Using DriverLINX® with HP VEE 5.0

by
Andrea Clary
Keithley Instruments, Inc.

Introduction

Today, many software development tools benefit from the continued development of ActiveX® controls. ActiveX controls are part of the component object model (COM) developed by Microsoft®. It is a method of providing reusable code, according to a known standard, for rapid application development. ActiveX can be thought of as an extension of the Object Linking and Embedding (OLE) concept. Just as a graph from a spreadsheet can be embedded into a word processing document, ActiveX controls can be used from any development environment that supports the technology. Within the data acquisition paradigm, these controls provide a consistent and easy interface to the hardware, independent of the programming language or development environment. HP VEE supports the use of ActiveX controls; therefore, it can use the DriverLINX driver to perform data acquisition tasks.

DriverLINX as an ActiveX Control

DriverLINX has two ActiveX controls: a Service Request (SR) and a Logical Device Descriptor (LDD). The use of the LDD is not mandatory when developing an application. The LDD is a control that provides information about the features of the installed hardware to the application. (For example, the LDD can determine if an installed board has analog input capabilities and if so, how many channels.) An application could be written such that a channel selector that ranges from channel 0 to the maximum channel of the board is available for the user of the application. By using the LDD, the program would be able to dynamically determine the appropriate number for the maximum channel (7 in the case of a DAS-800 Series board and 63 in the case of the DAS-1800HC Series board).

While the use of the LDD is optional, the use of the SR control is not. The SR is the interface for putting the features supported by the hardware into action. The LDD may inform the application of how many channels are available, but the SR selects which channel(s) are to be used.

ActiveX controls, according to the COM model standards, provide properties, methods, and events for the controlling application to use.

- *Properties* are the characteristics of the hardware that the control exposes (for example, the analog input of a multifunction board). Examples of properties would be starting channel, stopping channel, gain, clock rate, digital trigger, or gate.
- *Methods* are functions, or actions, that the control carries out. Programming the various properties of the control can configure an acquisition. However, nothing will happen until these properties of the SR control are communicated to the hardware. (The Refresh method is the command that carries out this communication.) In a generic sense, methods are how the control can send information or give commands.
- *Events* are how the application that hosts the control can receive information from the hardware or process. For example, if an application sets the properties of the SR control such that the board should acquire 500 samples from a channel according to an external clock and then stop, the SR control would post a message to the application when the 500 samples have been acquired. This type of information exchange can be extremely beneficial in situations where one is not exactly sure when the acquisition might be complete. Imagine an external clock signal that is not periodic; the 500 clock pulses that result in the 500 samples might occur in one minute or in 20 minutes. The use of events in the Windows® operating system frees up the CPU from continually testing for completion of the acquisition. The CPU is free to do other processes and the application will be informed by the ActiveX control when attention is required.

Another benefit of event-driven processing is robust and portable code across computers of vastly different CPU speeds. Imagine a simple application that needs to acquire only one sample from each of eight channels when the operator clicks a button with the mouse. A program that operates successfully on a 100MHz Pentium may not operate successfully when ported to a much faster machine, unless there is tight coordination between when the data is ready and when the CPU executes procedures to retrieve the data. Simple for-next loops or other polling approaches from traditional procedural programming days are vulnerable to incorrect operation when ported to faster computers.

HP VEE as an ActiveX Container

Hewlett-Packard's (now, Agilent Technologies') VEE is a powerful drag-and-drop graphical development environment. It supports the use of ActiveX controls in close accordance with Microsoft's COM model. Therefore, it can be called an ActiveX container.

Using DriverLINX with HP VEE

Every installation of Windows has many ActiveX controls (usually in OCX files) installed on the system. The first step with any development environment is to add the control to the project. From the HP VEE Device menu, select ActiveX Control References, then select the DriverLINX ActiveX Controls. This makes the two DriverLINX controls available to the HP VEE environment; however, one more step is required to place the controls into service. Again from the Device Menu, select ActiveX Controls, and a drop-down list will appear with all the controls that are available to the HP VEE environment. (The DriverLINXLDD and the DriverLINXSR controls will appear in this list.) From this list, select the DriverLINXSR control. This will cause the SR control to be added to the specific application being developed. *Figure 1* is a screen shot of the SR control when added to an application:

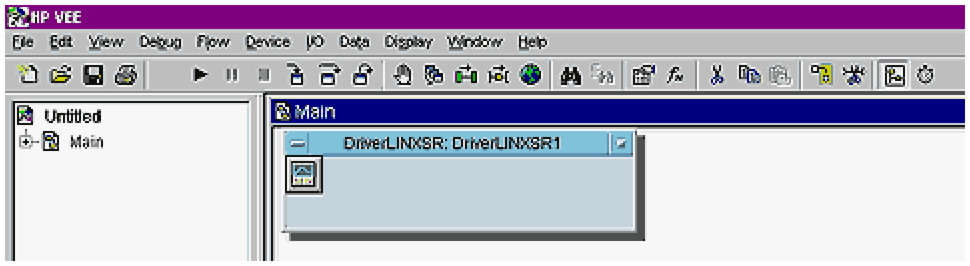
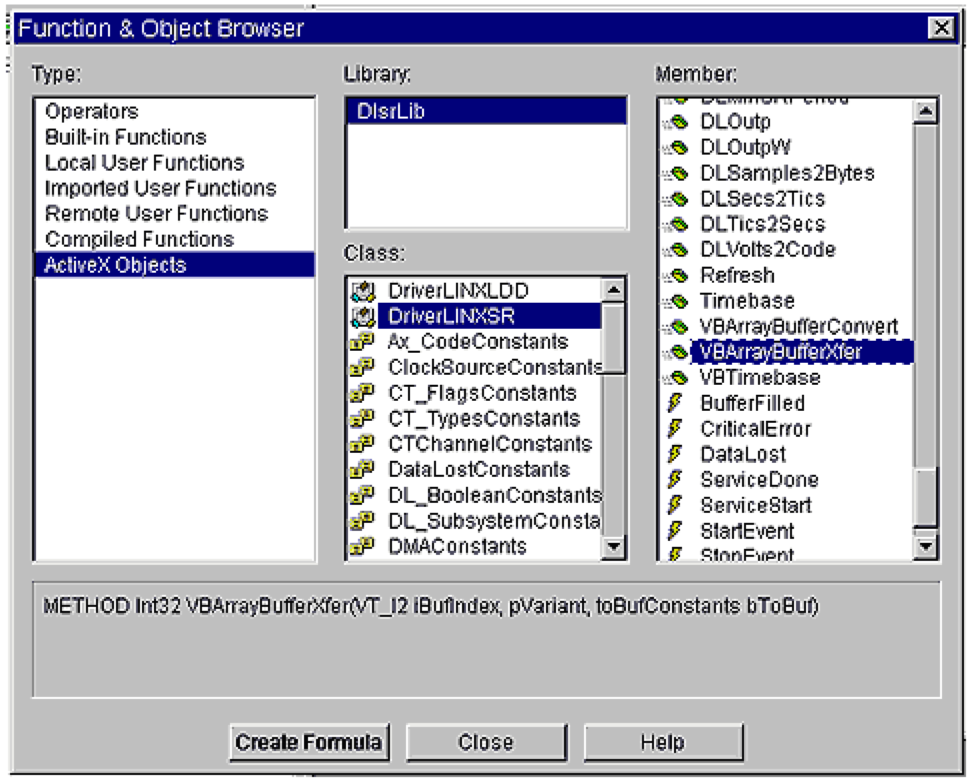

HP VEE
File Edit View Debug Flow Device I/O Data Display Window Help
Untitled
Main
Main
DriverLINXSR: DriverLINXSR1

Figure 1. Adding an SR Control to an Application

HP VEE also provides an object browser that is useful in determining which components of the control will be used in the application. All the various properties, methods, and events of the control can be viewed with the object browser. In *Figure 2*, the VBArrayBufferTransfer method of the SR control is highlighted. Also, the events of the control (Critical Error, Service Done, etc.) are visible.



METHOD Int32 VBArrayBufferXfer(VT_I2 iBufIndex, pVariant, toBufConstants bToBuf)

Figure 2. HP VEE Object Browser

Figure 3 is a simple interrupt mode application that uses the Keithley KPCI-3108 multifunction board to do A/D (analog to digital) conversions on channel 0 and display the result when the Service Done Event Message is posted to the application from the DriverLINX SR control. This example program is comprised of five blocks:

- SR control
- Initialize function
- Interrupt mode AI function
- Service_Done event handler
- Display block for the output

Each function block can have input and output terminals for data. The blocks also have sequence terminals to control the order in which the blocks are executed. In this example, the Initialize block executes first and then control is passed to the Interrupt mode AI block. Neither of these blocks currently uses data input or output.

Right clicking on the SR control and selecting the Create Event Handler option creates the Service_Done event handler. From the various events of the SR control, the Service_Done event was selected for this example. This block passes its data result on its output terminal to the Chan0_Volts data input terminal. Notice that the sequence block of the event handler is not wired, so that it is not executed at any specific time. This is consistent with the event-driven paradigm of Windows and ActiveX technology. When the interrupt mode acquisition is finished, the data will certainly be ready. Inside this Service_Done event handler, code is placed that converts the first reading to volts, then passes the result to the Chan0 Volts display.

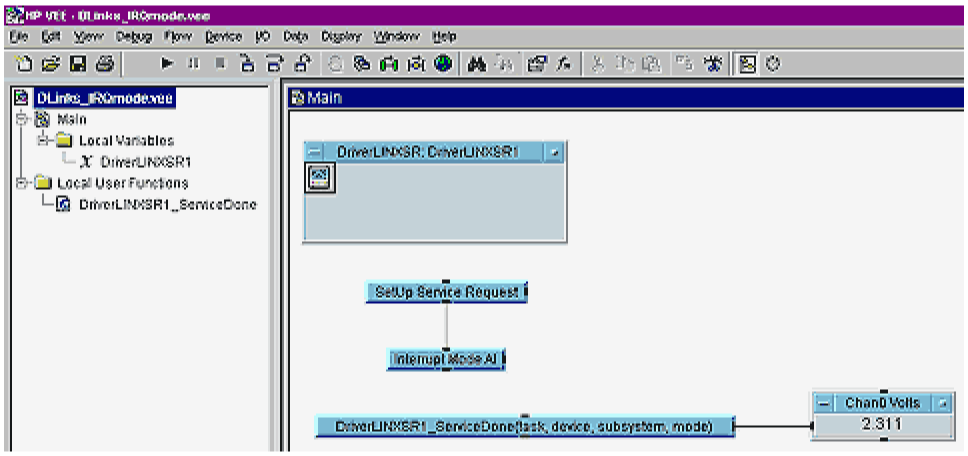
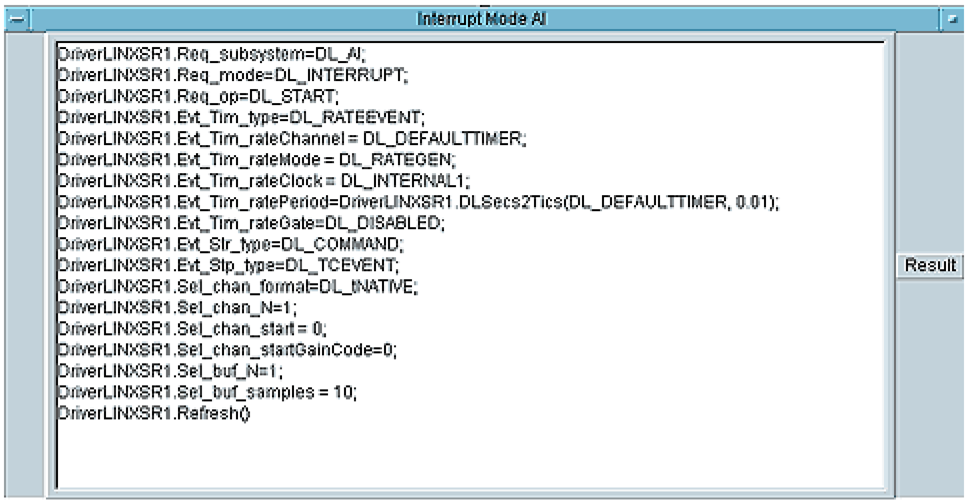

HP VEE - DLinks_IRQmode.vee
File Edit View Debug Flow Device I/O Data Display Window Help
DLinks_IRQmode.vee
Main
Local Variables
DriverLINXSR1
Local User Functions
DriverLINXSR1_ServiceDone
Main
DriverLINXSR: DriverLINXSR1
SetUp Service Request
Interrupt Mode AI
DriverLINXSR1_ServiceDone(task, device, subsystem, mode)
Chan0 Volts
2.311

Figure 3. Example of a Simple Interrupt Mode Application

Figure 4 is an expanded view of the Interrupt mode AI function block. The code inserted here is very much like code that would be used in any development environment that supports the use of objects. The specific instance of the SR control in this application is called DriverLINXSR1. Several properties have been set using standard notation for objects. The constants on the right side of the equations are part of the SR control. These constants and their meanings are outlined in the DriverLINX documentation. The last statement is a call to the Refresh method of the object that executes the interrupt mode acquisition.



```
DriverLINXSR1.Req_subsystem=DL_AI;
DriverLINXSR1.Req_mode=DL_INTERRUPT;
DriverLINXSR1.Req_op=DL_START;
DriverLINXSR1.Evt_Tim_type=DL_RATEEVENT;
DriverLINXSR1.Evt_Tim_rateChannel = DL_DEFAULTTIMER;
DriverLINXSR1.Evt_Tim_rateMode = DL_RATEGEN;
DriverLINXSR1.Evt_Tim_rateClock = DL_INTERNAL1;
DriverLINXSR1.Evt_Tim_ratePeriod=DriverLINXSR1.DLSecs2Tics(DL_DEFAULTTIMER, 0.01);
DriverLINXSR1.Evt_Tim_rateGate=DL_DISABLED;
DriverLINXSR1.Evt_Str_type=DL_COMMAND;
DriverLINXSR1.Evt_Stp_type=DL_TCEVENT;
DriverLINXSR1.Sel_chan_format=DL_tNATIVE;
DriverLINXSR1.Sel_chan_N=1;
DriverLINXSR1.Sel_chan_start = 0;
DriverLINXSR1.Sel_chan_startGainCode=0;
DriverLINXSR1.Sel_buf_N=1;
DriverLINXSR1.Sel_buf_samples = 10;
DriverLINXSR1.Refresh()
```

Figure 4. Interrupt Mode AI Function Block (Expanded View)

Download the example program (DLinks_IRQmode.zip).

For more HP VEE examples refer to Keithley's on-line technical support.

Conclusion

ActiveX control technology is making the job of software development easier. The DriverLINX controls (SR and LDD) provide a hardware-independent model for programming a data acquisition task. These controls allow the same code to be used with many different boards. Since ActiveX controls are not compiler or development environment specific, but rather conform to leading edge technology, the combination of Keithley hardware and DriverLINX driver permits the end user to develop in any Windows language of choice, without restriction.