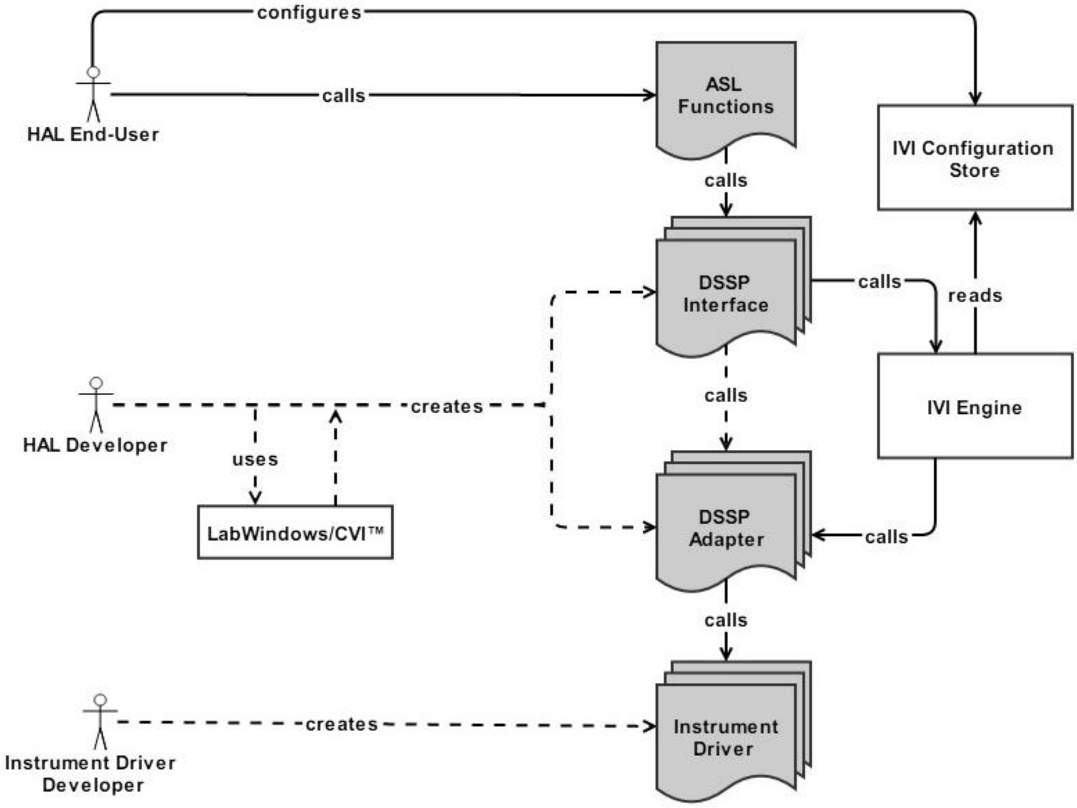
Implementing a C-based Hardware Abstraction Layer (HAL)

1. Introduction

This document covers the process to create a C-based Hardware Abstraction Layer (HAL) step by step. To get the most out of this document, first review Appendix A in the *Mitigate Hardware Obsolescence* white paper. The key value of a HAL is to implement an interchangeable layer for high-level applications so that user can change test system hardware without changing test system software. The following image shows the C-based HAL system diagram.



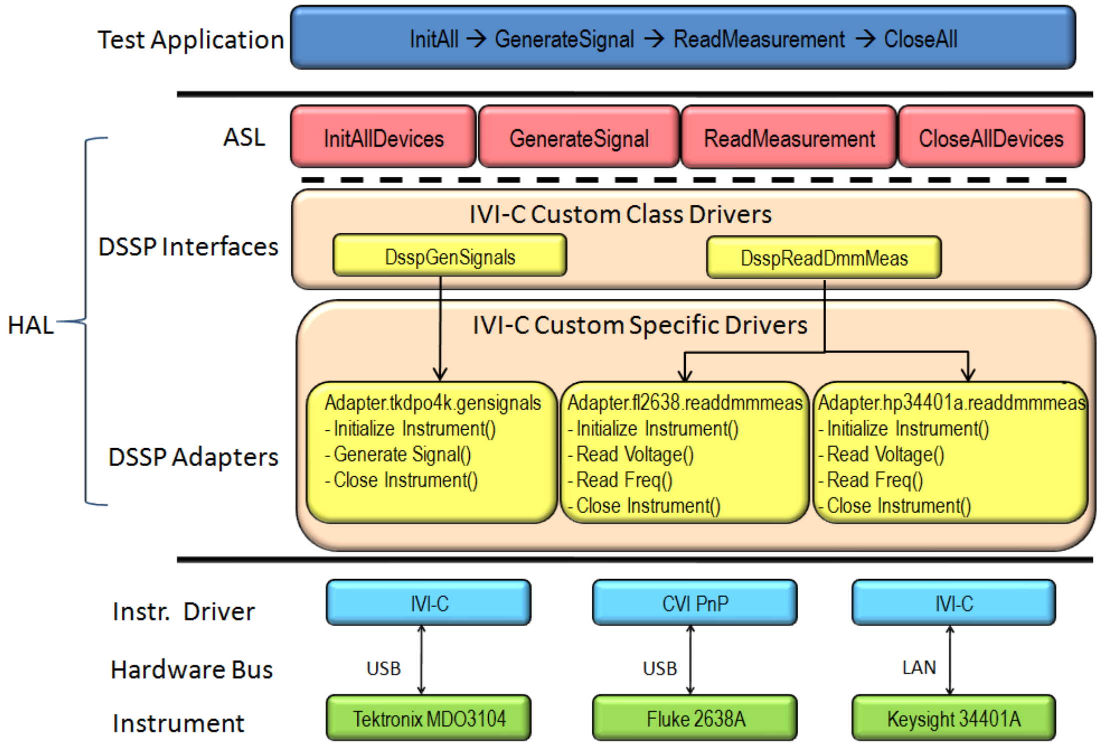
This example uses a function generator to generate signals and a digital multimeter to receive and measure the signals. This example illustrates how to write a C-based HAL from start to finish. The following table shows the instruments used in this example.

Instrument	Type	Bus Interface	Instrument Driver
Tektronix MDO3104	Function Generator	USB	IVI-C
Keysight 34401A	Digital Multimeter	LAN	IVI-C
Fluke 2638A	Digital Multimeter	USB	CVI PnP

This example will use the Tektronix MDO3104 oscilloscope as a function generator and two digital multimeters to demonstrate how the HAL supports hardware interchangeability. Refer to the document *How to Run the Example* for details on how to run the example.

2. Implementation

The following diagram shows the C-based HAL architecture, with specific functions and instrument models.



The example uses the IVI-C architecture provided by the NI IVI Compliance Package (ICP) to implement the HAL because ICP provides well-tested and highly used components that are required for a HAL: edit-time interface definition and binding, run-time binding to specific interface implementations based on configuration external to the client program (i.e., client programs can use different implementations without code changes), and tools for specifying system configuration. Additionally, LabWindows™/CVI™ provides IVI tools to help in the creation of HAL components.

The example creates a custom IVI-C Class Driver for each DSSP Interface and a custom IVI-C Specific Driver for each DSSP Adapter. Note that even though the example uses the ICP infrastructure for the HAL layer, IVI-C drivers are not required for the Instrument Driver layer. In the cases where you want to use an IVI-C driver in the Instrument Driver layer, the architecture uses IVI in both the HAL layer and the Instrument Driver layer.

The following table shows the Prefix and Published API Names in this example. (Refer to Section 2.1.3 for details.)

Type	Prefix	Published API
IVI-C Custom Class	ReadDmmMeas	N/A
IVI-C Custom Class	GenSignals	N/A
IVI-C Custom Specific	hp34401areaddmmmeas	ReadDmmMeas
IVI-C Custom Specific	fl2638readdmmmeas	ReadDmmMeas
IVI-C Custom Specific	tkdpo4kgensignals	GenSignals

2.1. DSSP Layer

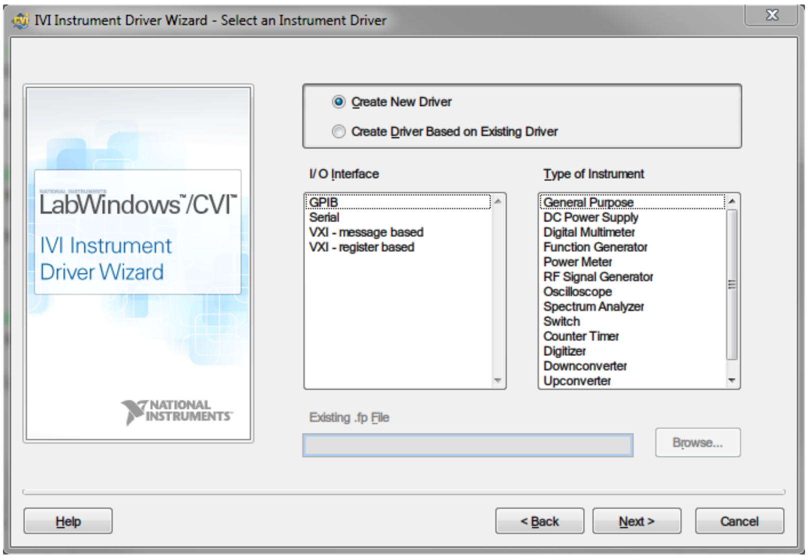
The DSSP layer includes two types of components: DSSP Interfaces, implemented as IVI-C Custom Class Drivers, and DSSP Adapters, implemented as IVI-C Custom Specific Drivers. The following sections describe how to build a DSSP Adapter and then derive a DSSP Interface from the DSSP Adapter. Another viable approach is to first define the DSSP Interface in a .fp file, generate the DSSP Interface shell from the .fp file, and then generate the DSSP Adapter from the DSSP Interface.

2.1.1. DSSP Adapters

Create the DSSP Adapter modules using the **IVI Instrument Driver Wizard** in LabWindows/CVI. Then you can modify the generated files to meet system requirements. The following sections show how to create a DSSP Adapter for the Keysight hp34401a IVI-C instrument driver.

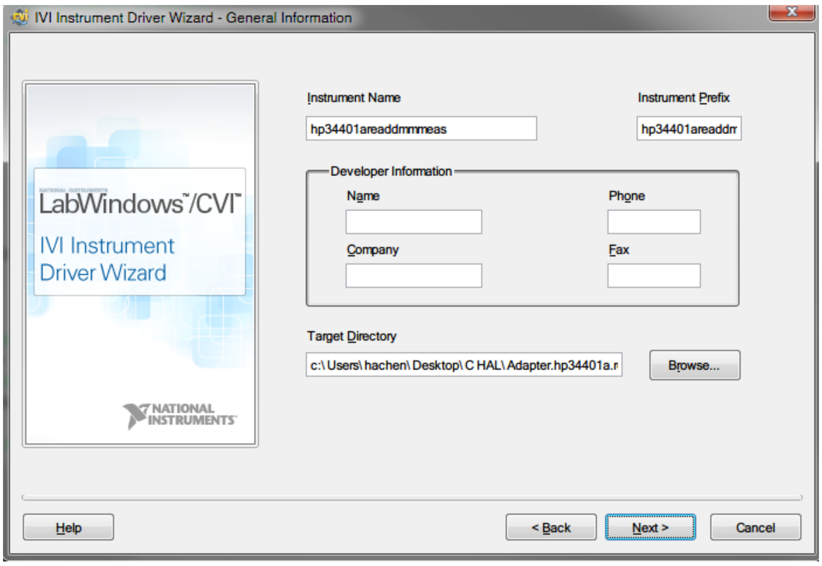
2.1.1.1. Use the IVI Specific Driver Wizard to Generate the DSSP Adapter Shell

1. In LabWindows/CVI, choose (**Tools»IVI Development»Create IVI Specific Driver**) to run the wizard and click **Next**.
2. Select **Create New Driver** and then click **Next**.

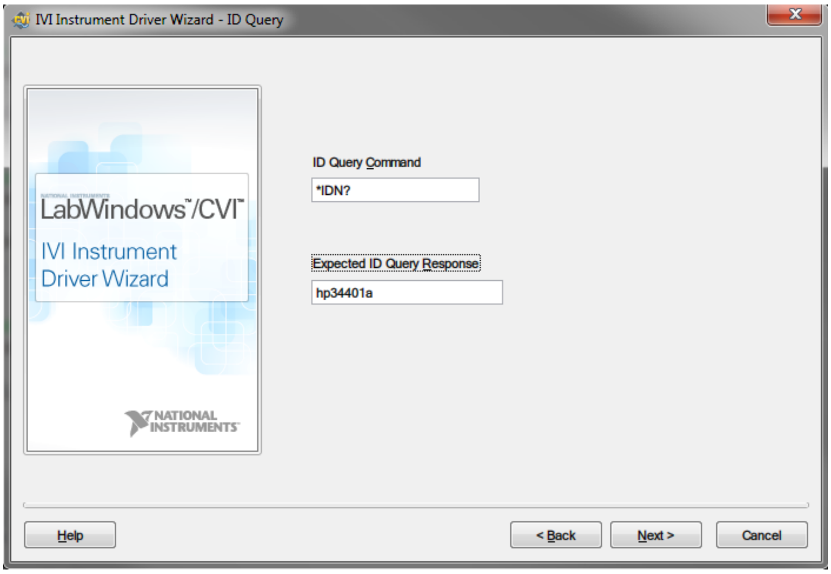


3. In this page, you need to provide the information of following editors:
   - **Instrument Name** is the unique identification of a DSSP Adapter. Name the module `hp34401areaddmmmeas`.
   - **Instrument Prefix** should be the same name as **Instrument Name**.
   - **Target Directory** specifies the location where the source code will be generated.

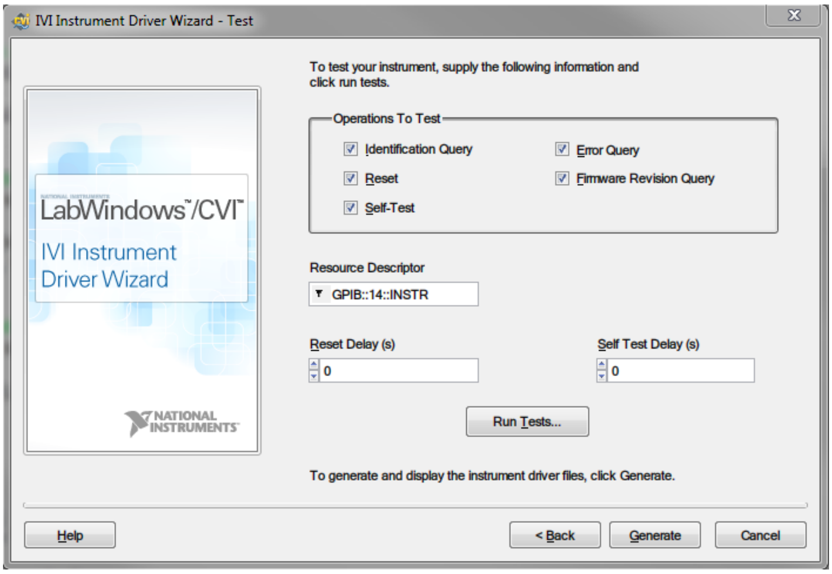
   Then, click **Next** all the way to **ID Query** page.



4. Enter the **Expected ID Query Response**. You can enter any string for this text box. The only reason to enter this is to enable the **Next** button to continue the wizard. Click **Next** all the way to **Test** page.

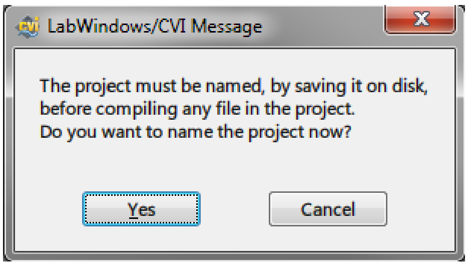


5. Click **Generate** to finish the wizard.



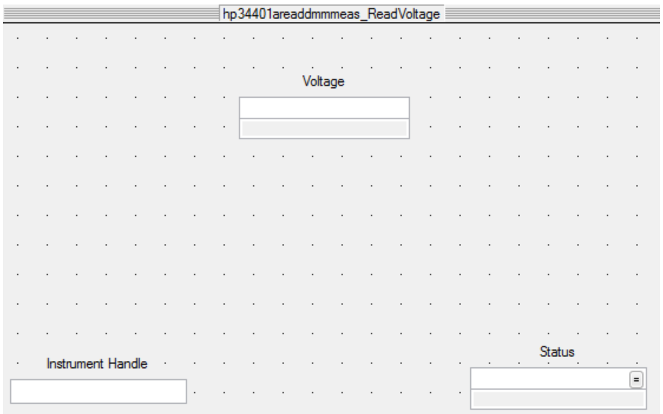
2.1.1.2. Modify the DSSP Adapter Source and Build the Project

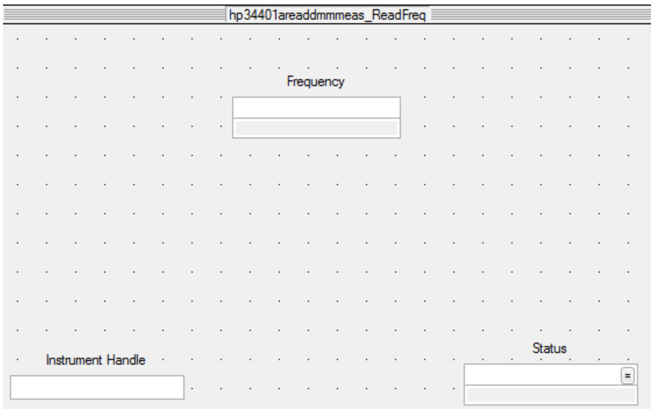
1. Create a new project (**File»New»Project**) in LabWindows/CVI.
2. Add the generated `.c`, `.h` and `.fp` files into the project. If you see the following popup dialog, click **Yes** and save the project.



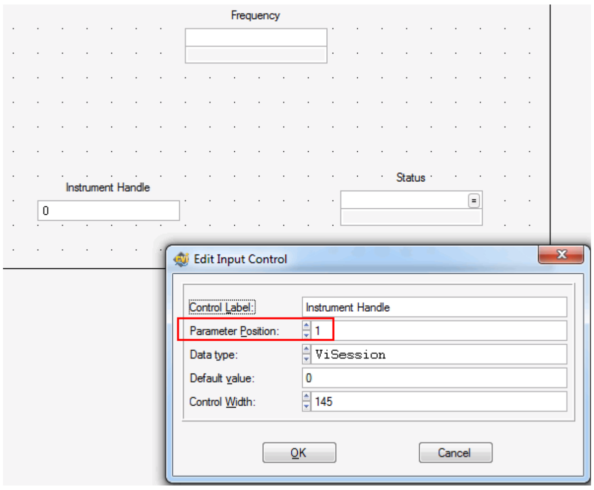
3. Add additional function panels in the `.fp` file for each function in the DSSP Interface you are implementing. For the hp34401a digital multimeter driver example, add two function panel windows:
   - ReadVoltage to measure voltage
     a) Voltage: ViReal64 Output Control
     b) Instrument Handle: ViSession Input Control
     c) Status: ViStatus Return Value Control

- ReadFreq to measure frequency
    a) Frequency: ViReal64 Output Control
    b) Instrument Handle: ViSession Input Control
    c) Status: ViStatus Return Value Control

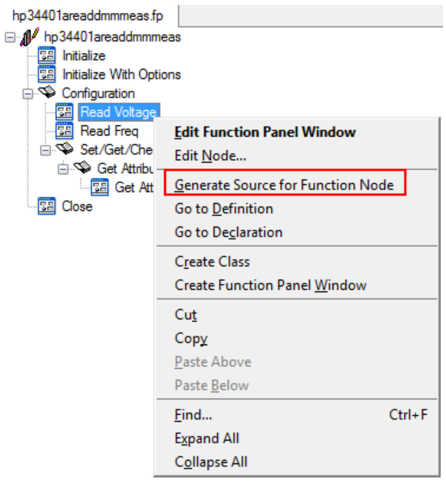




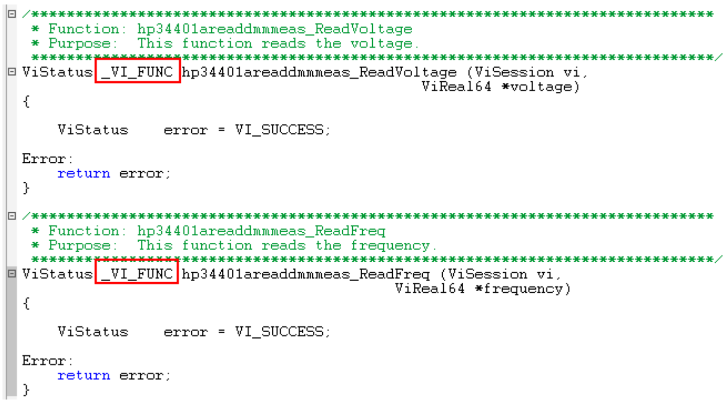
Note: Ensure that Instrument Handle is always the first parameter.



4. Right-click on the newly added function panel window and select **Generate Source for Function Node** to generate source code.

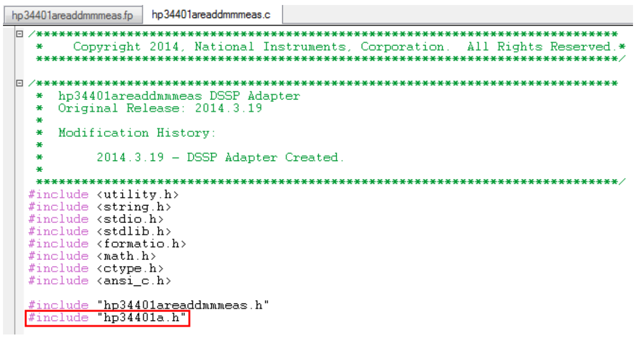


5. Add `_VI_FUNC` to the newly generated functions (for both `.c` and `.h` files).



6. To use the IVI-C specific driver hp34401a include the header file by adding the following line in the top of the `.c` file:

   `#include "hp34401a.h"`



7. Implement the following functions.
   - hp34401areaddmmmeas_init
   - hp34401areaddmmmeas_InitWithOptions
   - hp34401areaddmmmeas_close
   - hp34401areaddmmmeas_ReadVoltage
   - hp34401areaddmmmeas_ReadFreq

   These functions are included in the example source file. You can find them by right-clicking the `*.fp` function and selecting **Go to Definition**. For types of drivers other than IVI-C, you need to write some extra code to hold the IVI session (DSSP Adapter session) in global memory. Refer to the sample project Adapter.fl2638.readdmmmeas for a LabWindows/CVI PnP example.

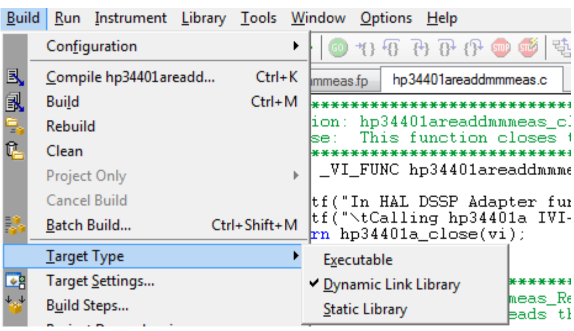
```c
/*****************************************************************************
 * Function: hp34401areaddmmmeas_init
 * Purpose:  VXIplug&play required function.  Calls the
 *           hp34401areaddmmmeas_InitWithOptions function.
 *****************************************************************************/
ViStatus _VI_FUNC hp34401areaddmmmeas_init (ViRsrc resourceName, ViBoolean IDQuery,
                                    ViBoolean resetDevice, ViSession *newVi)
{
    return hp34401areaddmmmeas_InitWithOptions (resourceName, IDQuery, resetDevice, "", newVi);
}

/*****************************************************************************
 * Function: hp34401areaddmmmeas_InitWithOptions
 * Purpose:  This function creates a new IVI session and calls the
 *           IviInit function.
 *****************************************************************************/
ViStatus _VI_FUNC hp34401areaddmmmeas_InitWithOptions (ViRsrc resourceName, ViBoolean IDQuery,
                                               ViBoolean resetDevice, ViConstString optionString,
                                               ViSession *newVi)
{
    printf("In HAL DSSP Adapter funtion: hp34401areaddmmmeas_InitWithOptions...\n");
    printf("\tCalling hp34401a IVI-C specific function: hp34401a_InitWithOptions...\n");
    return hp34401a_InitWithOptions (resourceName, IDQuery, resetDevice, optionString, newVi);
}

/*****************************************************************************
 * Function: hp34401areaddmmmeas_close
 * Purpose:  This function closes the instrument.
 *****************************************************************************/
ViStatus _VI_FUNC hp34401areaddmmmeas_close (ViSession vi)
{
    printf("In HAL DSSP Adapter funtion: hp34401areaddmmmeas_close...\n");
    printf("\tCalling hp34401a IVI-C specific function: hp34401a_close...\n");
    return hp34401a_close(vi);
}
```

Functions hp34401areaddmmmeas_ReadVoltage and hp34401areaddmmmeas_ReadFreq invoke functions exported from hp34401a specific driver. Each of them configures the measurement and trigger first, and then reads the measurement result.

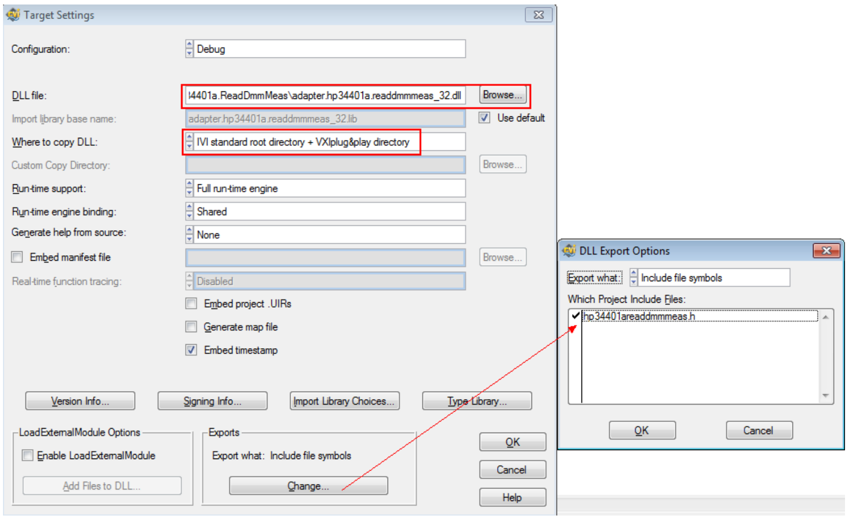
```c
/*****************************************************************************
 * Function: hp34401areaddmmmeas_ReadVoltage
 * Purpose:  This function reads the voltage.
 *****************************************************************************/
ViStatus _VI_FUNC hp34401areaddmmmeas_ReadVoltage (ViSession vi,
                                           ViReal64 *voltage)
{
    printf("In HAL DSSP Adapter funtion: hp34401areaddmmmeas_ReadVoltage...\n");

    ViStatus    error = VI_SUCCESS;

    printf("\tCalling hp34401a IVI-C specific function: hp34401a_ConfigureMeasurement...\n");
    checkErr( hp34401a_ConfigureMeasurement (vi, HP34401A_VAL_AC_VOLTS, HP34401A_VAL_AUTO_RANGE_ON, 0.0001));

    printf("\tCalling hp34401a IVI-C specific function: hp34401a_ConfigureTrigger...\n");
    checkErr( hp34401a_ConfigureTrigger (vi, HP34401A_VAL_IMMEDIATE, 0.0));

    Delay(1);

    printf("\tCalling hp34401a IVI-C specific function: hp34401a_Read...\n");
    checkErr( hp34401a_Read (vi, 5000, voltage));

Error:
    return error;
}

/*****************************************************************************
 * Function: hp34401areaddmmmeas_ReadFreq
 * Purpose:  This function reads the frequency.
 *****************************************************************************/
ViStatus _VI_FUNC hp34401areaddmmmeas_ReadFreq (ViSession vi,
                                        ViReal64 *frequency)
{
    printf("In HAL DSSP Adapter funtion: hp34401areaddmmmeas_ReadFreq...\n");

    ViStatus    error = VI_SUCCESS;

    printf("\tCalling hp34401a IVI-C specific function: hp34401a_ConfigureMeasurement...\n");
    checkErr( hp34401a_ConfigureMeasurement (vi, IVIDMM_VAL_FREQ, HP34401A_VAL_AUTO_RANGE_ON, 0.0001));

    printf("\tCalling hp34401a IVI-C specific function: hp34401a_ConfigureTrigger...\n");
    checkErr( hp34401a_ConfigureTrigger (vi, HP34401A_VAL_IMMEDIATE, 0.0));

    Delay(1);

    printf("\tCalling hp34401a IVI-C specific function: hp34401a_Read...\n");
    checkErr( hp34401a_Read (vi, 5000, frequency));
Error:
    return error;
}
```

8. Remove all unnecessary IVI infrastructure code.
    - Remove all get/set attribute functions except GetAttributeViBoolean in `.c` and `.h`.
    - Remove all unnecessary `#define` statements in the `.c` and `.h` files.
    - Remove all unnecessary callbacks in `.c`.
    - Remove all unnecessary exportable functions in `.c`, `.h` and `.fp`.
    - Replace code in the prefix_InitAttributes function with `return VI_SUCCESS;`.
    - Update comments if necessary.

    Normally, you only need to keep prefix_InitAttributes, prefix_init, prefix_InitWithOptions, prefix_close, prefix_GetAttributeViBoolean and the newly added functions.
9. Change the build target type to **Dynamic Link Library**.



10. Change Target Settings (**Build»Target Settings**). The DLL file will be stored in the `[path to current project]\adapter.hp34401a.readdmmmeas_32.dll` and will also be copied to the `<IVI Foundation>\IVI\Bin` directory.



11. `adapter.hp34401a.readdmmmeas_32.dll` file will be generated when you build the project. LabWindows/CVI copies the DLL to the `<IVI Foundation>\IVI\Bin` folder, so that it could be loaded there.

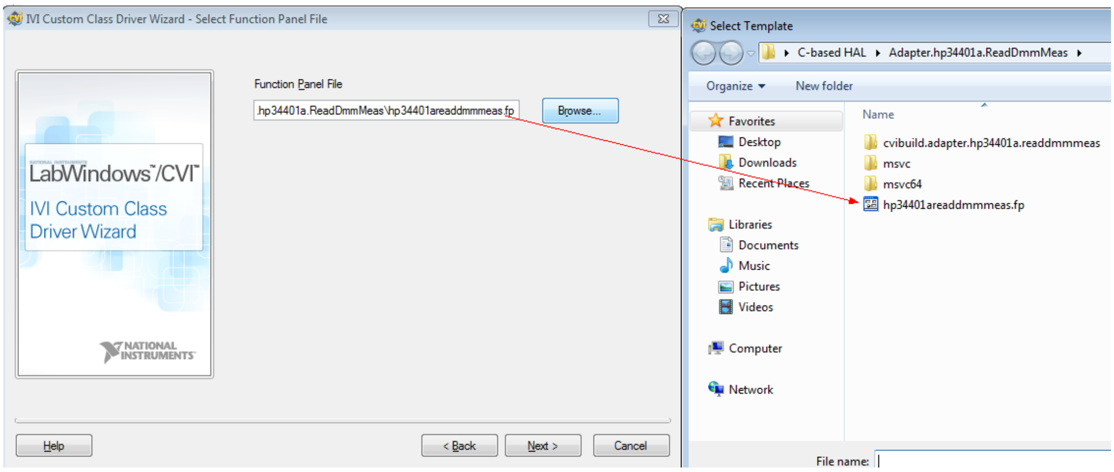
Note: Switch to **Build»Configuration»Release64/Debug64** for a 64-bit binary.

2.1.2. DSSP Interfaces

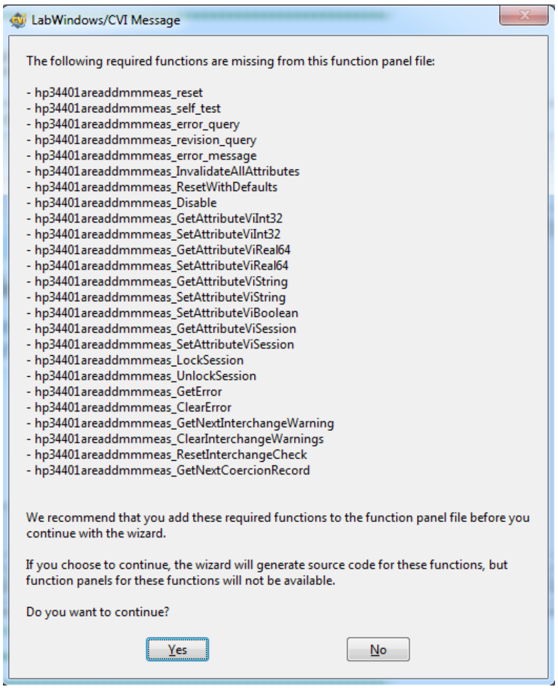
DSSP Interface modules can be created via IVI Custom Class Driver Wizard in LabWindows/CVI. Then customize the generated files to meet your needs. The following sections show how to create a Digital Multimeter (ReadDmmMeas) DSSP Interface.

2.1.2.1. Use the IVI Class Driver Wizard to Generate the DSSP Interface Shell

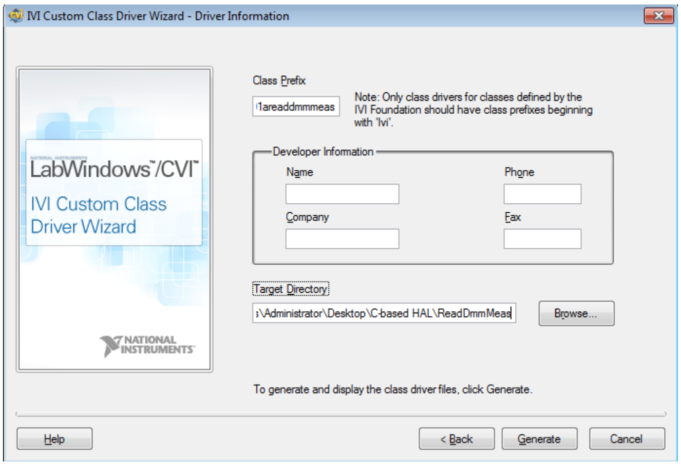
1. In LabWindows/CVI, choose (**Tools»IVI Development»Create IVI Custom Class Driver**) to run the wizard and click **Next**.
2. Select the `.fp` file path of a DSSP Adapter module you created in section 2.1.1



If you see the following warning, click **Yes** to continue.

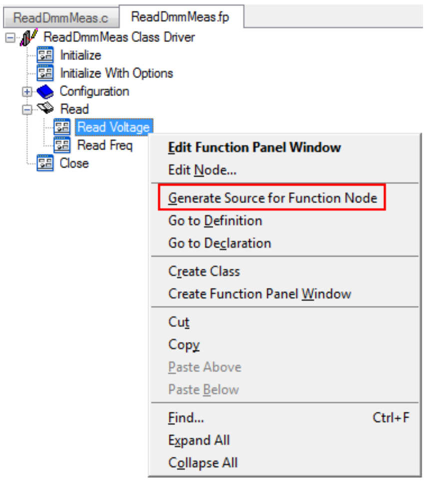


3. **Class Prefix** is a unique name to identify a DSSP Interface module. **Target Directory** is where the wizard should generate the source code. After entering the **Class Prefix** and **Target Directory**, click **Generate** to finish the wizard.



2.1.2.2. Modify the DSSP Interface Source and Build the Project

1. Create a new project (**File»New»Project**) in LabWindows/CVI.
2. Add the generated `ReadDmmMeas.c`, `ReadDmmMeas.h`, and `ReadDmmMeas.fp` files into the project.
3. The wizard could generate functions defined in the `.fp` selected in Step 2 of Section 2.1.2.1. You could add additional function panel windows to the `.fp` file if needed. In ReadDmmMeas DSSP Interface module, you do not need to add any functions panel windows.
4. If there are any new function panel windows, perform the following steps:
   a) Right-click on the newly added function panel window, and select **Generate Source for Function Node** to generate C source code.



b) In `ReadDmmMeas.c`, define function pointers for the new functions created in step 3.

```c
typedef ViStatus (_VI_FUNC *ReadVoltageFuncPtr) (ViSession vi, ViReal64 *voltage);
typedef ViStatus (_VI_FUNC *ReadFreqFuncPtr) (ViSession vi, ViReal64 *frequency);
```

c) Add function names and tables.

```c
/****************************************************************************
 *  ReadDmmMeas Function Names, Tables, and Typedefs
 ****************************************************************************/
enum
    {
    kReadVoltage = kNumInherentFunctions,
    kReadFreq,
    kNumFunctions
    };

static char *functionNames[kNumFunctions + 1] =
    {
    INHERENT_FUNCTION_NAMES,
    "ReadVoltage",
    "ReadFreq",
    VI_NULL
    };

static IviStringValueTable funcTable =
    {
    INHERENT_FUNCTION_TABLE(ReadDmmMeas),
    {kReadVoltage,      "ReadDmmMeas_ReadVoltage"},
    {kReadFreq,      "ReadDmmMeas_ReadFreq"},
    {VI_NULL, VI_NULL}
    };
```

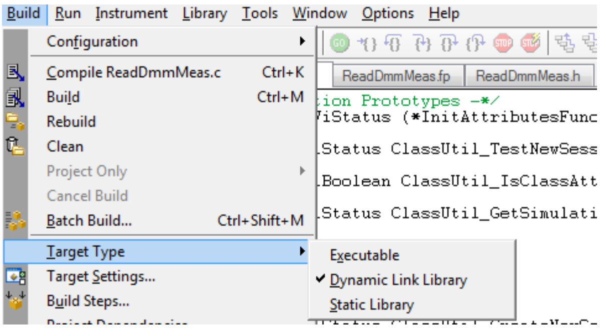
d) Implement the newly added functions.

```c
/****************************************************************************
 *  Function: ReadDmmMeas_ReadVoltage
 *  Purpose: Read the voltage.
 ****************************************************************************/
ViStatus _VI_FUNC ReadDmmMeas_ReadVoltage (ViSession vi, ViReal64 *voltage)
{
    printf("In HAL DSSP Interface function: ReadDmmMeas_ReadVoltage...\n");
    DECLARE(ReadVoltage)

    LOCK()

    SPECIFIC_CALL_BEGIN(ReadVoltage)
        SPECIFIC_CALL_ARG(voltage)
    SPECIFIC_CALL_END()

    READDMMMEAS_SIMULATE_BEGIN(ReadVoltage)
        SIMULATE_ARG(voltage)

    SIMULATE_END()

Error:
    UNLOCK()
    RETURN()
}

/****************************************************************************
 *  Function: ReadDmmMeas_ReadFreq
 *  Purpose: Read the frequency.
 ****************************************************************************/
ViStatus _VI_FUNC ReadDmmMeas_ReadFreq (ViSession vi, ViReal64 *frequency)
{
    printf("In HAL DSSP Interface function: ReadDmmMeas_ReadFreq...\n");
    DECLARE(ReadFreq)

    LOCK()

    SPECIFIC_CALL_BEGIN(ReadFreq)
        SPECIFIC_CALL_ARG(frequency)
    SPECIFIC_CALL_END()

    READDMMMEAS_SIMULATE_BEGIN(ReadFreq)
        SIMULATE_ARG(frequency)

    SIMULATE_END()

Error:
    UNLOCK()
    RETURN()
}
```

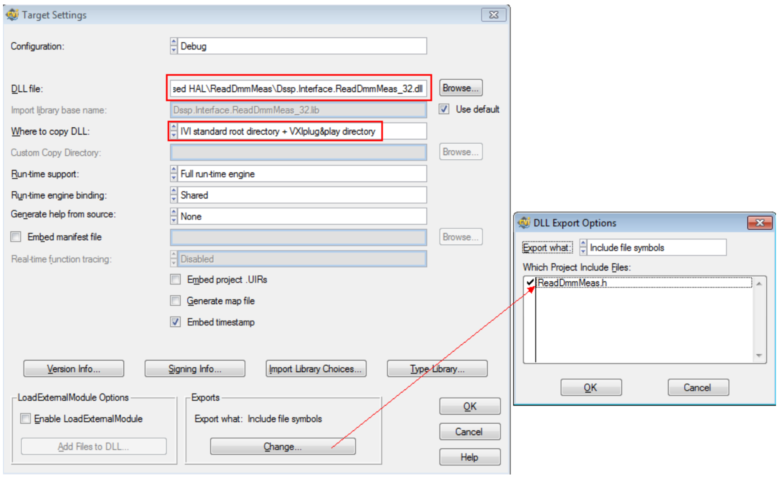
5. To avoid the conflict of global class utility functions among different DSSP Interface modules, add key word `static` to all class utility function prototypes and DllMain function.

```c
/*- Function Prototypes -*/
typedef ViStatus (*InitAttributesFuncPtr)(ViSession vi);

static ViStatus ClassUtil_TestNewSession(ViSession* newVi, ViStatus secondError);

static ViBoolean ClassUtil_IsClassAttribute (ViInt32 attributeId);

static ViStatus ClassUtil_GetSimulationDriverInfo   (ViSession vi,
                                                     ViConstString defaultSoftwareModule,
                                                     ViConstString functionName,
                                                     ViSession* simVi,
                                                     ViAddr* functionPtr,
                                                     ViString functionNames[]);

static ViStatus ClassUtil_CreateNewSession (ViConstString logicalName,
                                            ViConstString optionString,
                                            ViBoolean idQuery,
                                            ViBoolean resetDevice,
                                            ViSession* newSession,
                                            ViConstString className,
                                            ViString functionNames[],
                                            InitAttributesFuncPtr InitAttributes);

static ViStatus ClassUtil_DisposeSession (ViSession vi);
static ViStatus ClassUtil_GetAttributeViBooleanFromSpecificDriver (ViSession vi,
                                                                   ViConstString channelName,
                                                                   ViAttr attributeId,
                                                                   ViBoolean *value);


#define CLASSUTIL_DISPOSE_ON_ERROR()                                          \
    if (error < VI_SUCCESS)                                                   \
    {                                                                         \
        ClassUtil_DisposeSession (vi);                                        \
        Ivi_Dispose (vi);                                                     \
        *newSession = VI_NULL;                                                \
    }                                                                         \

/*****************************************************************************
 * Build Dll Main function
 *****************************************************************************/
static int __stdcall DllMain (HINSTANCE hinstDLL, DWORD fdwReason, LPVOID lpvReserved)
{
    switch (fdwReason)
        {
        case DLL_PROCESS_ATTACH:
            if (InitCVIRTE (hinstDLL, 0, 0) == 0)
                return 0;     /* out of memory */
            break;
        case DLL_PROCESS_DETACH:
            CloseCVIRTE ();
            break;
        }

    return 1;
}
```

6. Remove all unnecessary IVI infrastructure code.
    1. Remove all get/set attribute functions except GetAttributeViBoolean in `.c` and `.h`.
    2. Remove all unnecessary `#define` statements in the `.c` and `.h` files.
    3. Remove all unnecessary callbacks in `.c`.
    4. Remove all unnecessary exportable functions in `.c`, `.h` and `.fp`.
    5. Remove all unnecessary class utility functions.
    6. Replace code in the prefix_InitAttributes function with `return VI_SUCCESS;`.
    7. Update comments if necessary.
7. Change the build target type to **Dynamic Link Library**.



8. Change the target settings (**Build»Target Settings**).



9. `ReadDmmMeas_32.dll` file will be generated after building the project. CVI copy the DLL to the `<IVI`

Foundation>\IVI\Bin folder, so that it could be loaded there.

Note: You could switch to **Build»Configuration»Release64/Debug64** for a 64-bit binary.

2.1.3. System Configuration

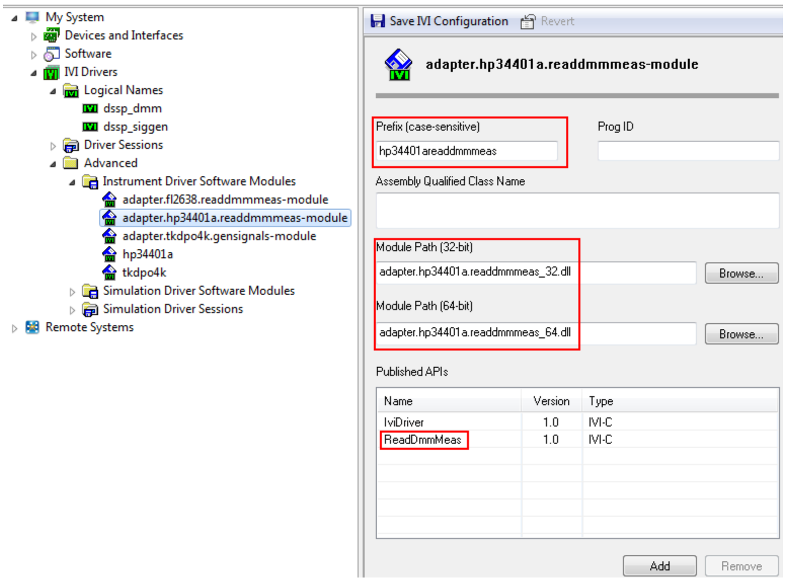
You can use the IVI System configuration tools in NI Measurement & Automation Explorer (MAX) to configure the C-based HAL system. There is a **Driver Session** and a **Software Module** for each DSSP Adapter. There is a **Logical Name** for each DSSP Interface. Please refer to the document *Using Measurement & Automation Explorer to Configure Your IVI System* for more details of system configuration.

This example uses 2 DSSP Interfaces (IVI-C Custom Class Drivers) and 3 DSSP Adapters (IVI-C Custom Specific Drivers). So in MAX, you should set up two Logical Names, three Driver Sessions, and three Software Modules for the system. The following steps describe how to configure the Digital Multimeter function DSSP Interface and hp34401a DSSP Adapter.

2.1.3.1. Software Module

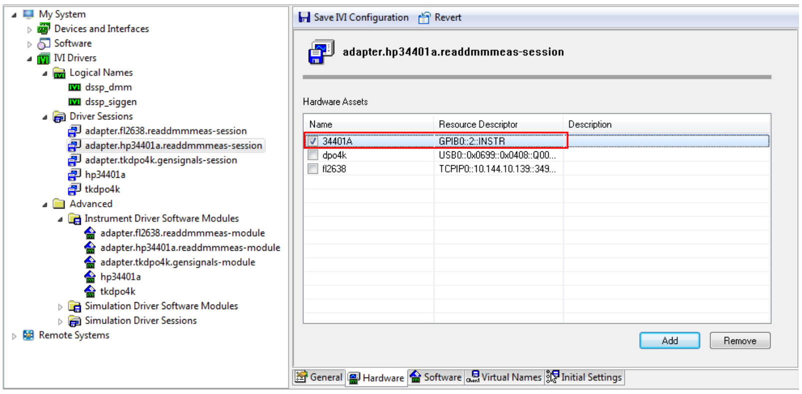
1. Expand **My System»IVI Drivers»Advanced**, right click the **Instrument Driver Software Modules** node, and select **Create New (case-sensitive)** to create a software module. Rename the module to adapter.hp34401a.readdmmmeas-module.
2. On the General page, enter the DLL name (**Module Path (32-bit) and (Module Path (64-bit)**), prefix excluding qualifier (**Prefix (case-sensitive)**), and qualifier (**Published APIs**) of the module, as follows:

Software Module	Prefix	Module Path (32-bit) (change "_32" to "_64" for 64-bit)	Published APIs
adapter.hp34401a.readdmmmeas-module	hp34401areaddmmmeas	adapter.hp34401a.readdmmmeas_32.dll	ReadDmmMeas

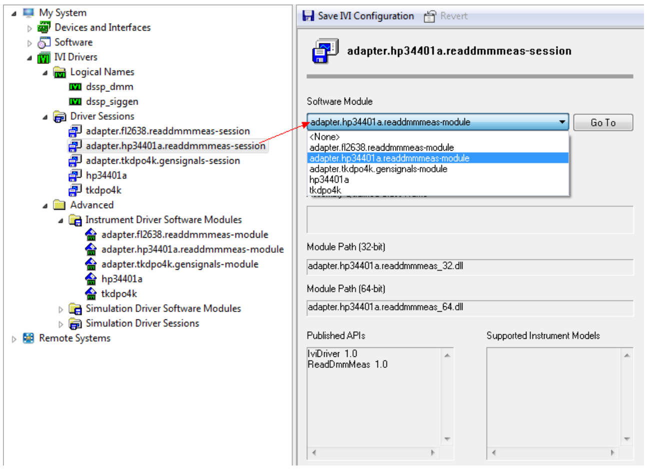


2.1.3.2. Driver Session

1. Right-click **Driver Sessions** node and select **Create New (case-sensitive)** to create a driver session named adapter.hp34401a.readdmmmeas-session.
2. On the **Hardware** tab, add your instruments.

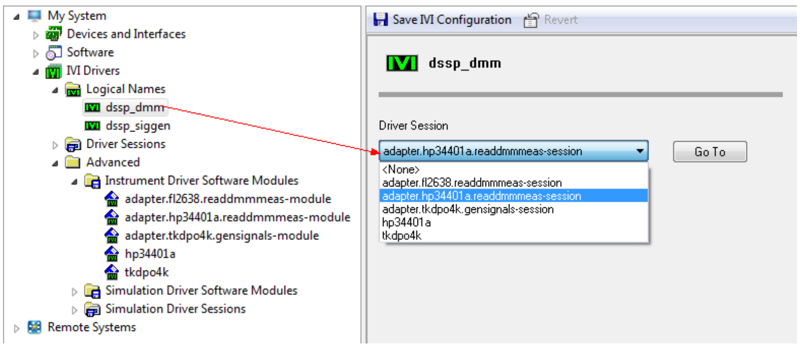


3. On the **Software** tab, select the related software module.



2.1.3.3. Logical Name

1. Right click **Logical Names** node and select **Create New (case-sensitive)** to create a Logical Name called dssp_dmm.
2. In **General** page, select the related driver session.



2.2. ASL Layer

The ASL layer provides application-specific interfaces to serve the upper layer application. This example defines four ASL layer interfaces:

- InitAllDevices(): initialize the signal generator and digital multimeter.
- GenerateSignal(): make the signal generator to generate signals.
- ReadMeasurement(): make the digital multimeter to measure the generated signals.
- CloseAllDevices(): close the signal generator and digital multimeter.

The ASL layer functions call the DSSP Interface (IVI-C Custom Class Driver) only.

```
// initialize the signal generator and digital multimeter.
ViStatus InitAllDevices(ViSession *siggen, ViRsrc siggenResourceName, ViSession *dmm, ViRsrc dmmResourceName)
{
    printf("Initialize devices...\n");

    ViStatus    error = VI_SUCCESS;
    checkErr( GenSignals_InitWithOptions(siggenResourceName, VI_FALSE, VI_TRUE, "simulate=1,DriverSetup=Model : MDO3054", siggen));
    checkErr( ReadDmmMeas_InitWithOptions(dmmResourceName, VI_TRUE, VI_TRUE, "simulate=1", dmm));

    printf("All devices have been initialized.\n\n");

Error:
    return error;
}

// make the signal generator to generate signals.
ViStatus GenerateSignal(ViSession vi, ViReal64 frequency, ViReal64 amplitude)
{
    printf("Generate signal...\n");

    ViStatus    error = VI_SUCCESS;
    checkErr( GenSignals_GenerateSignal(vi, frequency, amplitude));

    printf("Signal has been generated...\n\n");

Error:
    return error;
}

// make the digital multimeter to measure the generated signals.
ViStatus ReadMeasurement(ViSession vi, ViReal64 *voltage, ViReal64 *frequency)
{
    printf("Read signal...\n");

    ViStatus    error = VI_SUCCESS;
    checkErr( ReadDmmMeas_ReadVoltage(vi, voltage));
    checkErr( ReadDmmMeas_ReadFreq(vi, frequency));

    printf("Finish reading signal...\n\n");

Error:
    return error;
}

// close the signal generator and digital multimeter.
ViStatus CloseAllDevices(ViSession siggen, ViSession dmm)
{
    printf("Close devices...\n");

    ViStatus    error = VI_SUCCESS;
    checkErr( GenSignals_close(siggen));
    checkErr( ReadDmmMeas_close(dmm));

    printf("All devices have been closed.\n\n");

Error:
    return error;
}
```

3. Test Application

Enter the following code to implement a test application that uses a function generator to generate signals and a DMM to measure some characteristic of signal.

```
main ()
{
    ViStatus  error = VI_SUCCESS;
    ViSession siggen = VI_NULL;
    ViSession dmm = VI_NULL;

    // dssp_siggen is the logical name configured in MAX
    ViRsrc siggenResourceName = "dssp_siggen";
    // dssp_dmm is the logical name configured in MAX
    ViRsrc dmmResourceName = "dssp_dmm";

    // frequency and amplitude of the generated signal
    ViReal64 frequency = 100.0;
    ViReal64 amplitude = 2;

    // measurement results
    ViReal64 readVoltage = 0.0;
    ViReal64 readFrequency = 0.0;

    // call ASL layer functions
    checkErr( InitAllDevices(&siggen, siggenResourceName, &dmm, dmmResourceName));
    checkErr( GenerateSignal(siggen, frequency, amplitude));
    checkErr( ReadMeasurement(dmm, &readVoltage, &readFrequency));

    // pop up the measurement results
    ViChar result[512];
    sprintf(result, "AC Voltage: %f\nFrequency: %f\n", readVoltage, readFrequency);
    MessagePopup("message", result);

Error:
    if (error != VI_SUCCESS)
    {
        ViChar   errStr[2048];
        sprintf(errStr, "Error Code: %x.\nPlease refer to NI LabWindows/CVI Help for detals", error);
        MessagePopup ("Error!", errStr);
    }

    // close sessions
    if (siggen||dmm)
        CloseAllDevices(siggen, dmm);

    // do not close the console unless user press Enter
    printf("Press Enter to exit...");
    getchar();
}
```

Note that the test application calls and only calls ASL layer functions.

In MAX, dssp_siggen points to adapter.tkdpo4k.gensignals-session and dssp_dmm points to adapter.hp34401a.readdmmmeas-session. So the test application will control Tektronix MDO3104 to generate signals and control Keysight 34401A to do the measurements.

To use a Fluke 2638A instead of a Keysight 34401A to do the measurement, open MAX and make dssp_dmm point to adapter.fl2638.readdmmmeas-session. You do not need to change any of your test code!