Errata

This document describes the use of the Telos 2101 AP component from the Microsoft Visual C++ environment. The example uses version 1 or the AP component. The general approach remains the same for version 2 but the name of the control and some function prototypes are changed.

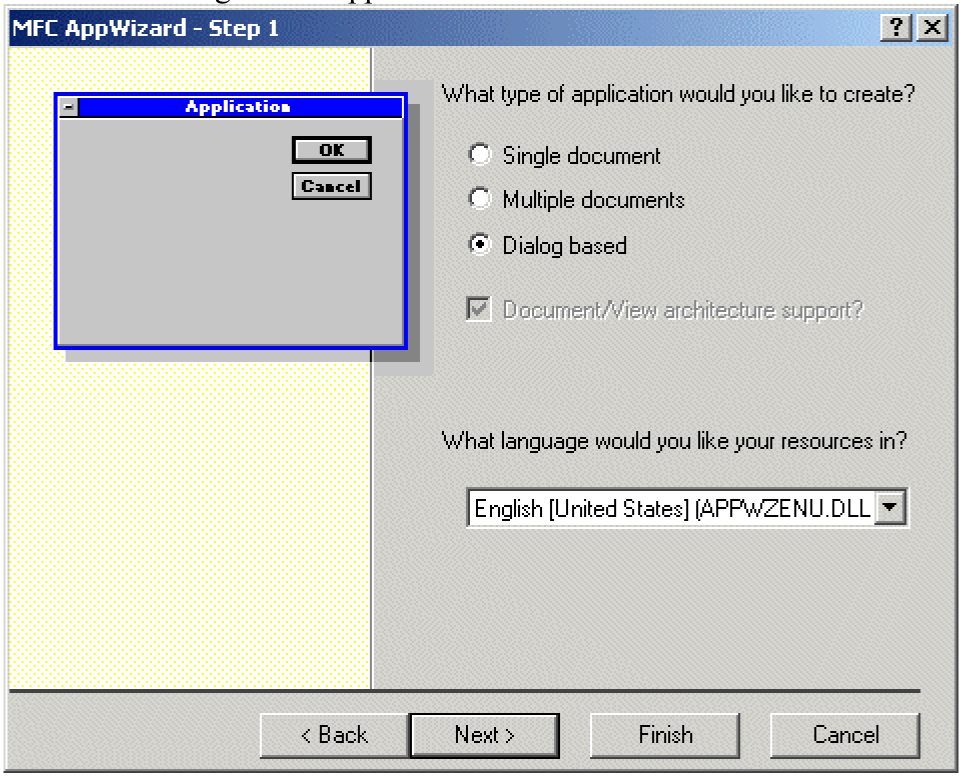
Telos 2101 ActiveX Control with C++

The Telos 2101 ActiveX (TLSAPX) control exposes the application programming interface (API) for System 2101. The specific details are shown for Visual C++ 6.0 but the same ideas can be applied to any other C++ tool.

This example uses a simple dialog-based application to exercise the API functions provided by the TLSAPX control. The example illustrates the basic approach to using the control but it does not exercise the entire API. Although you will have to adapt this example to suit your specific needs, the basic steps should remain the same.

Using the TLSAPX from Visual C++

STEP 1: Start Visual C++. Start a new "MFC AppWizard (exe)" project and click Next. Choose a "Dialog based" application:



MFC AppWizard - Step 1

What type of application would you like to create?

○ Single document
○ Multiple documents
● Dialog based

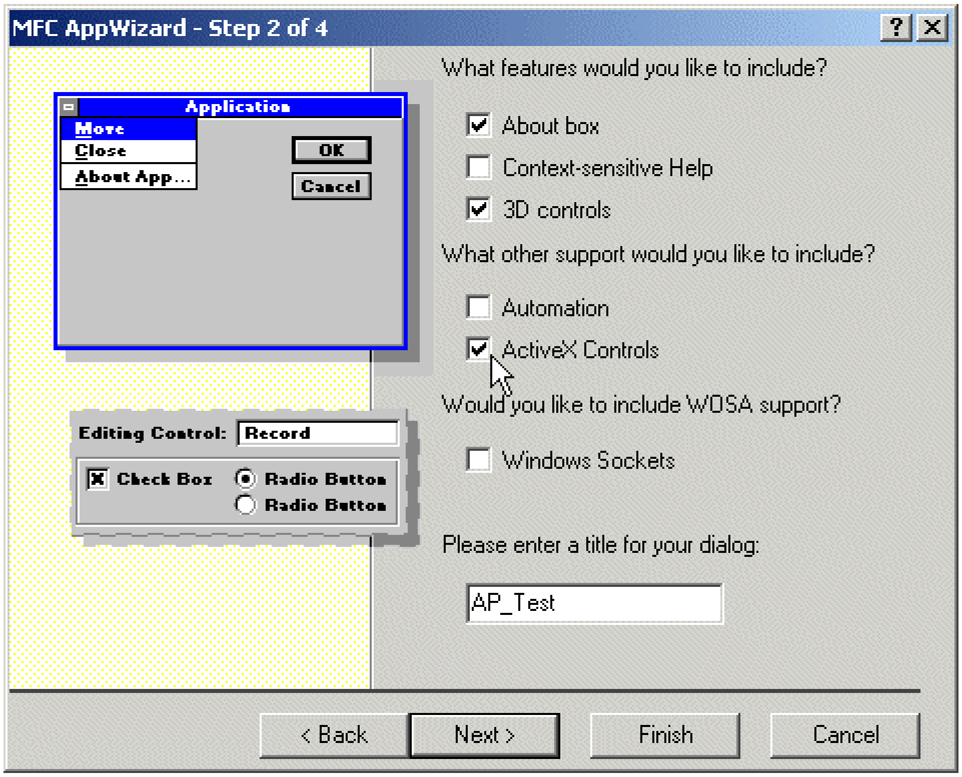
☑ Document/View architecture support?

What language would you like your resources in?

English [United States] (APPWZENU.DLL

Click Next.

On the next screen make sure that the "ActiveX Controls" checkbox is selected:



MFC AppWizard - Step 2 of 4

What features would you like to include?

☑ About box

☐ Context-sensitive Help

☑ 3D controls

What other support would you like to include?

☐ Automation

☑ ActiveX Controls

Would you like to include WOSA support?

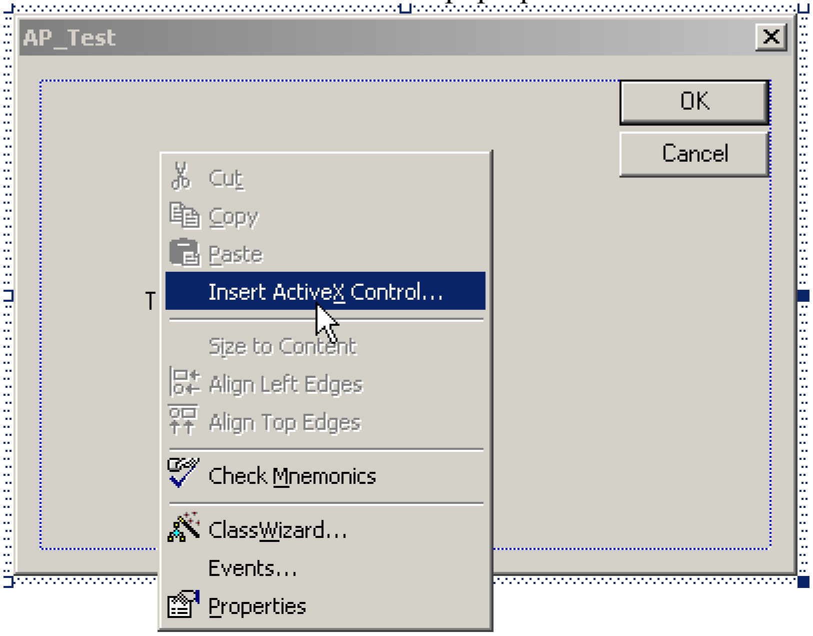
☐ Windows Sockets

Please enter a title for your dialog:

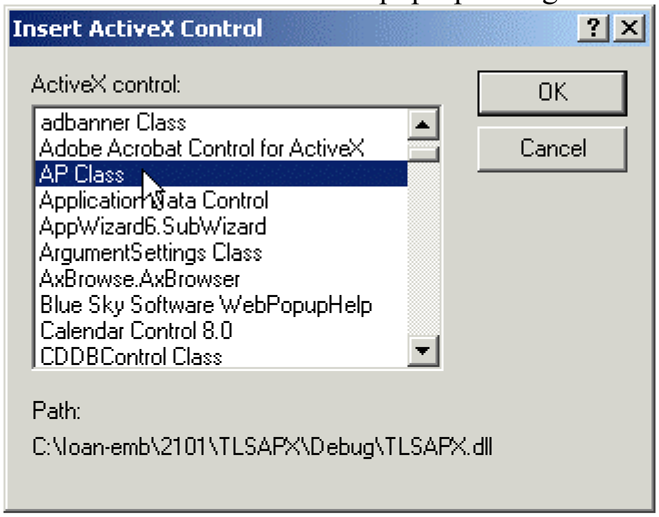
AP_Test

Click Next and complete the AppWizard with your desired defaults.

STEP 2: Go to the dialog view in the resource editor and right-click on the dialog. Select "Insert ActiveX control" from the pop-up menu.



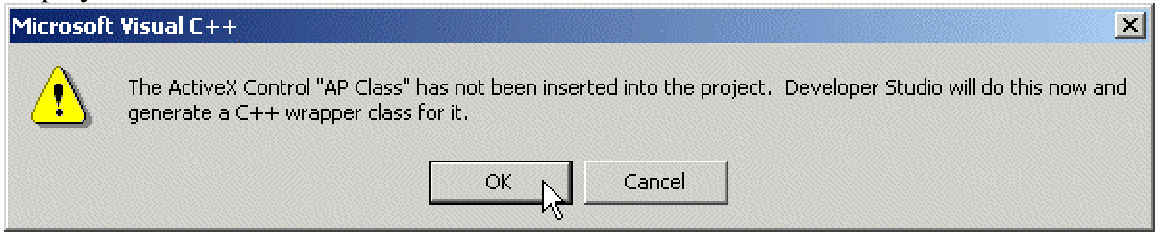
Choose “AP Class” from the pop-up dialog:



Path:
C:\Ioan-emb\2101\TLSAPX\Debug\TLSAPX.dll

A white rectangle will be displayed on the dialog marked with the letters “AP”. Right-click on the rectangle and make sure that the “Visible” property is not selected.

Step 3: Make sure that the AP control is selected and press CTRL-W to bring up the ClassWizard dialog. Click the “Member Variables” tab. Make sure the ID of the AP control is selected then click the “Add Variable” button. The following message will be displayed:



The ActiveX Control "AP Class" has not been inserted into the project. Developer Studio will do this now and generate a C++ wrapper class for it.

Click the OK button and a new class will be generated for the control. When prompted to name the variable enter “m_AP” as the variable name[1].

Step 4: In the OnInitDialog() function for your application dialog add the following lines of code:

```
__try
{
  m_AP.OpenControl();
}
__except(EXCEPTION_EXECUTE_HANDLER)
{
  AfxMessageBox("OpenControl failed!");
  return;
}

AfxMessageBox("OpenControl OK!");
```

---

[1] You may use any name here but the rest of the example will assume the variable is named m_AP.

The thing to note here is that you must use exception handling to obtain the success/failure status of each function call.

Remember that you must also call CloseControl when the application terminates to release any memory used by this control. You would need to detect the application close event and add the following lines:

```
  __try
  {
    m_AP.CloseControl();
  }
  __except(EXCEPTION_EXECUTE_HANDLER)
  {
    AfxMessageBox("CloseControl failed!");
    return;
  }

  AfxMessageBox("CloseControl OK!");
```

STEP 5: Go back to the dialog resource and add a “Connect” button. Use the ClassWizard to add a handler for this button. In the handler use the following code:

```
void CTlsapxVCDlg::OnButtonConnect()
{
  __try
  {
    m_AP.Connect("localhost","user","pass");
  }
  __except(EXCEPTION_EXECUTE_HANDLER)
  {
    AfxMessageBox("Connect failed!");
    return;
  }

  AfxMessageBox("Connect OK!");
}
```

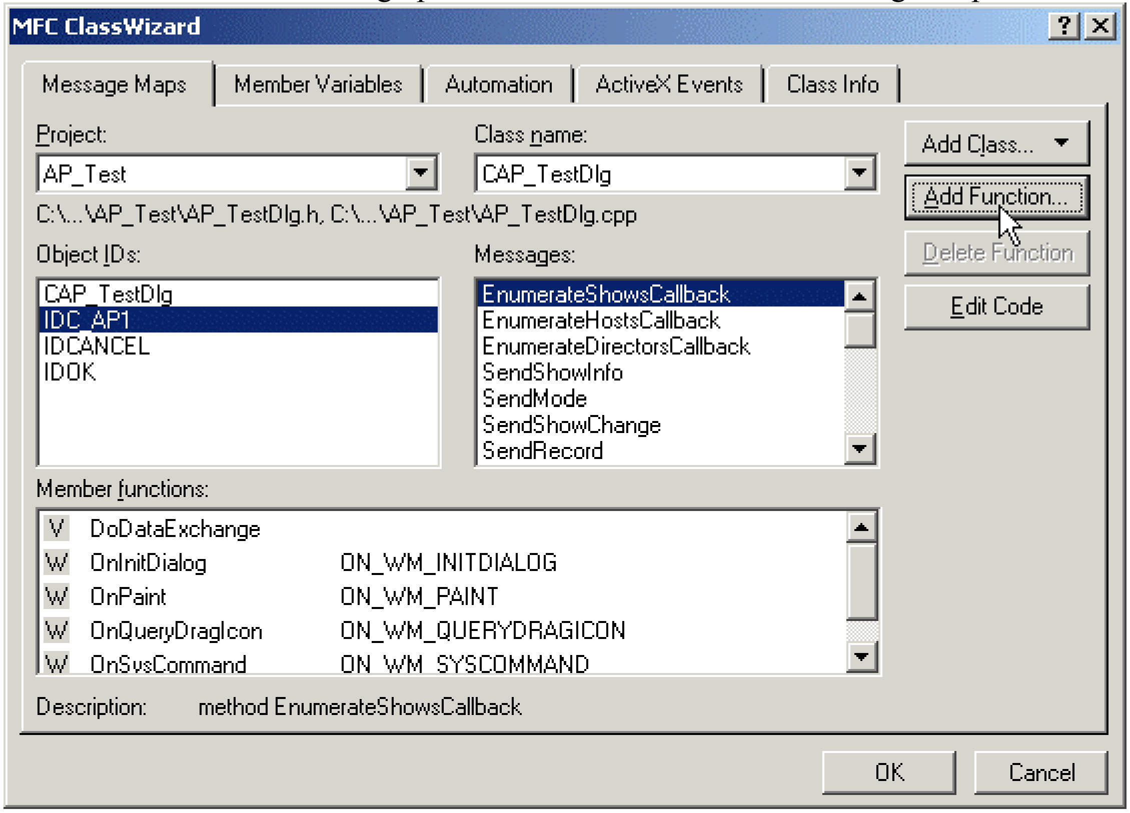
You should replace the connect parameters (host, user name, password) with values that are valid for the 2101 system you are using.

Add another button for “Enumerate shows” with the following handler:

```
void CTlsapxVCDlg::OnButtonEnumerateShows()
{
  __try
  {
    m_AP.EnumerateShows();
  }
  __except(EXCEPTION_EXECUTE_HANDLER)
  {
    AfxMessageBox("EnumerateShows failed!");
    return;
  }

  AfxMessageBox("EnumerateShows OK!");
}
```

STEP 6: We are now ready to add handlers for the events generated by the ActiveX control. Press Ctrl-W to bring up ClassWizard and select the “Message Maps” tab.



Project: AP_Test

Class name: CAP_TestDlg

C:\...\AP_Test\AP_TestDlg.h, C:\...\AP_Test\AP_TestDlg.cpp

Object IDs:
CAP_TestDlg
IDC_AP1
IDCANCEL
IDOK

Messages:
EnumerateShowsCallback
EnumerateHostsCallback
EnumerateDirectorsCallback
SendShowInfo
SendMode
SendShowChange
SendRecord

Member functions:

V	DoDataExchange	
W	OnInitDialog	ON_WM_INITDIALOG
W	OnPaint	ON_WM_PAINT
W	OnQueryDragIcon	ON_WM_QUERYDRAGICON
W	OnSysCommand	ON_WM_SYSCOMMAND

Description: method EnumerateShowsCallback

On the left select the ID of the control then on the right select the message of interest. Click the “Add Function…” button to add a handler for this event. You should add a handler for each of the events provided. When done click the “Edit Code” button.

STEP 7: Add the following code for the EnumerateShowsCallback handler:

```
void CTlsapxVCDlg::OnEnumerateShowsCallbackAp1(LPCTSTR strShowName, long bRequiresPass,
long bIsActive, LPCTSTR strHostName, long bIsLast)
{
  CString str;

  if(bIsActive)
  {
    str.Format("Show %s: Active on '%s': %s %s",
      strShowName,strHostName,
      bRequiresPass ? "REQUIRES PASS" : "NO PASSWORD",
      bIsLast ? "LAST ONE" : "THERE ARE MORE");
  }
  else
  {
    str.Format("Show %s: Not active: %s %s",strShowName,
      bRequiresPass ? "REQUIRES PASS" : "NO PASSWORD",
      bIsLast ? "LAST ONE" : "THERE ARE MORE");
  }

  AfxMessageBox(str);
}
```

Please note that this example handler only displays the information received to the screen. Your application may want to store the information received to display it to the user in a friendlier manner.

Add the following code for the SendError handler:

```
void CTlsapxVCDlg::OnSendErrorAp1(long nErrorNumber, LPCTSTR strErrorText)
{
  CString str;
  str.Format("ERROR(%d): %s",nErrorNumber,strErrorText);
  AfxMessageBox(str);
}
```

STEP 8: We’re now ready to put it all together. Compile the project and run it. The application should display a “OpenControl OK” message. Click the “Connect” button on the dialog. You should receive a “Connect OK” or “Connect failed” message. If connect fails then check to make sure the 2101 system is up and running. Also check the host name, user name and password to make sure they are valid.

Once connect succeeds click the “Enumerate Shows” button. You should receive one message for each show defined in the system. The message box will indicate if the show is active and if so the name of the host where it is running.

You should also notice that the SendError handler receives events from the control. Even on success the SendError handler receives messages to indicate success. Success codes are less than 1000 while error codes are greater than 1000.

Summary

This example application showed how to use the 2101 ActiveX control from a C++ environment. The specific details are for Visual C++ 6.0 but the same ideas can be applied to other C++ tools.

Although this example does not cover the entire 2101 API it illustrates the basic techniques for communicating with the ActiveX control. The example can be easily extended to cover all functionality needed by your application.