Getting Started with IVI Drivers

Your Guide to Using IVI with PAWS

Version 1.0

© Copyright IVI Foundation, 2009
All rights reserved

The IVI Foundation has full copyright privileges of all versions of the IVI Getting Started Guide. For persons wishing to reference portions of the guide in their own written work, standard copyright protection and usage applies. This includes providing a reference to the guide within the written work. Likewise, it needs to be apparent what content was taken from the guide. A recommended method in which to do this is by using a different font in italics to signify the copyrighted material.

Contents

Chapter 1 Introduction 4

Purpose 4

Why Use an Instrument Driver? 4

Why IVI? 5

Why Use an IVI Driver? 7

Flavors of IVI Drivers 8

Shared Components 8

Download and Install IVI Drivers 8

Familiarizing Yourself with the Driver 9

Examples 9

Chapter 2 Using IVI with PAWS 11

The Environment 11

Example Requirements 11

Download and Install the Driver 11

Prepare the PAWS Environment 12

Add the WCEM Interface Functions 14

Connect to the IVI-COM Driver 15

Build the Project 20

Prepare the Run-Time System Environment 20

Load and Run the Project 21

Further Information 21

Chapter 1
Introduction

Purpose

Welcome to ***Getting Started with IVI Drivers: Your Guide to Using IVI with PAWS***. This guide introduces key concepts about IVI drivers and shows you how to create a short program to perform a measurement. The guide is part of the IVI Foundation's series of guides, ***Getting Started with IVI Drivers***.

Getting Started with IVI Drivers is intended for individuals who write and run programs to control test-and-measurement instruments. Each guide focuses on a different programming environment. As you develop test programs, you face decisions about how you communicate with the instruments. Some of your choices include Direct I/O, VXI*plug&play* drivers, or IVI drivers. If you are new to using IVI drivers or just want a quick refresher on the basics, ***Getting Started with IVI Drivers*** can help.

Getting Started with IVI Drivers shows that IVI drivers can be straightforward, easy-to-use tools. IVI drivers provide a number of advantages that can save time and money during development, while improving performance as well. Whether you are starting a new program or making improvements to an existing one, you should consider the use of IVI drivers to develop your test programs.

So consider this the "hello, instrument" guide for IVI drivers. If you recall, the "hello world" program, which originally appeared in *Programming in C: A Tutorial,* simply prints out "hello, world." The "hello, instrument" program performs a simple measurement on a simulated instrument and returns the result. We think you'll find that far more useful.

Why Use an Instrument Driver?

To understand the benefits of IVI drivers, we need to start by defining instrument drivers in general and describing why they are useful. An instrument driver is a set of software routines that controls a programmable instrument. Each routine corresponds to a programmatic operation, such as configuring, writing to, reading from, and triggering the instrument. Instrument drivers simplify instrument control and reduce test program development time by eliminating the need to learn the programming protocol for each instrument.

Starting in the 1970s, programmers used device-dependent commands for computer control of instruments. But lack of standardization meant even two digital multimeters from the same manufacturer might not use the same commands. In the early 1990s a group of instrument manufacturers developed Standard

Commands for Programmable Instrumentation (SCPI). This defined set of commands for controlling instruments uses ASCII characters, providing some basic standardization and consistency to the commands used to control instruments. For example, when you want to measure a DC voltage, the standard SCPI command is "`MEASURE:VOLTAGE:DC?`".

In 1993, the VXI*plug&play* Systems Alliance created specifications for instrument drivers called VXI*plug&play* drivers. Unlike SCPI, VXI*plug&play* drivers do not specify how to control specific instruments; instead, they specify some common aspects of an instrument driver. By using a driver, you can access the instrument by calling a subroutine in your programming language instead of having to format and send an ASCII string as you do with SCPI. With ASCII, you have to create and send the instrument the syntax "`MEASURE:VOLTAGE:DC?`", then read back a string, and build it into a variable. With a driver you can merely call a function called MeasureDCVoltage( ) and pass it a variable to return the measured voltage.

Although you still need to be syntactically correct in your calls to the instrument driver, making calls to a subroutine in your programming language is less error prone. If you have been programming to instruments without a driver, then you are probably all too familiar with hunting around the programming guide to find the right SCPI command and exact syntax. You also have to deal with an I/O library to format and send the strings, and then build the response string into a variable.

Why IVI?

The VXI*plug&play* drivers do not provide a common programming interface. That means programming a Keithley DMM using VXI*plug&play* still differs from programming an Agilent DMM. For example, the instrument driver interface for one may be ke2000_read while another may be hp34401_get or something even farther afield. Without consistency across instruments manufactured by different vendors, many programmers still spent a lot of time learning each individual driver.

To carry VXI*plug&play* drivers a step (or two) further, in 1998 a group of end users, instrument vendors, software vendors, system suppliers, and system integrators joined together to form a consortium called the Interchangeable Virtual Instruments (IVI) Foundation. If you look at the membership, it's clear that many of the foundation members are competitors. But all agreed on the need to promote specifications for programming test instruments that provide better performance, reduce the cost of program development and maintenance, and simplify interchangeability.

For example, for any IVI driver developed for a DMM, the measurement command is *IviDmmMeasurement.Read*, regardless of the vendor. Once you learn how to program the commands specified by IVI for the instrument class, you can use any vendor's instrument and not need to relearn the commands. Also commands that are common to all drivers, such as *Initialize* and *Close*, are identical regardless of

the type of instrument. This commonality lets you spend less time hunting around the help files and programming an instrument, leaving more time to get your job done.

That was the motivation behind the development of IVI drivers.The IVI specifications enable drivers with a consistent and high standard of quality, usability, and completeness. The specifications define an open driver architecture, a set of instrument classes, and shared software components. Together these provide consistency and ease of use, as well as the crucial elements needed for the advanced features IVI drivers support: instrument simulation, automatic range checking, state caching, and interchangeability.

The IVI Foundation has created IVI class specifications that define the capabilities for drivers for the following eight instrument classes:

Class	IVI Driver
Digital multimeter (DMM)	IviDmm
Oscilloscope	IviScope
Arbitrary waveform/function generator	IviFgen
DC power supply	IviDCPwr
Switch	IviSwtch
Power meter	IviPwrMeter
Spectrum analyzer	IviSpecAn
RF signal generator	IviRFSigGen

IVI Class Compliant drivers usually also include capability that is not part of the IVI Class. It is common for instruments that are part of a class to have numerous functions that are beyond the scope of the class definition. This may be because the capability is not common to all instruments of the class or because the instrument offers some control that is more refined than what the class defines.

IVI also defines custom drivers. Custom drivers are used for instruments that are not members of a class. For example, there is not a class definition for network analyzers, so a network analyzer driver must be a custom driver. Custom drivers provide the same consistency and benefits described below for an IVI driver, except interchangeability.

IVI drivers conform to and are documented according to the IVI specifications and usually display the standard IVI logo.

Why Use an IVI Driver?

Why choose IVI drivers over other possibilities? Because IVI drivers can increase performance and flexibility for more intricate test applications. Here are a few of the benefits:

Consistency – IVI drivers all follow a common model of how to control the instrument. That saves you time when you need to use a new instrument.

Ease of use – IVI drivers feature enhanced ease of use in popular Application Development Environments (ADEs). The APIs provide fast, intuitive access to functions. IVI drivers use technology that naturally integrates in many different software environments.

Quality – IVI drivers focus on common commands, desirable options, and rigorous testing to ensure driver quality.

Simulation – IVI drivers allow code development and testing even when an instrument is unavailable. That reduces the need for scarce hardware resources and simplifies test of measurement applications. The example programs in this document use this feature.

Range checking – IVI drivers ensure the parameters you use are within appropriate ranges for an instrument.

State caching – IVI drivers keep track of an instrument's status so that I/O is only performed when necessary, preventing redundant configuration commands from being sent. This can significantly improve test system performance.

Interchangeability – IVI drivers enable exchange of instruments with minimal code changes, reducing the time and effort needed to integrate measurement devices into new or existing systems. The IVI class specifications provide syntactic interchangeability but may not provide behavioral interchangeability. In other words, the program may run on two different instruments but the results may not be the same due to differences in the way the instrument itself functions.

Flavors of IVI Drivers

To support all popular programming languages and development environments, IVI drivers provide either an IVI-C or an IVI-COM (Component Object Model) API. Driver developers may provide either or both interfaces, as well as wrapper interfaces optimized for specific development environments.

Although the functionality is the same, IVI-C drivers are optimized for use in ANSI C development environments; IVI-COM drivers are optimized for environments that support the Component Object Model (COM). IVI-C drivers extend the VXI*plug&play* driver specification and their usage is similar. IVI-COM drivers provide easy access to instrument functionality through methods and properties.

All IVI drivers communicate to the instrument through an I/O Library. Our examples use the Virtual Instrument Software Architecture (VISA), a widely used standard library for communicating with instruments from a personal computer.

Shared Components

To make it easier for you to combine drivers and other software from various vendors, the IVI Foundation members have cooperated to provide common software components, called IVI Shared Components. These components provide services to drivers and driver clients that need to be common to all drivers. For instance, the IVI Configuration Server enables administration of system-wide configuration.

Important! You must install the IVI Shared Components before an IVI driver can be installed.

The IVI Shared Components can be downloaded from vendors' web sites as well as from the IVI Foundation Web site.

To download and install shared components from the IVI Foundation Web site:

1. Go to the IVI Foundation Web site at http://www.ivifoundation.org.
2. Locate Shared Components.
3. Choose the IVI Shared Components msi file for the Microsoft Windows Installer package or the IVI Shared Components exe for the executable installer.

Download and Install IVI Drivers

After you've installed Shared Components, you're ready to download and install an IVI driver. For most ADEs, the steps to download and install an IVI driver are identical. For the few that require a different process, the relevant ***Getting Started with IVI Drivers*** guide provides the information you need.

IVI Drivers are available from your hardware or software vendor's web site or by linking to them from the IVI Foundation web site.

To see the list of drivers registered with the IVI Foundation, go to http://www.ivifoundation.org.

Familiarizing Yourself with the Driver

Although the examples in ***Getting Started with IVI Drivers*** use a DMM driver, you will likely employ a variety of IVI drivers to develop test programs. To jumpstart that task, you'll want to familiarize yourself quickly with drivers you haven't used before. Most ADEs provide a way to explore IVI drivers to learn their functionality. In each IVI guide, where applicable, we add a note explaining how to view the available functions. In addition, browsing an IVI driver's help file often proves an excellent way to learn its functionality.

Examples

As we noted above, each guide in the ***Getting Started with IVI Drivers*** series shows you how to use an IVI driver to write and run a program that performs a simple measurement on a simulated instrument and returns the result. The examples demonstrate common steps using IVI drivers. Where practical, every example includes the steps listed below:

- Download and Install the IVI driver– covered in the Download and Install IVI Drivers section above.
- Determine the VISA address string – Examples in ***Getting Started with IVI Drivers*** use the simulate mode, so we chose the address string **GPIB0::23::INSTR,** often shown as GPIB::23. If you need to determine the VISA address string for your instrument and the ADE does not provide it automatically, use an IO application, such as National Instruments Measurement and Automation Explorer (MAX) or Agilent Connection Expert.
- Reference the driver or load driver files – For the examples in the IVI guides, the driver is the **Agilent 34401A IVI-COM Specific Driver, Version 1.1.0.11, March 2006 (from Agilent Technologies)** or the **Agilent 34401A IVI-C Specific driver, Version 4.1, October 2006 (from National Instruments).**
- Create an instance of the driver in ADEs that use COM – For the examples in the IVI guides, the driver is the **Agilent 34401A (IVI-COM) or HP 34401 (IVI-C)**.
- Write the program:
    - Initialize the instrument – Initialize is required when using any IVI driver. Initialize establishes a communication link with the instrument and must be called before the program can do anything with the instrument. We set reset to **true**, ID query to **false**, and simulate to **true**.

      Setting reset to true tells the driver to initially reset the instrument. Setting the ID query to false prevents the driver from verifying that the connected instrument is the one the driver was written for. Finally, setting simulate to true tells the driver that it should not attempt to connect to a physical instrument, but use a simulation of the instrument.

- Configure the instrument – We set a range of **1.5 volts** and a resolution of **0.001 volts (1 millivolt)**.
- Access an instrument property – We set the trigger delay to **0.01 seconds**.
- Set the reading timeout – We set the reading timeout to **1000 milliseconds (1 second).**
- Take a reading
- Close the instrument – This step is required when using any IVI driver, unless the ADE explicitly does not require it. We close the session to free resources.

Important! Close may be the most commonly missed step when using an IVI driver. Failing to do this could mean that system resources are not freed up and your program may behave unexpectedly on subsequent executions.

- Check the driver for any errors.
- Display the reading.

Note: *Examples that use a console application do not show the display.*

Now that you understand the logic behind IVI drivers, let's see how to get started.

Chapter 2
Using IVI with PAWS

The Environment

PAWS Developer's Studio gives you the capability to edit, compile, modify, debug, document, execute, and simulate the test procedures developed in Abbreviated Test Language for All Systems (ATLAS) in the Windows NT/2000/XP Platform environment. PAWS supports a full range of the most commonly used ATLAS Language subsets. A PAWS Toolkit can modify the ATLAS Language subset to meet a particular Automatic Test Equipment (ATE) configuration. Its output can be executed on the associated Debugging PAWS/RTS run-time system or can be translated to run on your unique run-time system.

Creation of a PAWS project typically consists of three main steps:

- Prepare the PAWS environment
- Connect to the instrument driver
- Prepare the run-time system environment and run the project

Example Requirements

- TYX PAWS Studio 1.34.6
- Microsoft Visual Studio
- Agilent 34401A IVI-COM, Version 1.1.0.11, March 2006 (from Agilent Technologies)
- Agilent IO Libraries Suite 14.2

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. You can also refer to Chapter 1, Download and Install IVI Drivers, for instructions.

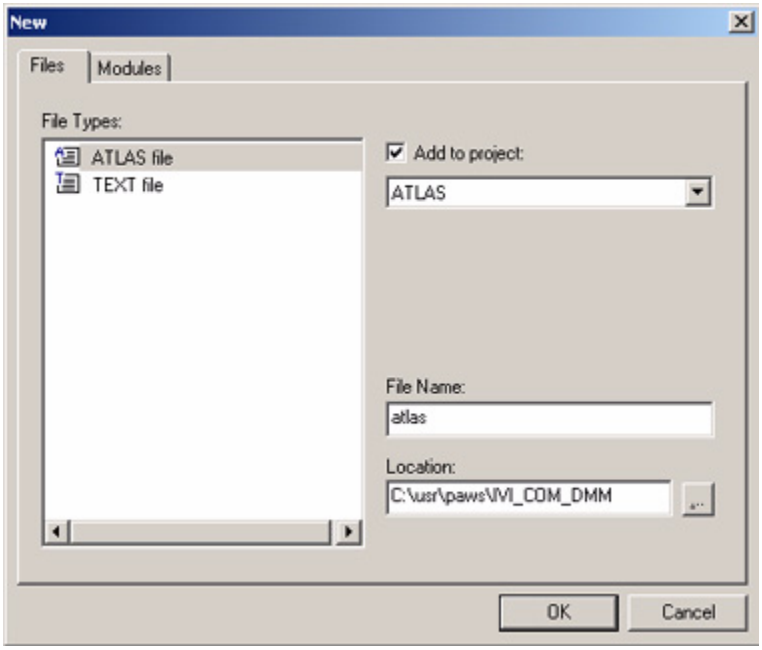
This example uses an IVI-COM driver. PAWS supports both IVI-COM and IVI-C drivers.

***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 PAWS Studio. Install the driver and continue with your program.*

Prepare the PAWS Environment

To use an IVI Driver in PAWS, you must first prepare the PAWS environment.

1. Launch PAWS Studio and create a new project. In the New PAWS Project dialog, enter a project name and directory for the project. Click OK.
2. To add a new file to the ATLAS Project Module, from the Main Menu select File and click New File. The New dialog appears.
3. Select ATLAS file. Check the option to add the file to the project.
4. Enter **atlas** in the File Name field. Click OK. The atlas.atl file appears.



5. Insert the following code in the atlas.atl file and save it:

```
001000 BEGIN, ATLAS PROGRAM 'IVI-COM DMM'                    $
001010 REQUIRE, 'DMM-DC-VOLTS', SENSOR(VOLTAGE), DC SIGNAL,
            CONTROL, VOLTAGE RANGE -300 V TO 300 V,
            MAX-TIME RANGE 1 SEC TO 5 SEC,
           CNX HI LO                                        $
C$
 001110 DECLARE, VARIABLE, 'MEASURED' IS DECIMAL             $
```

```
C$
E02000 OUTPUT, C'\LF\ATLAS PROGRAM STARTS HERE\LF\'          $
C$
 002200 VERIFY, (VOLTAGE INTO 'MEASURED'),
                DC SIGNAL USING 'DMM-DC-VOLTS',
          NOM 0 V UL 0.5 V LL -0.5 V,
          VOLTAGE RANGE -0.5 V TO 0.5 V,
                MAX-TIME 5 SEC,
               CNX HI X20-2 LO X20-3                        $
 002300 OUTPUT, C'DC VOLT MEASUREMENT 1 = ', 'MEASURED', C'
VDC'  $
C$
 999000 OUTPUT, C'\LF\ATLAS PROGRAM ENDS HERE\LF\'           $
C$
 999999 TERMINATE, ATLAS PROGRAM 'IVI-COM DMM'               $
```

6 To add a Device Database Module, from the Main Menu select File and click New Module.

7 Select DEVICEDB and click OK.

8 To add a new file to the DEVICEDB Project Module, from the Main Menu select File and click New File. The New dialog appears.

9 Select DEVICE file. Check the option to add the file to the project.

10 Enter **dmm** in the File Name field. Click OK.

11 In the PAWS Project window, select the file dmm.ddb.

12 Insert the following code in the dmm.dbb file and save it:

```
begin dev DMM;

      cnx hi DMM-Hi, lo DMM-Lo;

   begin FNC = 101; ** DC Voltage Measurement
         sensor(voltage)dc signal;
         control
         {
               voltage range -300 v to 300 v continuous;
```

```
        }
    end; ** DC Voltage Measurement


    end; *end dmm
```

Note: *To understand the file contents, refer to the Paws Studio online help.*

13 To build the project, from the Main Menu select Build and click Rebuild All.

Add the WCEM Interface Functions

The WCEM interface C++ functions are invoked by the ATLAS code. These C++ functions include the IVI-COM calls to control the instrument.

1 From the Main Menu, select File and click New Module. The New dialog appears.

2 Select CEM and enter **WCEM** in the Module Name field. Click OK.

3 From the Main Menu, select View and click CEM Wizard. The CEM Wizard dialog appears.

4 Click on the Advanced tab. The CEM Wizard Advanced Settings dialog appears.

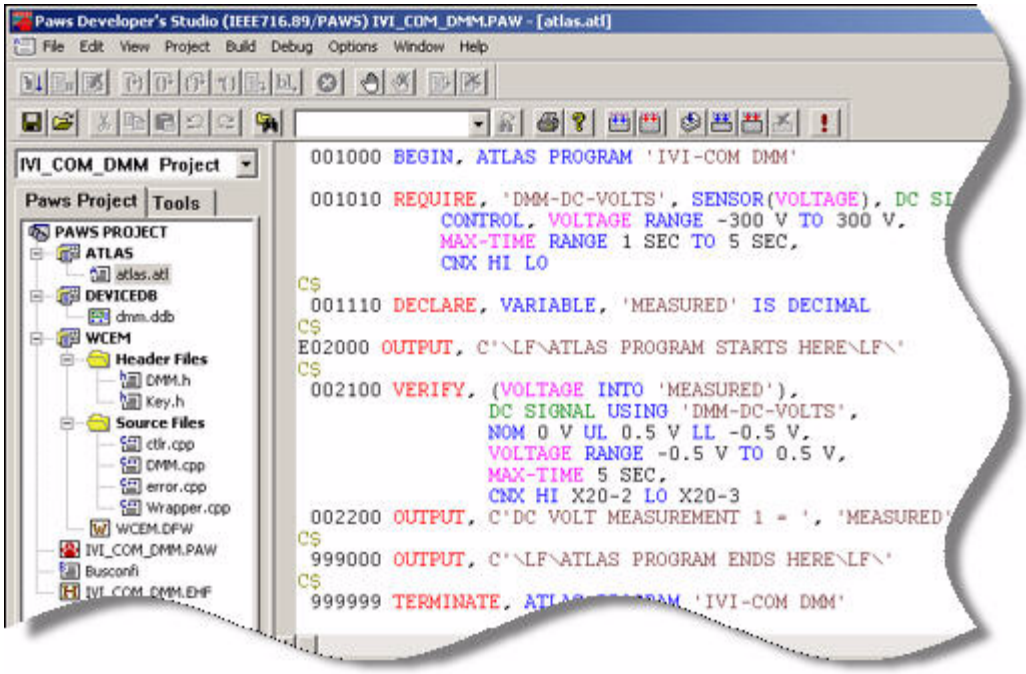
5 Check Open, Unload and Disable CEM Logging. Click OK. The CEM Wizard dialog appears.

6 Right-click on DMM and click on Add Interface Function. The Add Function(s) to Action(s) dialog appears.

7 For Setup, select Setup in the Action(s) dialog and enter **DMM_Setup** in the User Function Name(s) field. Click OK.

8 For Fetch and Init, select the appropriate action in the Action(s) dialog and enter **DMM_Fetch** and **DMM_Init** in the User Function Name(s) field. Click OK.

Note: *This will add several files to your WCEM module: ctrl.cpp, DMM.cpp, Error.cpp, Wrapper.cpp, and Key.h. ctrl.cpp includes the doOpen and doUnload functions. Dmm.cpp includes the three DMM_ functions.*



9 To add a new *.h file called DMM.h to include the code necessary to reference the driver header files and librairies , right-click on the WCEM module.

10 Select Add New File to Module. The New dialog appears.

11 Select CEM header file and check Add to Project.

12 Enter **DMM** in the File Name field. Click OK.

Connect to the IVI-COM Driver

1 From the PAWS Project window, select the DMM.h file. Insert the following code in the DMM.h file:

```
#ifndef __DMM_h__
#define __DMM_h__


#include <atlbase.h>


#include "Visacom_i.c"
#import "IviDriverTypeLib.dll" named_guids,
raw_interfaces_only, raw_native_types no_namespace
```

```
#import "IviDmmTypeLib.dll" named_guids,
raw_interfaces_only, raw_native_types no_namespace

#import "Agilent34401.dll" named_guids, raw_interfaces_only,
raw_native_types no_namespace


#endif
```

Note: *This header file allows access to the COM, IVI-specific, and device environments.*

2 From the PAWS Project window, select the ctrl.cpp file. Insert the following code in the ctrl.cpp file directly below #include "key.h" code:

```
#include "DMM.h"


extern CComPtr<IAgilent34401> driver;

extern CComPtr<IAgilent34401DCVoltage> pDMMDCVolt;

extern CComPtr<IAgilent34401Trigger> pDMMTrig;

extern CComPtr<IAgilent34401Measurement> pDMMData;


extern HRESULT hr;
```

3 From the PAWS Project window, select the DMM.cpp file. Insert the following code in the DMM.cpp file directly below #include "key.h" code:

```
#include "DMM.h"


CComPtr<IAgilent34401> driver;

CComPtr<IAgilent34401DCVoltage> pDMMDCVolt;

CComPtr<IAgilent34401Trigger> pDMMTrig;

CComPtr<IAgilent34401Measurement> pDMMData;


HRESULT hr;
```

4 To create the driver object and open an I/O session to the instrument with the Initialize method, select the ctrl.cpp file from the PAWS Project window.

Note: *To determine the syntax for the commands you want to use in your program, consult the driver help file.*

5 In the doOpen() interface function, insert the following code below the line Please insert your CEM driver code here:

```
hr = driver.CoCreateInstance(CLSID_Agilent34401);
if (FAILED(hr))
{
   Display("\033[30;41m Bad return from
   CoCreateInstance() method\033[m\n");
}

hr = driver->Initialize(CComBSTR("GPIB0::23::INSTR"),
                        VARIANT_FALSE,
                        VARIANT_TRUE,
                        CComBSTR("Simulate=TRUE"));
if (FAILED(hr))
{
   Display("\033[30;41m Bad return from Initialize()
   method\033[m\n");
}
```

Note: *doOpen is called for the first time when the run-time system loads the PAWS project. If you intend to use non-ATLAS modules in your ATLAS code, subsequent calls to doOpen will be made and you will need to execute the code above only once.*

6 To set the function for DC Voltage, the range to 1.5 volts, and the resolution to 1 millivolt, select the DMM.cpp file from the PAWS Project window.

7 In the `DMM_Setup()` interface function, insert the following code below the line Please insert your CEM driver code here ：

```
hr = driver->put_Function(Agilent34401FunctionDCVolts);
if (FAILED(hr))
{
   Display("\033[30;41m Bad return from
   put_Function(Agilent34401FunctionDCVolts)method\033[m\n"
   );
}
hr = driver->get_DCVoltage(&pDMMDCVolt);
```

```
if (FAILED(hr))
{
   Display("\033[30;41m Bad return from
   get_DCVoltage(&pDMMDCVolt) method\033[m\n");
}
hr = pDMMDCVolt->Configure(1.5, 0.001);
if (FAILED(hr))
{
   Display("\033[30;41m Bad return from >Configure(1.5,
   0.001) method\033[m\n");
}
```

8 To set the trigger delay to 0.01, select the DMM.cpp file from the PAWS Project window.

9 In the `DMM_Init()` interface function, insert the following code below the line Please insert your CEM driver code here：

```
hr = driver->get_Trigger(&pDMMTrig);
if (FAILED(hr))
{
   Display("\033[30;41m Bad return from
   get_Trigger(&pDMMTrig)  method\033[m\n");
}
hr = pDMMTrig->Configure(Agilent34401TriggerSourceImmediate,
0.01);
if (FAILED(hr))
{
   Display("\033[30;41m Bad return from
   Configure(Agilent34401TriggerSourceImmediate, 0.01)
   method\033[m\n");
}
hr = driver->get_Measurement(&pDMMData);
if (FAILED(hr))
{
   Display("\033[30;41m Bad return from
   get_Measurement(&pDMMData) method\033[m\n");
}
```

```
hr = pDMMData->Initiate();
if (FAILED(hr))
{
   Display("\033[30;41m Bad return from Initiate()
   method\033[m\n");
}
```

10 To set the reading timeout to 1 second, select the DMM.cpp file from the PAWS Project window.

11 In the `DMM_Fetch()` interface function, insert the following code below the line Please insert your CEM driver code here：

```
   double Data = 0;
DATUM *fdat;

hr = pDMMData->Fetch(1000, &Data);
if (FAILED(hr))
{

   Display("\033[30;41m Bad return from Fetch(1000,
   &Data) method\033[m\n");
}

fdat = FthDat();
DECDatVal(fdat, 0) = Data;
FthCnt(1);
```

Note: *The first argument in the* ***Fetch*** *method specifies how long to wait in the* ***Fetch*** *operation since it is possible that no data is available or the trigger event did not occur. The second parameter contains the actual reading or a value indicating that an over-range condition occurred.*

12 To conserve resources, you should close the driver session. From the PAWS Project window, select the ctrl.cpp file.

13 In the doUnload() interface function, insert the following code below the line // Please insert your CEM driver code here:

```
hr = driver->Close();
if (FAILED(hr))
{
   Display("\033[30;41m Bad return from Close()
   method\033[m\n");
```

```
}


driver.Release();
pDMMData.Release();
pDMMTrig.Release();
pDMMDCVolt.Release();
```

Note: *The function doUnload gets called only once, when the Paws project is unloaded from the run-time system.*

Build the Project

1. From the Main Menu, select Options and click on CEM. The WCEM dialog appears.
2. Select the Files tab and enter C:\Program Files\IVI\Include in the Include Path field. Click Apply.

   Note: *The path includes the Visacom_i.c file from the DMM.h header.*
3. From the Main Menu, select Build and click on Rebuild All. The Output area appears with a message that indicates the project was successfully built.

   Note: *Most of the IVI-COM driver methods return HRESULT value, which is used in this example across the entire driver implementation for error handling. Error handling code is recommended as you develop a PAWs application. In our example, the error handling code begins with if (FAILED(hr)) and ends with the } after the Display line for most of the methods.*

Prepare the Run-Time System Environment

1. In the Project Workspace area, double-click on the Busconfi. The DMM name in a Busconfi file corresponds to the name defined in the dmm.ddb file following the `begin dev` statement.
2. Set the Listen Address MLA and Talk Address MTA to the device address of 23.
3. The Bus number must correspond to the Channel number.

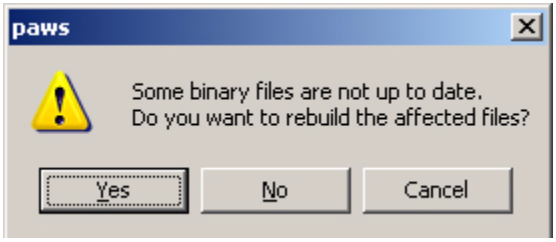
   The Busconfi code should appear as below:

```
;       IEEE-488 Bus Configuration File -


"Channel"        1


DMM BUS 1       MLA 23       MTA 23
```

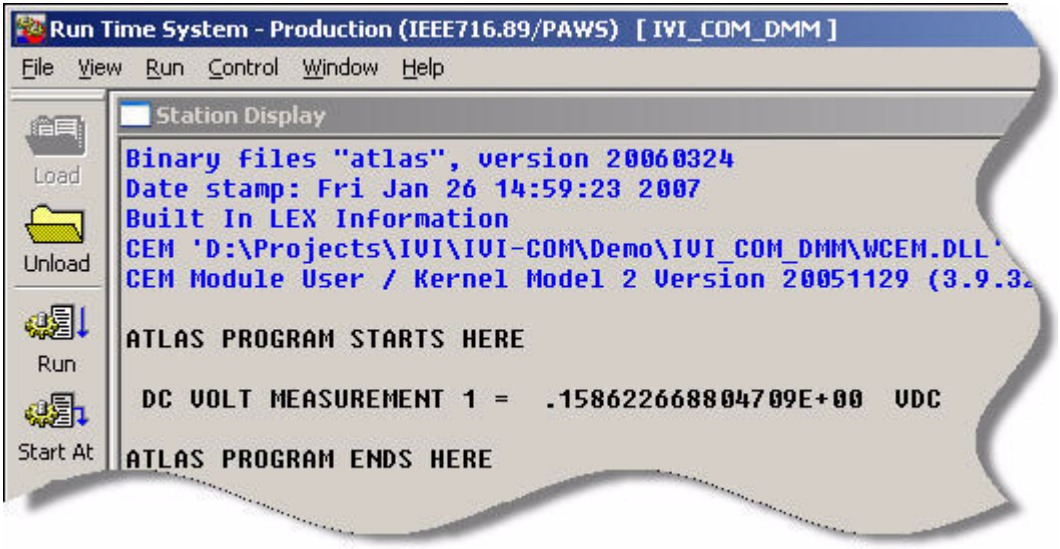
Load and Run the Project

1 From the Main Menu, select Build and click on Execute Wrts.

***Note**: The message below may appear before invoking the Run-Time system interface. This message usually appears when you have changed the code without recompiling it. Click Yes to let the project be rebuilt before execution. If you make changes to the Busconfi file only, you do not need to rebuild the project, because it is used as a reference file at run-time.*



2 From the list at left, select Run. The program runs and returns a DC Voltage measurement.



3 From the Main Menu, select File and click Exit to exit from the Run-Time system.

Further Information

Learn more about PAWS at http://www.tyx.com/pawsdeva.html.