Chapter 2
Using IVI with Visual C++

The Environment

Microsoft Visual C++ is a software development environment for the C++ programming language and is available as part of Microsoft Visual Studio. Visual C++ allows you to create, debug, and execute conventional applications as well as applications that target the .NET Framework.

Example Requirements

- Visual C++
- Microsoft Visual Studio 2010
- IVI-COM: Agilent 34401A IVI-COM, Version 1.2.2.0, October 2008 (from Agilent Technologies); or
- IVI-C: Agilent 34401A IVI-C, Version 4.4, July 2010 (from National Instruments)
- Agilent IO Libraries Suite 16.1
- National Instruments IVI Compliance Package version 4.0 or later

Download and Install the Driver

If you have not already installed the driver, go to the vendor Web site and follow the instructions to download and install it.

Since Visual C++ supports both IVI-COM and IVI-C drivers, this example is written two ways, first to show how to use an IVI-COM driver in Visual C++, and second to show how to use an IVI-C driver in Visual C++.

***Note**: If you do not install the appropriate instrument driver, the project will not build because the referenced files are not included in the program. If you need to download and install a driver, you do not need to exit Visual Studio. Install the driver and continue with your program.*

Using IVI-COM in C++

The following sections show how to get started with an IVI-COM driver in Visual C++.

Create a New Project and Import the Driver Type Libraries

To use an IVI Driver in a Visual C++ program, you must provide the path to the type libraries it uses.

1. Launch Visual Studio 2010 and create a Visual C++ Win32 Console Application with the name “IviDemo”. Use the default settings.

   Note: *The program already includes some required code, including the standard header file:*

   ```
   #include stdafx.h.
   ```

2. In Solution Explorer, right click on the “IviDemo” project node and click on “Properties”.This will open the “IviDemo Property Pages” dialog.
3. In the tree view on the left of the dialog, expand “Configuration Properties”, then click on “VC++ Directories”.
4. Locate the “Include Directories” row in the right hand pane and click on the drop down icon in the column that contains the directory paths. Click on “<Edit…>”.
5. Add the following two entries to your path.

   The first entry will point to the default directory for IVI drivers. On 32-bit Windows, use:

   ```
   “C:\Program Files\IVI Foundation\IVI\Bin”
   ```

   On 64-bit Windows, use:

   ```
   “C:\Program Files (x86)\IVI Foundation\IVI\Bin”
   ```

   ```
   The second entry points to the VISA DLL that many drivers require:
   ```

   ```
   “$(VXIPNPPATH)VisaCom”
   ```

   Note: *The second entry will point to the correct VISA COM directory regardless of whether you are operating with 32-bit or 64-bit Windows.*
6. Click OK twice to save changes and exit the “IviDemo Property Pages” dialog.

Import COM Type Libraries

COM type libraries must be imported before they can be accessed. To import the type libraries, type the following statements following the header file reference:

```
#import <IviDriverTypeLib.dll> no_namespace
#import <IviDmmTypeLib.dll> no_namespace
#import <GlobMgr.dll> no_namespace
#import <Ag34401.dll> no_namespaceInitialize
```

Note: *The `#import` statements access the driver type libraries used by the Agilent 34401 DMM. The `no_namespace` attribute allows the code to access the interfaces in the type libraries from the global namespace.*

At this point the Visual C++ editor may flag the #import statements as errors. To fix the errors, select “Rebuild Solution” from the Build menu.

Initialize COM

1 Initialize the COM library, and check for errors. Add the following lines at the beginning of the _tmain function (immediately before the `return` statement):

```
HRESULT hr = ::CoInitialize(NULL);
if (FAILED(hr)) exit(1);
```

2 To close the COM library before exiting, type the following line at the end of your code, right before the return line:

```
::CoUninitialize();
```

Create an Instance of the Driver

To create an instance of the driver, type

```
{
    IIviDmmPtr dmm(__uuidof(Agilent34401));
}
```

Note: *This creates a smart pointer that provides easy access to the COM object.*

You are now ready to write the program to control the simulated instrument.

Initialize the Instrument

You can now write the main constructs for your program.

Below the smart pointer statement, type

```
dmm->Initialize("GPIB::23", false, true,
"simulate=true");
```

Note*: As soon as you type `->`, Intellisense displays options and helps ensure you use correct syntax and values.*

Configure the Instrument

To set the range to 1.5 volts and resolution to 0.001 volts, type

```
dmm->Configure(IviDmmFunctionDCVolts, 1.5, 0.001);
```

Set the Trigger Delay

To set the trigger delay to 0.01 seconds, type

```
dmm->Trigger->Delay = 0.01;
```

Set the Reading Timeout/Display the Reading

Create a variable to represent the reading, make a reading with a timeout of 1 second (1000 milliseconds), and display the result to the console:

```
double reading = dmm->Measurement->Read(1000);
wprintf(L"Reading: %g\n", reading);
```

Error Checking

To catch errors in the code, activate error checking.

1. Surround the preceding statements with a try block. Add the following lines before the call to the Initialize method:

```
try
{
```

2. Process errors in a catch block. Add the following lines after the call to the wprintf method that follows the Read method:

```
}
catch (_com_error e)
{
        wprintf(L"Error: %s", e.ErrorMessage());
}
```

Close the Session

Close out the instance of the driver and free resources. Add the following line after the closing bracket of the catch block:

```
dmm->Close();
```

View the Results

Prompt the user to press any key to continue. Without these lines, the console window would immediately close before the user could view the information that was written to it. Add the following lines immediately before the return statement:

```
printf("\nDone - Press any key to exit");
getchar();
```

Complete Source Code

The complete source code for the IviDemo.cpp file is shown below:

```
// IviDemo.cpp: Defines the entry point for the console
application.
//
#include "stdafx.h"

#import <IviDriverTypeLib.dll> no_namespace
#import <IviDmmTypeLib.dll> no_namespace
#import <GlobMgr.dll> no_namespace
#import <Ag34401.dll> no_namespace

int _tmain(int argc, _TCHAR* argv[])
{
    HRESULT hr = ::CoInitialize(NULL);
    if (FAILED(hr))  exit(1);

    {
        IIviDmmPtr dmm(__uuidof(Agilent34401));

        try
        {
            dmm->Initialize("GPIB::23", false, true,
"simulate=true");
            dmm->Configure(IviDmmFunctionDCVolts, 1.5,
0.001);
            dmm->Trigger->Delay = 0.01;
            double reading = dmm->Measurement->Read(1000);
            wprintf(L"Reading: %g\n", reading);
        }
        catch (_com_error e)
        {
            wprintf(L"Error: %s", e.ErrorMessage());
        }
        dmm->Close();
    }

    ::CoUninitialize();

    printf("\nDone - Press any key to exit");
```

```
        getchar();

        return 0;
}



{
    HRESULT hr;
    hr = CoInitialize(NULL);
    if(FAILED(hr))
           exit(1);
    {
           IAgilent34401Ptr dmm(__uuidof(Agilent34401));
           try{
dmm->Initialize("GPIB::23",false,true,"simulate=true");
                  dmm->DCVoltage->Configure(1.5, 0.001);
                  dmm->Trigger->Delay = 0.01;
           double reading = dmm->Measurement->Read(1000);
                  wprintf(L"Reading: %g\n", reading);
           }
           catch(_com_error e){
                  wprintf(L"Error: %s\n", e.ErrorMessage);
           }
           dmm->Close();
    }
    CoUninitialize();
    return 0;
}
```

Build and Run the Application

Build your application and run it to verify it works properly.

1. From the Build menu, select “Build”, and click “Rebuild Solution”.
2. From the Debug menu, select “StartDebugging” to run the application.

Using IVI-C in Visual C++

The following sections show to get started with IVI-C in Visual C++.

Create a New Project and Import the Driver Type Libraries

To use an IVI-C Driver in a Visual C++ program, you must provide paths to the header files and libraries it uses.

1. Launch Visual Studio 2010 and create a Visual C++ Win32 Console Application with the name “IviDemo2”. Use the default settings.

   Note: *The program already includes some required code, including the standard header file:*

   ```
   #include "stdafx.h"
   ```

2. In Solution Explorer, right click on the “IviDemo2” project node and click on “Properties”. This will open the “IviDemo2 Property Pages” dialog.
3. In the tree view on the left of the dialog, expand “Configuration Properties”, then click on “VC++ Directories”.
4. Locate the “Include Directories” row in the right hand pane and click on the drop down icon in the column that contains the directory paths. Click on “<Edit…>”.
5. Add the following two entries to your path. The first entry will point to the default directory for IVI drivers.

   On 32-bit Windows, use:

   ```
   "C:\Program Files\IVI Foundation\IVI\Include"
   ```

   On 64-bit Windows, use:

   ```
   "C:\Program Files (x86)\IVI Foundation\IVI\Include"
   ```

   The second entry points to the VISA DLL that many drivers require:

   ```
   "$(VXIPNPPATH)WinNT\include"
   ```

   Note: *The second entry will point to the correct VISA directory regardless of whether you are operating with 32-bit or 64-bit Windows.*
6. Locate the “Library Directories” row in the right hand pane and click on the drop down icon in the column that contains the directory paths. Click on “<Edit…>”.
7. Add the following two entries to your path. The first entry will point to the default directory for IVI drivers.

   On 32-bit Windows, use:

   ```
   "C:\Program Files\IVI Foundation\IVI\Lib\msc"
   ```

   On 64-bit Windows, use:

```
"C:\Program Files (x86)\IVI Foundation\IVI\Lib\msc"
```

The second entry points to the VISA DLL that many drivers require:

```
"$(VXIPNPPATH)WinNT\lib\msc"
```

Note: *The second entry will point to the correct VISA directory regardless of whether you are operating with 32-bit or 64-bit Windows.*

8 Next, expand "Linker" in the tree view on the left of the "IviDemo2 Property Pages" dialog, then click on "Input".

9 Locate the "Additional Dependencies" row in the right hand pane and click on the drop down icon in the column that contains the list of .lib files. Click on "<Edit…>".

10 Add the following library file to the list:

```
"hp34401a.lib"
```

11 Click OK twice to save changes and exit the "IviDemo2 Property Pages" dialog.

Include Driver Header

To add the hp34401a instrument driver header file to your program, type the following statement following the existing header file reference:

```
#include "hp34401a.h"
```

Select "Rebuild Solution" from the Build menu.

Declare Variables

Declare the program variables. Add the following lines at the beginning of the _tmain function (immediately before the `return` statement):

```
ViSession session;
ViStatus error = VI_SUCCESS;
ViReal64 reading;
```

Define Error Checking

Next define error checking for your program. First you will define a macro to catch the errors. It is better to define it once at the beginning of the program that to add the logic to each of your program statements. After the `#include` statements, type the following lines:

```
#ifndef checkErr
#define checkErr(fCall) \
if (error = (fCall), (error = (error < 0) ? error :
VI_SUCCESS)) \
{goto Error;} else error = error
#endif
```

Next add code to handle any errors that occur. Add the following lines before the return statement:

```
        Error:
            if (error != VI_SUCCESS)
            {
                ViChar errStr[2048];
                hp34401a_GetError (session, &error, 2048,
        errStr);
                printf ("Error!", errStr);
    }
```

Note: *Including error handling in your programs is good practice. This code checks for errors in your program.*

Initialize the Instrument

To initialize the instrument, add the following Initialize with Options function right after the variable declarations you added in the previous section:

```
checkErr( hp34401a_InitWithOptions
("GPIB::23::INSTR",VI_FALSE, VI_TRUE,
                                        "Simulate = 1",
&session));
```

This initializes the instrument with the following parameters:

- GPIB0::23::INSTR is the Resource Name (instrument at GPIB address 23).
- VI_FALSE indicates that an ID Query should not be performed by this function.
- VI_TRUE resets the device .
- Simulate=1 is the Options parameter that sets the driver to simulation mode.
- &session assigns the Instrument Handle to the variable “session” defined above.Configure the Instrument

Configure the Instrument

To set the range to 1.5 volts and resolution to 0.001 millivolts, type:

```
checkErr( hp34401a_ConfigureMeasurement (session,
HP34401A_VAL_DC_VOLTS, 1.5, 0.001));
```

Set the Trigger and Trigger Delay

To set the trigger source to immediate and the trigger delay to 0.01 seconds, type:

```
checkErr( hp34401a_ConfigureTrigger (session,
HP34401A_VAL_IMMEDIATE,
```

```
0.01));
```

Set the Reading Timeout/Display the Reading

To take a reading from the instrument and to set the reading timeout to 1 second (1000 ms) type, and display the result using the printf function:

```
checkErr( hp34401a_Read (session, 1000, &reading);
printf ("Reading = %f", reading);
```

Note: *The Read function takes a reading from the instrument and assigns the result to the variable "reading" defined above.*

Close the Session

To close out the instance of the driver and free resources, add the following lines immediately before the return statement:

```
If (session)
hp34401a_Close(session);
```

View the Results

Prompt the user to press any key to continue. Without these lines, the console window would immediately close before the user could view the information that was written to it. Add the following lines immediately before the return statement:

```
printf("\nDone - Press any key to exit");
getchar();
```

Complete Source Code

The complete source code for the IviDemo2.cpp file is shown below:

```
// IviDemo2.cpp : Defines the entry point for the console
application.
//

#include "stdafx.h"
#include <hp34401a.h>

#ifndef checkErr
#define checkErr(fCall) \
if (error = (fCall), (error = (error < 0) ? error :
VI_SUCCESS)) \
```

```
{goto Error;} else error = error
#endif

int _tmain(int argc, _TCHAR* argv[])
{
    ViSession session;
    ViStatus error = VI_SUCCESS;
    ViReal64 reading;
    checkErr( hp34401a_InitWithOptions ("GPIB::23::INSTR",
VI_FALSE, VI_TRUE,
                                        "Simulate=1", &session));
    checkErr( hp34401a_ConfigureMeasurement (session,
HP34401A_VAL_DC_VOLTS,
                                             1.5, 0.0001));
    checkErr( hp34401a_ConfigureTrigger (session,
HP34401A_VAL_IMMEDIATE, 0.01));
    checkErr( hp34401a_Read (session, 1000, &reading));
    printf ("Reading = %f", reading);

    Error:
        if (error != VI_SUCCESS)
        {
            ViChar errStr[2048];
            hp34401a_GetError (session, &error, 2048, errStr);
            printf ("Error!", errStr);
        }

        if (session)
            hp34401a_close (session);

    printf("\nDone - Press any key to exit");
    getchar();

    return 0;
}
```

Build and Run the Application

Build your application and run it to verify it works properly.

1. From the Build menu, select “Build”, and click “Rebuild Solution”.
2. From the Debug menu, select

3 “StartDebugging” to run the application.

Microsoft® and Visual Studio® are registered trademarks of Microsoft Corporation in the United States and/or other countries.