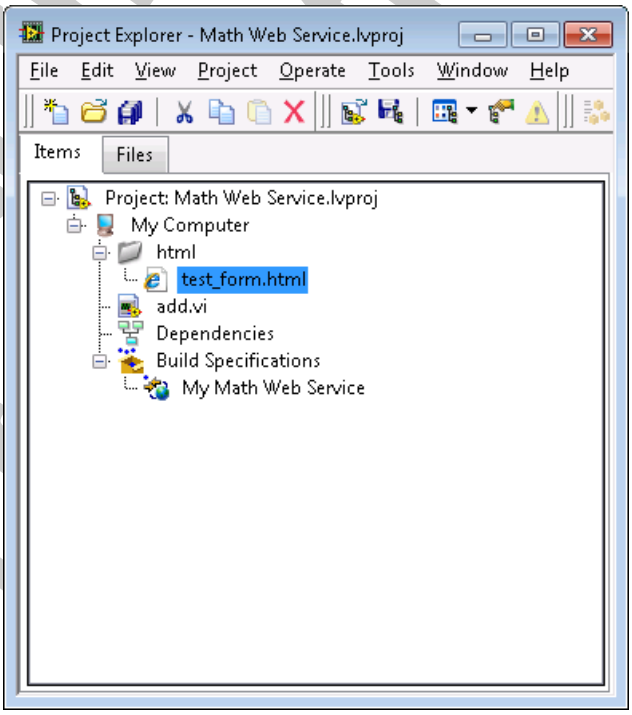


Figure 6-10. Math Web Service Project with Auto-Populated HTML Folder

National Instruments Not for distribution

4. Create a VI to interact with your HTML form.

   ❑ Add a new VI to your Math Web Service LabVIEW Project and save it to `<Exercises>\LabVIEW Connectivity\Web Services\add-by-post.vi`.

5. Create the front panel as shown in Figure 6-11 using the following items.

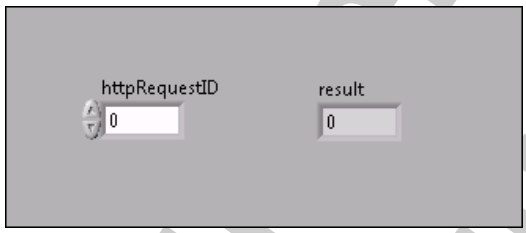


Figure 6-11. Add-by-post VI Front Panel

   ❑ Numeric Control—Set the representation to U32 and label the control `httpRequestID` (the name and data type must match exactly).

   ❑ Numeric Indicator—Set representation to DBL and label the indicator `result`

   ❑ Configure the connector pane with an input terminal connected to the **httpRequestID** control and an output terminal connected to the **result** indicator, as shown in Figure

6. Create the block diagram as shown in Figure 6-12 using the following items:

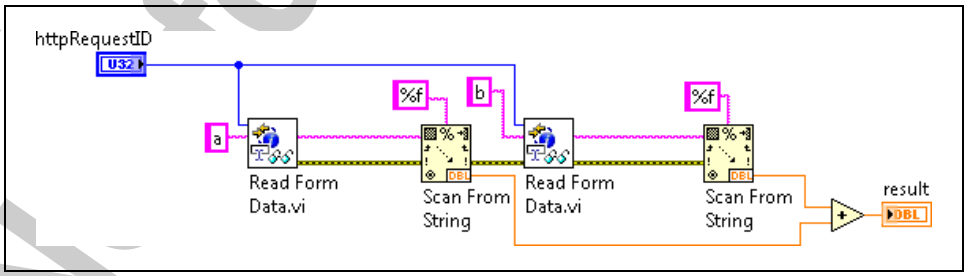


Figure 6-12. Add-by-post VI Block Diagram

National Instruments Not for Distribution

- ❑ Two Read Form Data VIs
  - – Wire String Constants with values of `a` and `b` to the **key** inputs of VIs.

    This tells the VI to read the values of the variables with these names from the HTML form you created earlier.
- ❑ Two Scan From String functions
  - – Wire String Constants with a value of `%f` to the **format string** inputs of each of these functions.
- ❑ Add function
- ❑ Complete the block diagram wiring as shown in Figure 6-12.

7. Save and close `add-by-post.vi`.
8. Add the `add-by-post.vi` and `test_form.html` to your build specification.
   - ❑ Open **My Math Web Service** build specification in the Project Explorer.
     - – Select the **Source Files** category and add the **add-by-post.vi** to the **Service VIs** list.
     - – In the **Configure RESTful VI** dialog box, change the **Output format** to **Text**, and then click the **OK** button.
     - – Move the **html** folder to the **Always Included** list.
   - ❑ Select the **URL Mappings** category.
     - – Click the plus sign (**+**) to the right of the **URL mappings** list.
     - – Enter `/html` to add a new URL mapping for the static directory.
     - – With the **/html** mapping selected, select **Static document** in the **Mapping information** section.
     - – Select the **/add-by-post** mapping and change the **HTTP method** to **POST**.

National Instruments Not for Distribution

9. Add the `test_form.html` file to the build specification as shown in Figure 6-13.

   ❑ Select the **Destinations** category.

   – Click the plus sign (**+**) at the bottom of the **Destinations** list and enter `html` in the **Destination label** field.

   – Place a checkmark in the **Preserve disk hierarchy** checkbox.

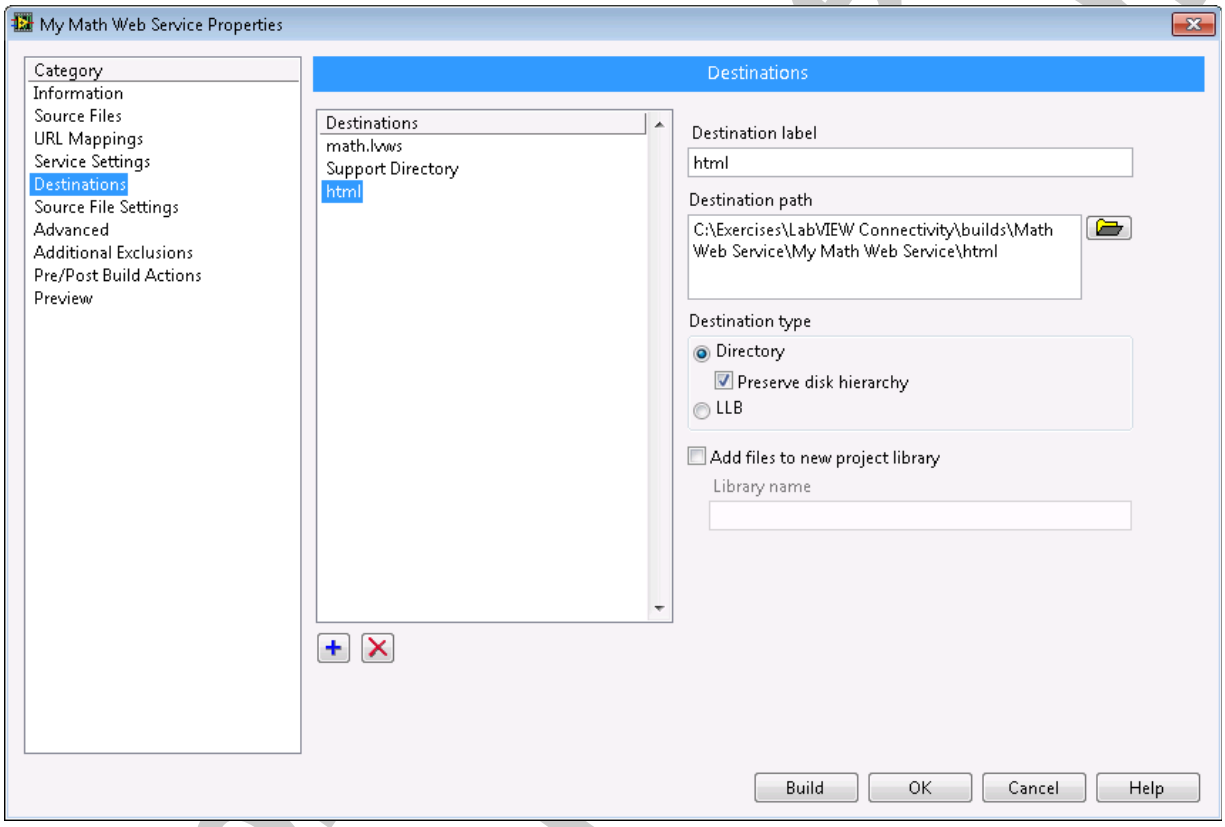


Figure 6-13. Web Services Destinations Settings

10. Place all files inside the `html` folder into the newly-created html destination.

    ❑ Select the **Source File Settings** category.

    – Under **Project Files**, select the **html** folder and place a check in the **Set destination for all contained items** checkbox.

    – From the pull-down menu, select **html**.

    – Place a checkmark in the **Set save settings for all contained items** checkbox and the **Use default save settings** checkbox.

National Instruments Not for Students

– The completed form should resemble Figure 6-14.

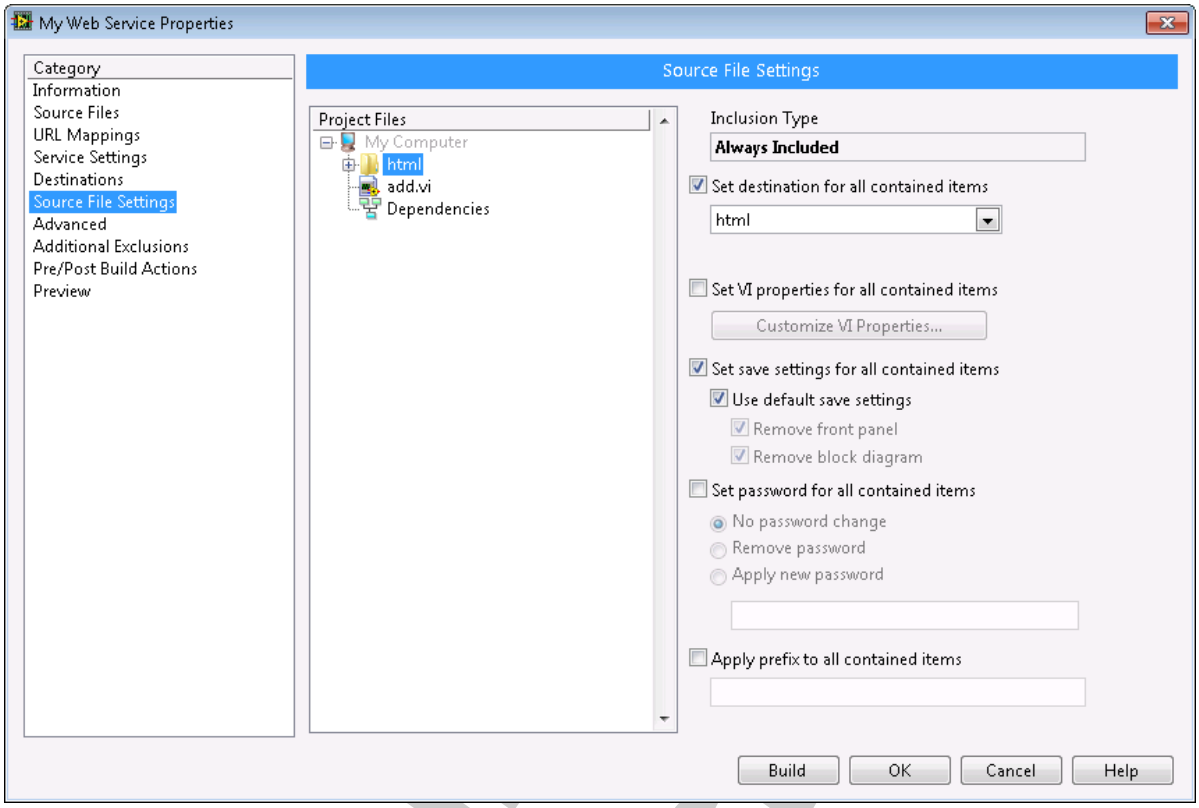


Figure 6-14. Source File Settings

11. Build and deploy the Web service.

Testing

1. Open your Web browser and load the HTML form you created earlier.

❑ Enter the following URL:

```
http://localhost:8080/math/html/test_form.html
```

National Instruments
Not for Distribution

2. Enter numbers into each field and click the **Submit Query** button.

   Your new VI should be invoked and the result displayed, as shown in Figure 6-15 and Figure 6-16.

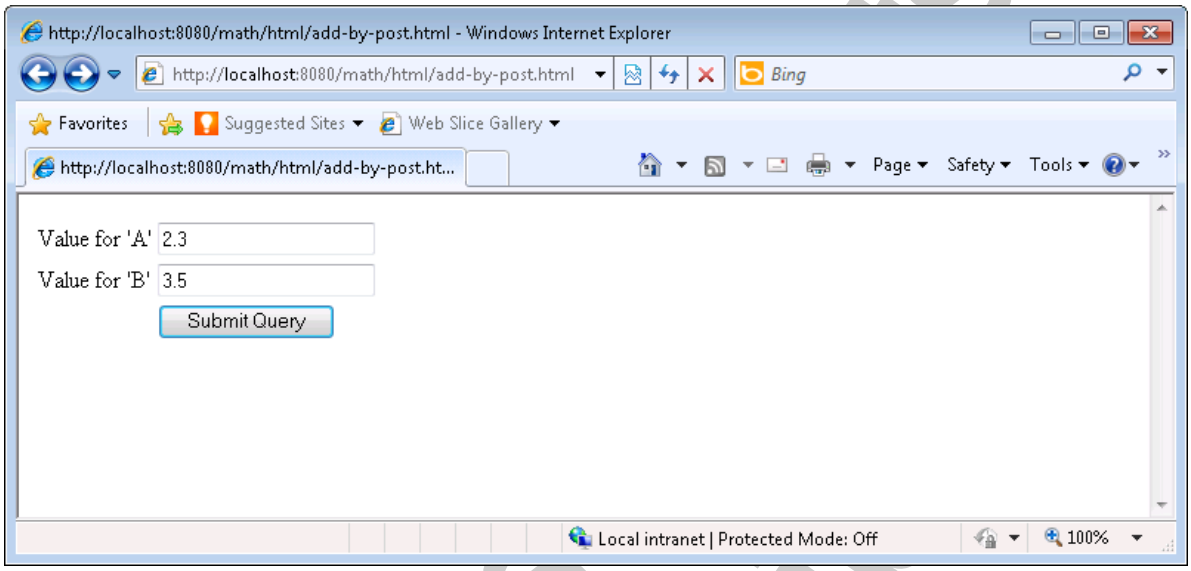


Figure 6-15. Add-by-post Submit Query

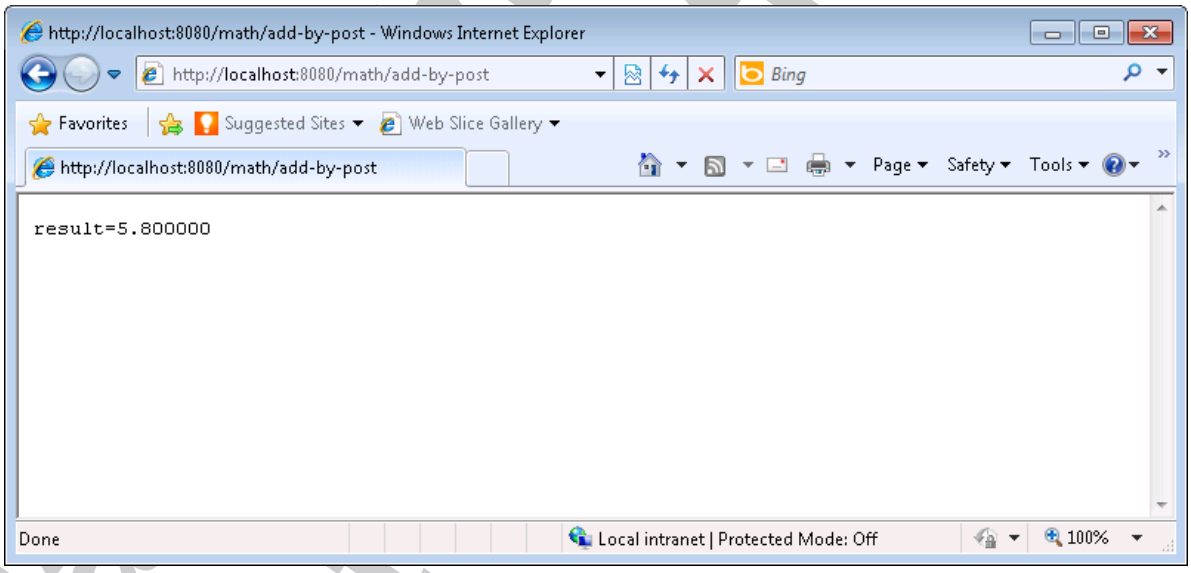


Figure 6-16. Add-by-post Result

3. Close your Web browser.

End of Exercise 6-2

National Instruments Corporation Not for Distribution

Exercise 6-3 Generate Image with Web Method

Goal

Add a new `/math/get_waveform_data` method to your previously created Web service. This method will generate an image of an indicator and download it as a `.png` file. This will demonstrate the creation and use of Web services that use Stream Output rather than Terminal Output. Stream Output mode puts the Web service developer in complete control of the data sent back from the Web service.

Scenario

You want to display the image of a waveform graph control on a Web page.

Design

You will create a VI that accepts a set point via an input terminal, assigns that set point to a slider control, captures the image of that slider control, converts it into a `.png` file, and downloads the image to the browser with the image/png MIME type.

Then, you will invoke the method from a Web browser and see the image displayed set to the correct value.

Implementation

1. Open `<Exercises>\LabVIEW Connectivity\Web Services\Math Web Service.lvproj`.
2. Create a VI to display an image in a Web browser.
   - ❑ Add a new VI in your Math Web Service LabVIEW Project and save it to `<Exercises>\LabVIEW Connectivity\Web Services\get_waveform_data.vi`.

National Instruments Not for Distribution

3. Create the front panel as shown in Figure 6-17, using the following items.

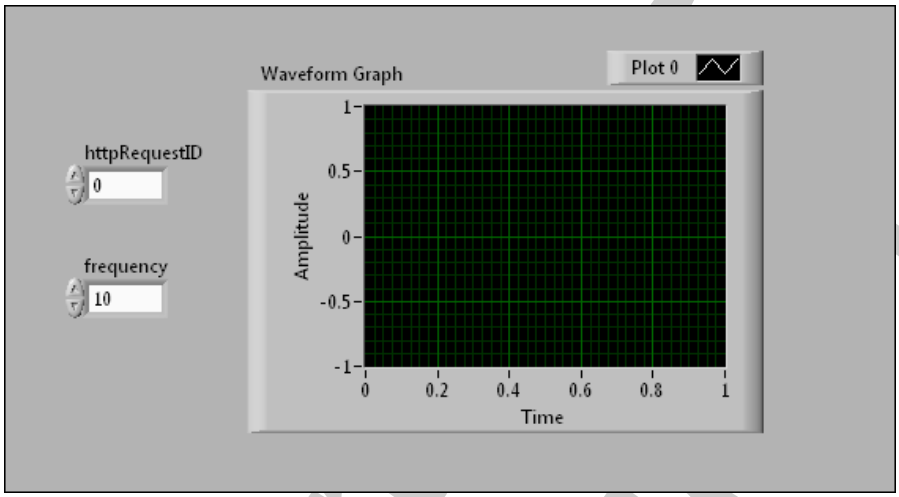


Figure 6-17. get_waveform_data VI Front Panel

- ❑ Numeric Control—set the representation to U32 and label it `httpRequestID` (the name and data type must match exactly)
- ❑ Numeric Control—set the representation to DBL and label it `frequency`
- ❑ Waveform Graph
- ❑ Configure the connector pane with input terminals connected to the **httpRequestID** and **frequency** controls. Do not connect any output terminals.

4. Configure the block diagram as shown in Figure 6-18 using the following items.

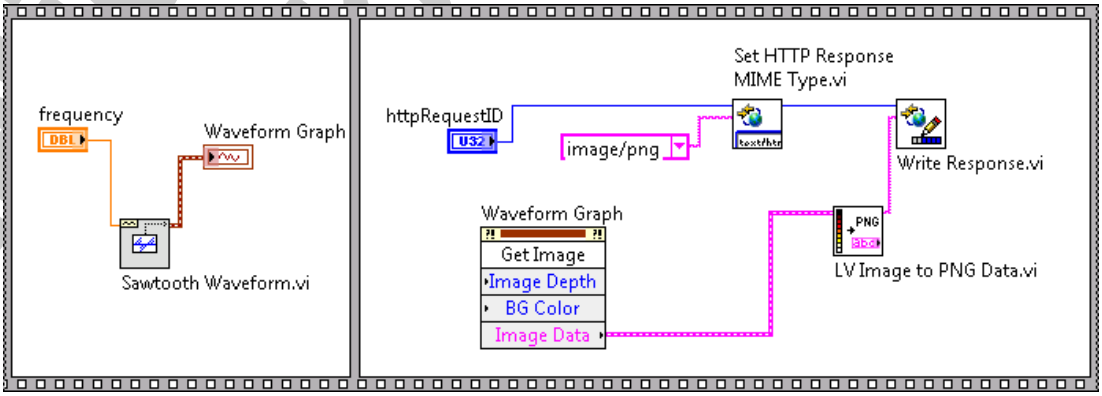


Figure 6-18. get_waveform_data Block Diagram

National Instruments Corporation
For Distribution

- ❑ Flat Sequence Structure with two frames

- ❑ Sawtooth Waveform VI

- ❑ Set HTTP Response MIME Type VI

- ❑ Write Response VI

- ❑ LV Image to PNG Data VI
- ❑ In the first frame of the Flat Sequence Structure, wire the **frequency** terminal to the **frequency** input of the Sawtooth Waveform VI.
- ❑ Wire the Sawtooth Waveform **signal out** to the **Waveform Graph** terminal.
- ❑ In the second frame, wire the **httpRequestID** control terminal to the Set HTTP Response MIME Type VI.
- ❑ Create a constant from the **MIME type** terminal and set it to `image/png`.
- ❑ Wire the **httpRequestID** to the **httpRequestID** input terminal on the Write Response VI.
- ❑ Right-click the **Waveform Graph** indicator terminal and select **Create»Invoke Node»Get Image** from the shortcut menu. Place the Invoke Node in the second frame.
  - Wire the **Image Data** output of the Waveform Graph Invoke Node to **image data** input of the LV Image to PNG Data VI.
- ❑ Wire the **png data** output of the LV Image to PNG Data VI to the **response string** input of the Write Response VI.

5. Save and close the VI.
6. Add the `get_waveform_data.vi` to the build specification.
   - ❑ Open the My Math Web Service build specification in the Project Explorer.

National Instruments
Not for Distribution

❑ Select the **Source Files** category.

– Add the **get_waveform_data.vi** to the list of Service VIs.

– In the **Configure RESTful VI** dialog box, set the **Output Type** to **Stream**, and then click the **OK**.

❑ Select the **URL Mappings** category

– Notice that the build specification has automatically created a new URL mapping for the new VI, complete with a `:frequency` placeholder to map a parameter from the URL to the frequency control.

– Ensure that **HTTP method** is set to **GET**. The completed page should resemble Figure 6-19.

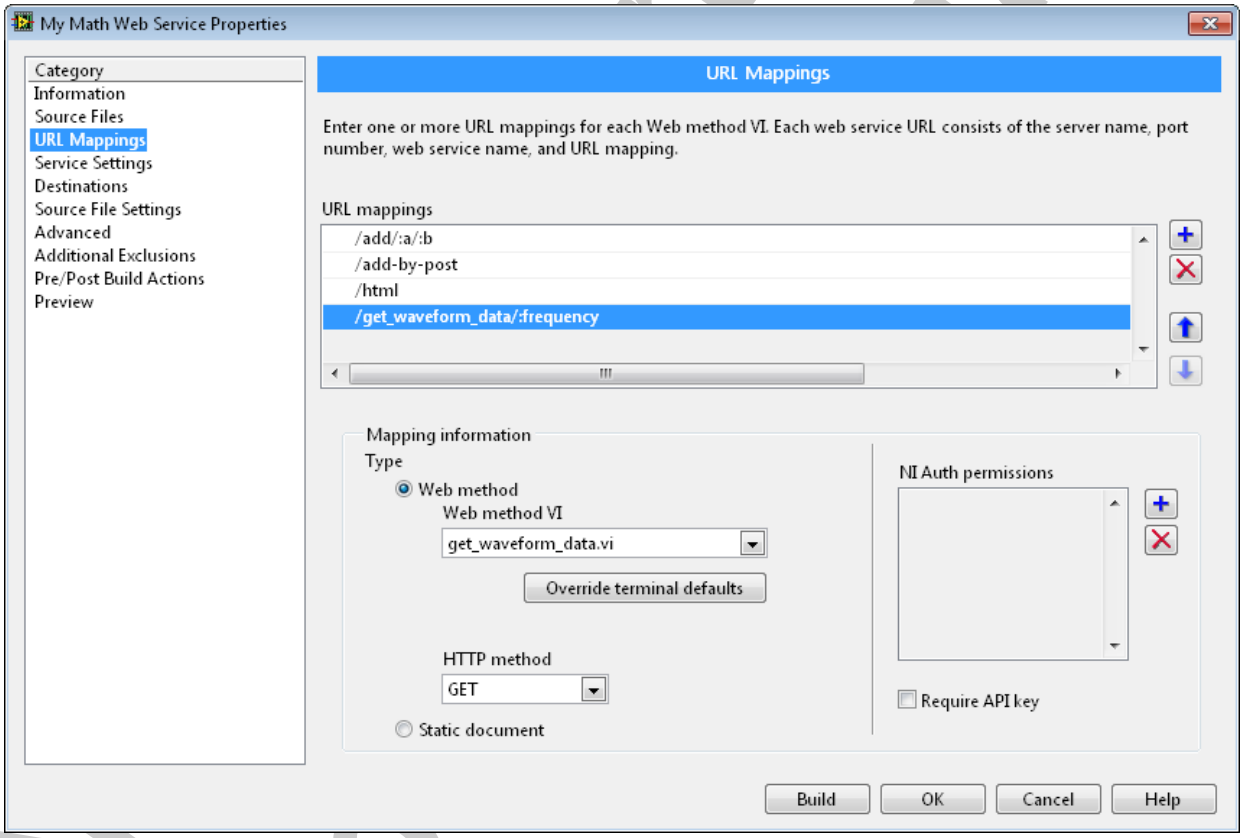


Figure 6-19. URL Mappings Settings for get_set_point_image

7. Build and deploy the Web service.

National Instruments Confidential

Testing

1. Open your Web browser and enter the URL for your new Web method:

   ```
   http://localhost:8080/math/get_waveform_data/10
   ```

2. Notice the waveform graph displayed in your Web browser.

3. Enter different values as the final parameter of the URL and notice that the waveform graph displays the results.

End of Exercise 6-3

National Instruments
Not for Distribution

Exercise 6-4 Create an HTTP Client in LabVIEW

Goal

Use the HTTP Client VIs in LabVIEW to communicate with the Web methods you created in Exercise 6-3.

Scenario

You have two methods that belong to a Web service running on your local server. Create a single VI that uses the HTTP Client API to communicate with these Web methods.

Design

Using the GET VI and POST VI from the HTTP Client API to communicate with two Web methods in the Web service mathWS. You will perform the following actions.

- Use the GET and POST VIs to establish communication with the math Web service.
- Use VIs from the String palette to create a UI for changing the parameters sent to the Web Server from the HTTP VIs

Inputs and Outputs

Table 6-1. HTTP Client Call Add Method Inputs and Outputs

Type	Name	Properties	Default Value
Enum	HTTP Method	U16	GET
String Control	Root URL String	—	`http://localhost:8080/math/`
Numeric Control	Input 1	DBL	0
Numeric Control	Input 2	DBL	0
String Indicator	Response Body	—	Empty

National Instruments Not for Distribution

Implementation

1. Open a blank VI and save it as `HTTP Client Call Add Method.vi` in the `<Exercises>\LabVIEW Connectivity\HTTP Client` directory.
2. Create the front panel shown in Figure 6-20 using the following items.

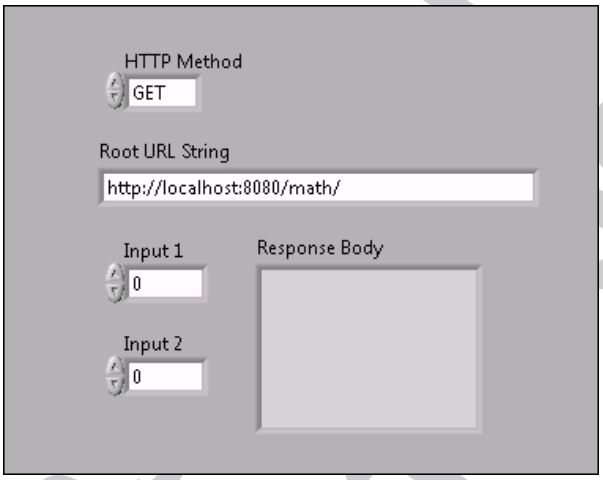


Figure 6-20. HTTP Client Call Add Method Front Panel

- ❑ Enum—labeled `HTTP Method` with two items, `GET` and `POST`
- ❑ String Control—labeled `Root URL String`
  - Enter `http://localhost:8080/math/` in the control and then right-click the control and select **Data Operations»Make Current Value Default**
- ❑ Two Numeric Controls—Set the representation to DBL
  - Label one control `Input 1`
  - Label the second control `Input 2`
- ❑ String Indicator—labeled `Response Body`

National Instruments
Not for Distribution

In the following steps, you complete the block diagram shown in Figure 6-21 and Figure 6-22.

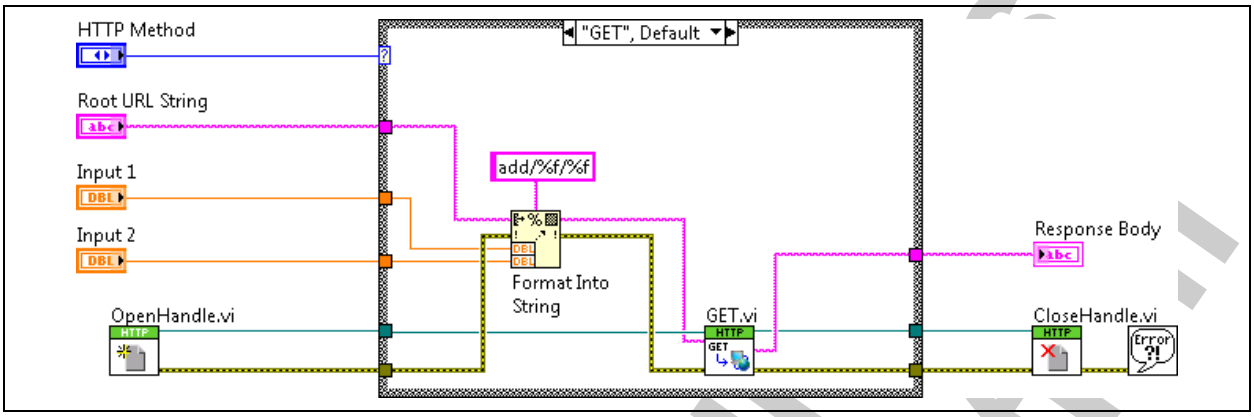


Figure 6-21. HTTP Client Call Add Method Block Diagram—GET Case

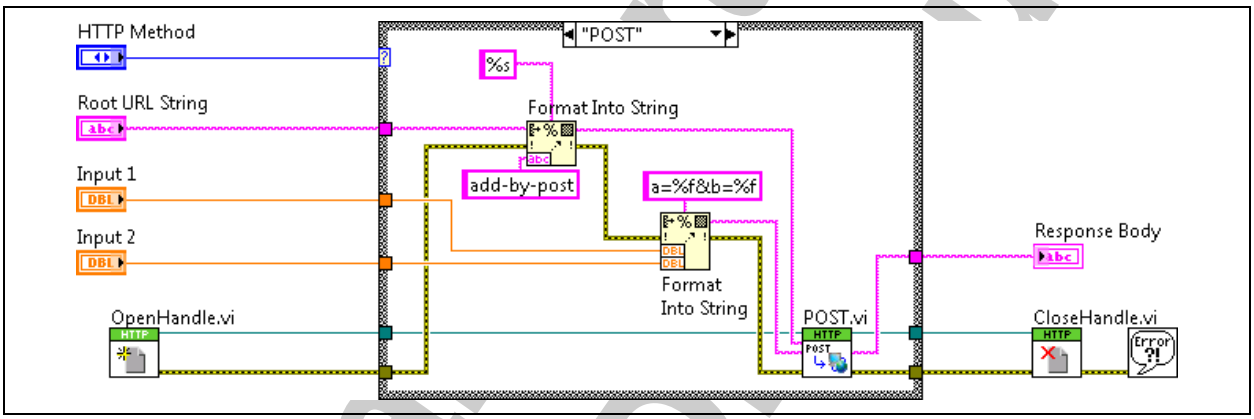


Figure 6-22. HTTP Client Call Add Method Block Diagram—POST Case

3. Complete the GET case.

❑ Place a Case structure on the block diagram.

– Wire the **HTTP Method** enum to the **Case Selector** of the Case structure.

– Select the **GET** case.

❑ Place the following VIs on the block diagram.

– OpenHandle.vi

– GET.vi

– CloseHandle.vi

– Simple Error Handler.vi

National Instruments Not for Distribution

- ❑ Place a Format Into String function in the Case structure.
  - – Expand the function to add a second input parameter.
  - – Wire a constant with a value of `add/%f/%f` into the **format string** input.
- ❑ Wire the GET case as shown in Figure 6-21.

4. Complete the POST case.
   - ❑ Select the **POST** case.
   - ❑ Place a POST.vi in the Case structure.
   - ❑ Place a Format Into String function in the POST case.
     - – Wire a String Constant with a value of `add-by-post` to **input 1** of the first Format Into String function.
     - – Wire a String Constant of `%s` to the **format string** input.
   - ❑ Place a second Format Into String function in the POST case.
     - – Expand the function to add a second input parameter.
     - – Wire a String Constant with a value of `a=%f&b=%f` into the **format string** input.
   - ❑ Wire the POST case as shown in Figure 6-22.

Testing

1. Run the VI.
2. Enter values in the **Input 1** and **Input 2** controls.
3. Click the **Run** button.
4. Change the HTTP Method to POST and run the VI again.
5. Exchange IP addresses with a partner and change the Root URL String to `http://<partner's IP address>/math/`, and then run the VI again.

National Instruments Not for Distribution

Notes

National Instruments
Not for Distribution

Notes

National Instruments
Not for Distribution