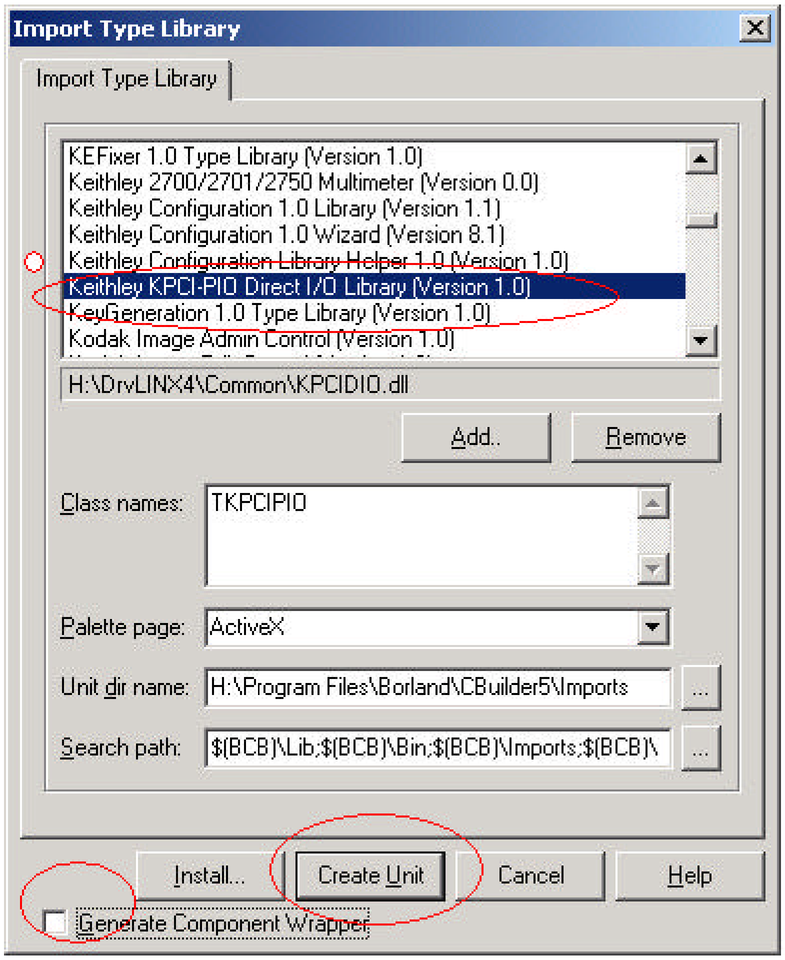
Using Direct I/O COM API of DriverLINX in Borland Builder

Components Used:

Windows 2000 SP1
DriverLINX version: KPCIPIO-850A04
Borland Builder 5.0 Standard
KPCI-PIO24

First step is to use the Import Type Library feature on the Project Menu within the Borland IDE. Below is a screen capture:



KEFixer 1.0 Type Library (Version 1.0)
Keithley 2700/2701/2750 Multimeter (Version 0.0)
Keithley Configuration 1.0 Library (Version 1.1)
Keithley Configuration 1.0 Wizard (Version 8.1)
Keithley Configuration Library Helper 1.0 (Version 1.0)
Keithley KPCI-PIO Direct I/O Library (Version 1.0)
KeyGeneration 1.0 Type Library (Version 1.0)
Kodak Image Admin Control (Version 1.0)

H:\DrvLINX4\Common\KPCIDIO.dll

Class names: TKPCIPIO

Palette page: ActiveX

Unit dir name: H:\Program Files\Borland\CBuilder5\Imports

Search path: $(BCB)\Lib;$(BCB)\Bin;$(BCB)\Imports;$(BCB)\

☐ Generate Component Wrapper

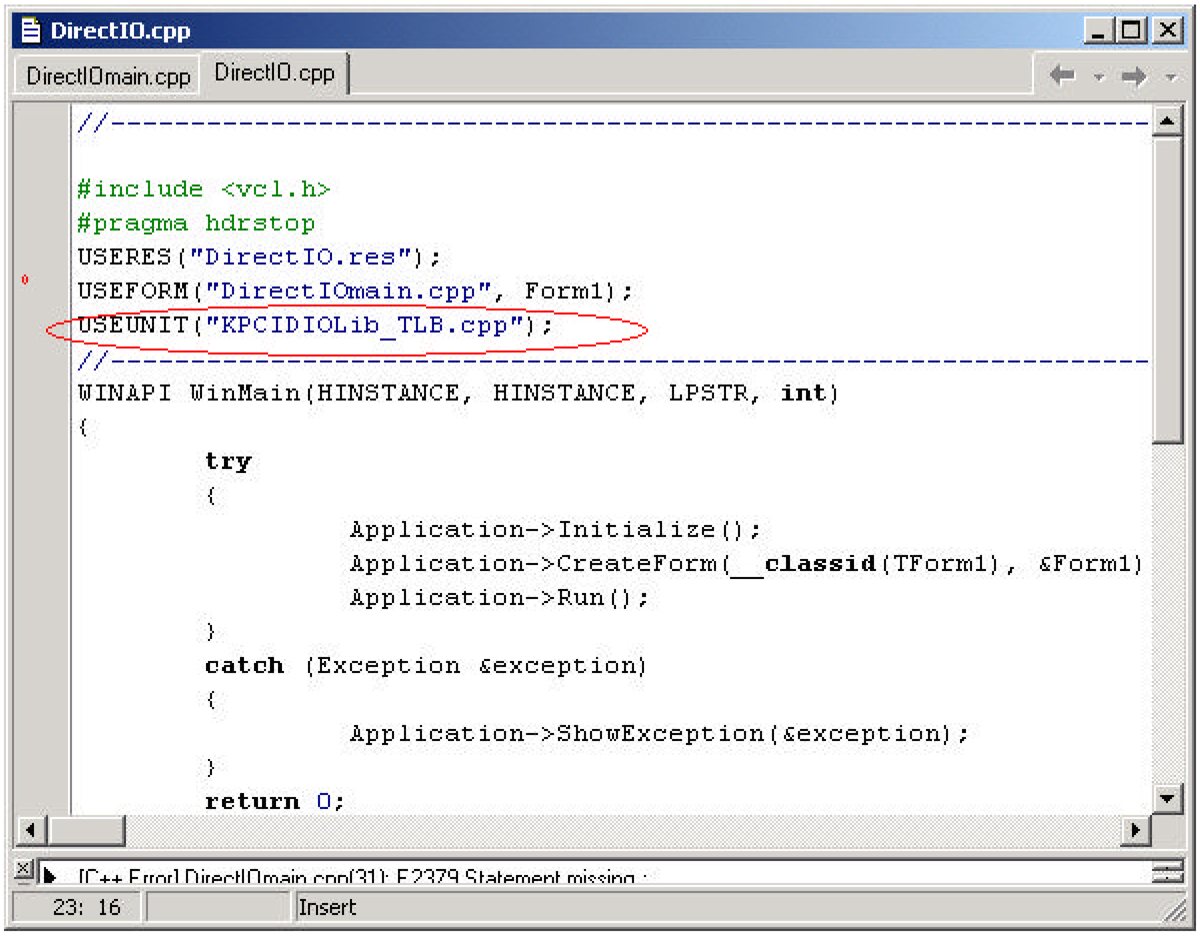
Scroll down to and select the Keithley Direct I/O Library
Remove check from the 'Generate Component Wrapper' check box.
Push the Create Unit button.

Several files will be put into the imports folder (\Program Files\Borland\CBuilder5\Imports\). The important ones for this topic are the KPCIDIOLib_TLB.h and KPCIDIOLib_TLB.cpp files.

Now that we have wrapper files, we can begin coding.

The screen shots below show some specific code for a project. In this case the two wrapper files *were relocated to the same folder as the project*.

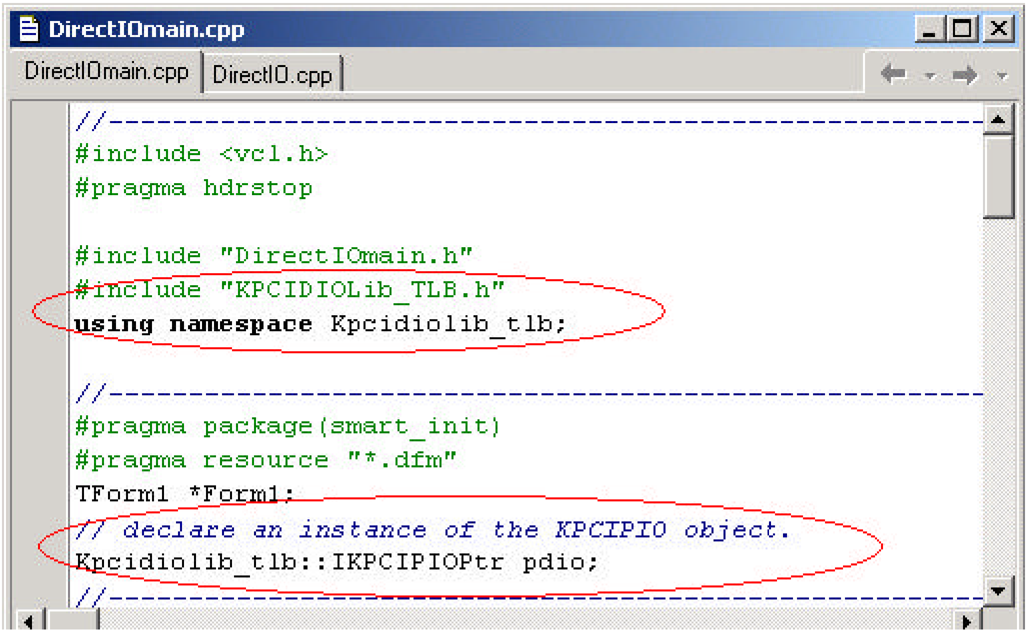
Notice a USEUNIT clause has been added to the application's cpp file:



```
//---------------------------------------------------------------

#include <vcl.h>
#pragma hdrstop
USERES("DirectIO.res");
USEFORM("DirectIOmain.cpp", Form1);
USEUNIT("KPCIDIOLib_TLB.cpp");
//---------------------------------------------------------------
WINAPI WinMain(HINSTANCE, HINSTANCE, LPSTR, int)
{
        try
        {
                Application->Initialize();
                Application->CreateForm(__classid(TForm1), &Form1)
                Application->Run();
        }
        catch (Exception &exception)
        {
                Application->ShowException(&exception);
        }
        return 0;
```

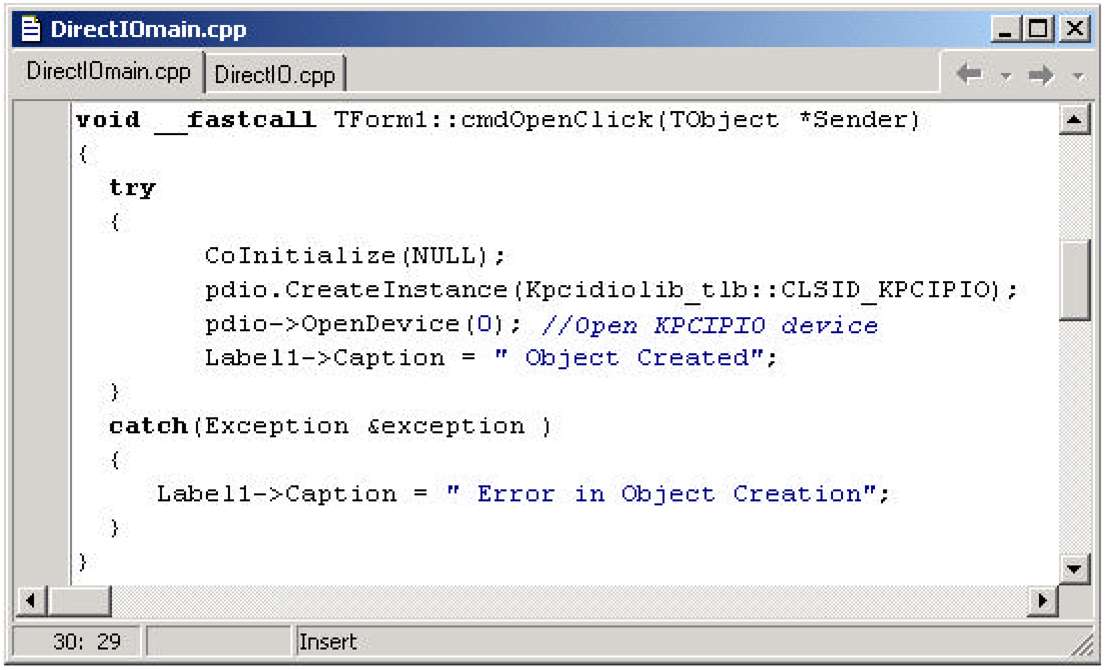
```
[C++ Error] DirectIOmain.cpp(31): E2379 Statement missing ;
```

Declarations in our source cpp file:



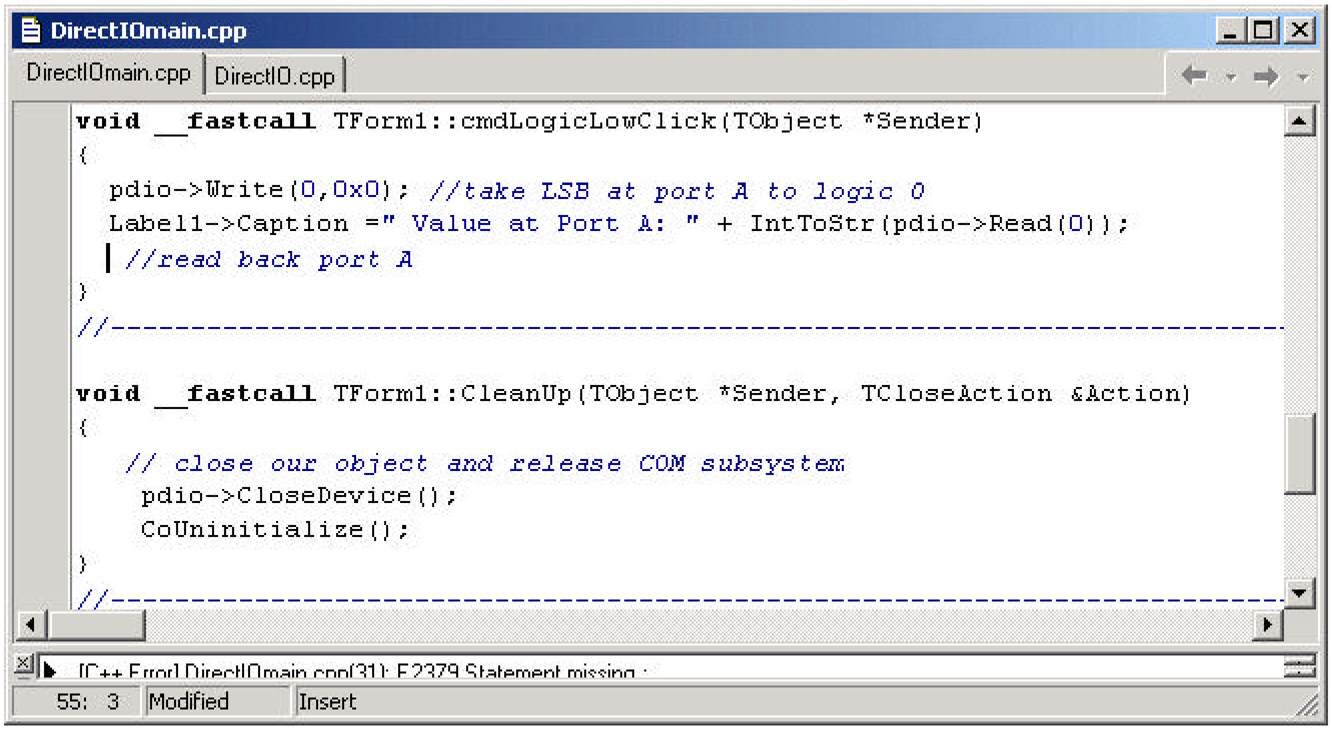
```
//---------------------------------------------------------------------------
#include <vcl.h>
#pragma hdrstop

#include "DirectIOmain.h"
#include "KPCIDIOLib_TLB.h"
using namespace Kpcidiolib_tlb;

//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;
// declare an instance of the KPCIPIO object.
Kpcidiolib_tlb::IKPCIPIOPtr pdio;
//---------------------------------------------------------------------------
```

Instantiate the Object and Call OpenDevice method:



```
void __fastcall TForm1::cmdOpenClick(TObject *Sender)
{
  try
  {
        CoInitialize(NULL);
        pdio.CreateInstance(Kpcidiolib_tlb::CLSID_KPCIPIO);
        pdio->OpenDevice(0); //Open KPCIPIO device
        Label1->Caption = " Object Created";
  }
  catch(Exception &exception )
  {
     Label1->Caption = " Error in Object Creation";
  }
}
```

Use the methods of the object and close on exit:
(Note: the code below does not show the port configuration step, e.g., pdio->Write(3,0x80); to make all ports outputs prior to any attempt to write values to the ports)



```
void __fastcall TForm1::cmdLogicLowClick(TObject *Sender)
{
  pdio->Write(0,0x0); //take LSB at port A to logic 0
  Label1->Caption =" Value at Port A: " + IntToStr(pdio->Read(0));
  //read back port A
}
//---------------------------------------------------------------------------

void __fastcall TForm1::CleanUp(TObject *Sender, TCloseAction &Action)
{
   // close our object and release COM subsystem
    pdio->CloseDevice();
    CoUninitialize();
}
//---------------------------------------------------------------------------
```

```
[C++ Error] DirectIOmain.cpp(31): E2379 Statement missing ;
```

Conclusion: The Direct I/O COM API of DriverLINX provides a simple and fast interface to the hardware. With only a few steps, it can be used from the Borland environment.