A Comparison of the ActiveX Programming Interface with the DLL Interface using DriverLINX

By Dave Sherman,
Keithley Instruments, Inc.

Introduction

DriverLINX is a set of 32-bit drivers for data acquisition hardware and can be used in development environments such as Microsoft Visual Basic or Visual C++. The underlying concept of DriverLINX programming is that the user specifies a service request and submits it to the DriverLINX Application Programming Interface (API). This service request has various properties that specify the hardware to use, the type of operation to be performed, and whether it will be a foreground or background (message-based) operation. A foreground operation would involve making direct application requests, such as setting a digital output line or reading an analog input. A background operation would involve an application that, once started, runs by itself while the main application is doing something else. When the background application is complete, a Windows message (such as "the buffer has filled" or "the task has completed") is posted to the application.

DLL API programming

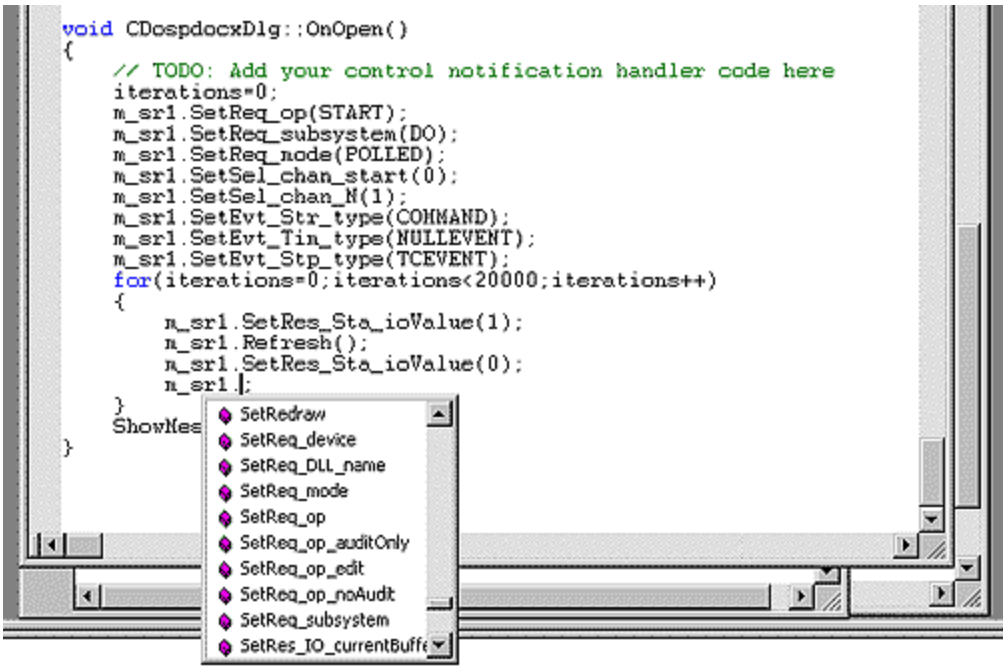
When programming in C++, the DLL interface is recommended. This involves creating a service request in the form of a data structure, the members of which reflect the parameters for the data acquisition task. The program first has to open an instance of the DriverLINX driver for the particular hardware being used and the driver needs to have a handle to the window requesting the task. A pointer to the service request structure then must be declared and memory is allocated using the C++ "new" operator. Once the necessary members of the structure are set, the pointer is passed as an argument to the DriverLINX function, which executes the request. If messaging is to be used, the programmer has to create the appropriate "WindowProc" function, and decode the message parameters to see if the message was posted by DriverLINX and then take the appropriate action.

ActiveX API programming

Users of Microsoft Visual Basic or Borland Delphi are instructed to use the ActiveX interface for developing applications. This is done by loading an ActiveX control into the application and setting the same parameters that would be set for the service request in C++. Many of the details involved in using C++, such as declaring the service request, allocating memory for the service request structure, initializing the members of the structure, and processing messages, are handled automatically in the ActiveX control. A Visual Basic user need only set the properties of the service request, calling one of the methods built into the control to execute the service request, then making use of the predefined message handling functions to process the results of the service request. This greatly simplifies the steps involved in making a service request,and reduces the possibility of errors, such as forgetting to de-allocate the memory used by the service request or neglecting to close the DriverLINX driver properly.

Using the ActiveX API in Visual C++

Microsoft Visual C++ has the ability to use ActiveX controls when building an application using the Microsoft Foundation Classes (MFC). When making an MFC application, the programmer enables the option of adding support for ActiveX controls. This makes all of the registered ActiveX controls in Windows available, including the DriverLINX ActiveX controls. If the programmer wants to use this control in his Visual C++ project, he adds the control as a component in his project. With this done, the programmer sets the members of the service request by calling a function that sets that parameter, such as SetReq_op(), which sets the requested operation. A nice feature of Visual C++ called Intellisense(r) provides a list of all the functions in the ActiveX control, which appears as the code is written, allowing the user to see the correct coding of the functions. The DLL interface, in contrast, requires the programmer to know exactly the name of the data structure member being set; otherwise, the compiler will produce errors because it doesn't recognize the name of the member.



Tradeoffs of using the ActiveX control

The ActiveX control does simplify making a service request; however, the control has code incorporated into it to perform the memory allocations and to translate the function calls associated with the control into the data structures passed to the DriverLINX DLLs. As a result, there is more processor overhead involved when running the application.

To assess the reduction in speed, two programs were created to perform the same task, one using the DLL interface and the other using the ActiveX controls. Both were made as MFC applications in Visual C++ 6.0, running on a 200MHz Pentium Pro under Windows NT Workstation 4.0. The task was to perform a foreground (polled) operation, toggling a digital output line on and off on a Keithley PIO-24 Digital I/O card. This task was performed 20,000 times, with an oscilloscope connected to the line to observe the length of a single transition.

The result of this testing indicated that, while there was not much tradeoff for a single task, in the long run, these tasks would add up. A single transition using the DLL interface took 0.5 milliseconds to perform, while a single transition took 0.6 milliseconds using the ActiveX interface. Correspondingly, the time to perform 20,000 iterations took 19 seconds using the DLL interface, but 23 seconds using the ActiveX interface. Note that this was performing a foreground polled operation; a background message-based application will have significantly less overhead.

OCX code listing:

```
            BOOL CDospdocxDlg::OnInitDialog()
{
        CDialog::OnInitDialog();

        // Add "About..." menu item to system menu.

        // IDM_ABOUTBOX must be in the system command range.
        ASSERT((IDM_ABOUTBOX & 0xFFF0) == IDM_ABOUTBOX);
        ASSERT(IDM_ABOUTBOX < 0xF000);

        CMenu* pSysMenu = GetSystemMenu(FALSE);
        if (pSysMenu != NULL)
        {
                CString strAboutMenu;
                strAboutMenu.LoadString(IDS_ABOUTBOX);
                if (!strAboutMenu.IsEmpty())
                {
                        pSysMenu->AppendMenu(MF_SEPARATOR);
                        pSysMenu->AppendMenu(MF_STRING, IDM_ABOUTBOX, strAboutMenu);
                }
        }

        // Set the icon for this dialog. The framework does this automatically
        //  when the application's main window is not a dialog
        SetIcon(m_hIcon, TRUE);                 // Set big icon
```

```
        SetIcon(m_hIcon, FALSE);               // Set small icon

        // TODO: Add extra initialization here
        m_sr1.SetReq_device(0);
        m_sr1.SetReq_DLL_name("KMBPIO");
        m_sr1.SetReq_op(INITIALIZE);
        m_sr1.SetReq_subsystem(DEVICE);
        m_sr1.SetReq_mode(OTHER);
        m_sr1.Refresh();
        ShowMessage(&m_sr1);
        m_sr1.SetReq_op(CONFIGURE);
        m_sr1.SetReq_mode(OTHER);
        m_sr1.SetReq_subsystem(DO);
        m_sr1.SetEvt_Stp_type(NULLEVENT);
        m_sr1.SetEvt_Tim_type(DIOSETUP);
        m_sr1.SetEvt_Tim_dioChannel(0);
        m_sr1.SetEvt_Tim_dioMode(DIO_BASIC);
        m_sr1.SetEvt_Stp_type(NULLEVENT);
        m_sr1.SetSel_chan_N(0);
        m_sr1.Refresh();
        ShowMessage(&m_sr1);

        return TRUE;  // return TRUE  unless you set the focus to a control
}
void CDospdocxDlg::OnExit()
{
        // TODO: Add your control notification handler code here
        m_sr1.SetReq_DLL_name(NULL);
        OnOK();

}
void CDospdocxDlg::ShowMessage(CDlsr *sr)
{
        if(sr->GetRes_result()!=NoErr)
        {
                sr->SetReq_op(MESSAGEBOX);
                sr->Refresh();
        }
}

void CDospdocxDlg::OnOpen()
{
        // TODO: Add your control notification handler code here
        iterations=0;
        m_sr1.SetReq_op(START);
        m_sr1.SetReq_subsystem(DO);
        m_sr1.SetReq_mode(POLLED);
        m_sr1.SetSel_chan_start(0);
        m_sr1.SetSel_chan_N(1);
        m_sr1.SetEvt_Str_type(COMMAND);
        m_sr1.SetEvt_Tim_type(NULLEVENT);
        m_sr1.SetEvt_Stp_type(TCEVENT);
        for(iterations=0;iterations<20000;iterations++)
        {
                m_sr1.SetRes_Sta_ioValue(1);
                m_sr1.Refresh();
                m_sr1.SetRes_Sta_ioValue(0);
                m_sr1.Refresh();
        }
        ShowMessage(&m_sr1);
}
```

DLL code listing:

```
            void CDospddllDlg::OnOpen()
{
        // TODO: Add your control notification handler code here
        hWnd=GetSafeHwnd();
        DriverInstance=OpenDriverLINX(hWnd,"kmbpio");
        pSR=(DL_ServiceRequest*)new DL_ServiceRequest;
        memset(pSR,0,sizeof(DL_ServiceRequest));
        DL_SetServiceRequestSize(*pSR);
        pSR->operation=INITIALIZE;
        pSR->subsystem=DEVICE;
        pSR->device=0;
        pSR->mode=OTHER;
        pSR->hWnd=hWnd;
        DriverLINX(pSR);
        showMessage(pSR);
        memset(pSR,0,sizeof(DL_ServiceRequest));
        DL_SetServiceRequestSize(*pSR);
        pSR->operation=CONFIGURE;
        pSR->mode=OTHER;
        pSR->subsystem=DO;
        pSR->device=0;
```

```
        pSR->timing.typeEvent=DIOSETUP;
        pSR->timing.u.diSetup.channel=0;
        pSR->timing.u.diSetup.mode=DIO_BASIC;
        pSR->hWnd=hWnd;
        DriverLINX(pSR);
        showMessage(pSR);

}
void CDospddllDlg::showMessage(DL_ServiceRequest *SR)
{
        if(SR->result!=NoErr)
        {
                SR->operation=MESSAGEBOX;
                DriverLINX(SR);
        }
}
void CDospddllDlg::OnButton1()
{
        // TODO: Add your control notification handler code here
        memset(pSR,0,sizeof(DL_ServiceRequest));
        DL_SetServiceRequestSize(*pSR);
        pSR->operation=START;
        pSR->device=0;
        pSR->mode=POLLED;
        pSR->subsystem=DO;
        pSR->channels.nChannels=1;
        pSR->channels.chanGain[0].channel=0;
        pSR->status.typeStatus=IOVALUE;
        pSR->hWnd=hWnd;
        for(iterations=0;iterations<20000;iterations++)
        {
                pSR->status.u.ioValue=0;
                DriverLINX(pSR);
                pSR->status.u.ioValue=1;
                DriverLINX(pSR);
        }
        showMessage(pSR);
}
```