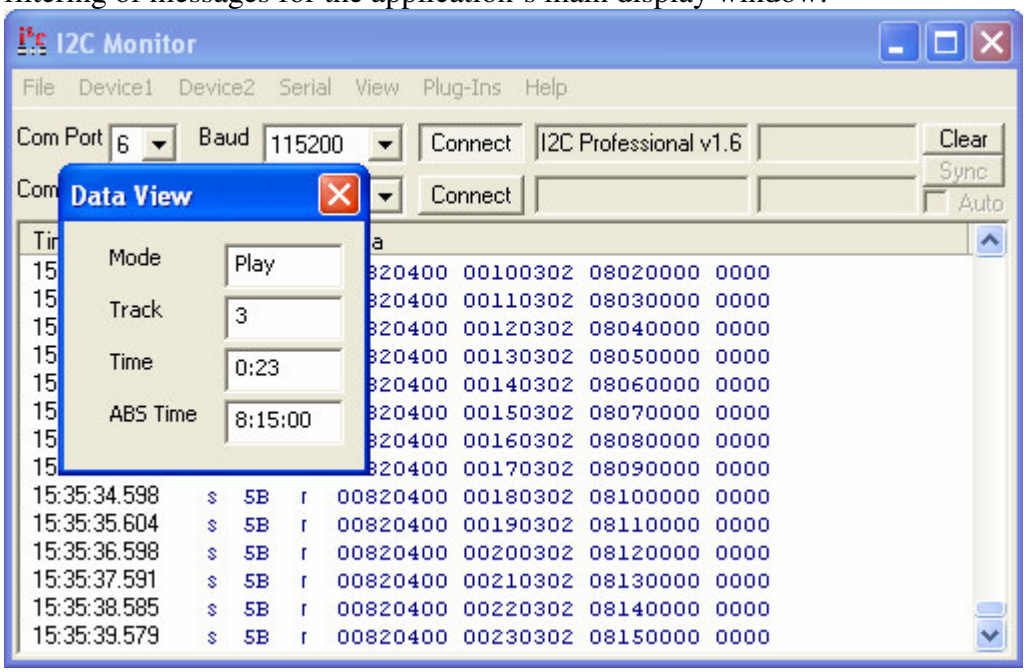
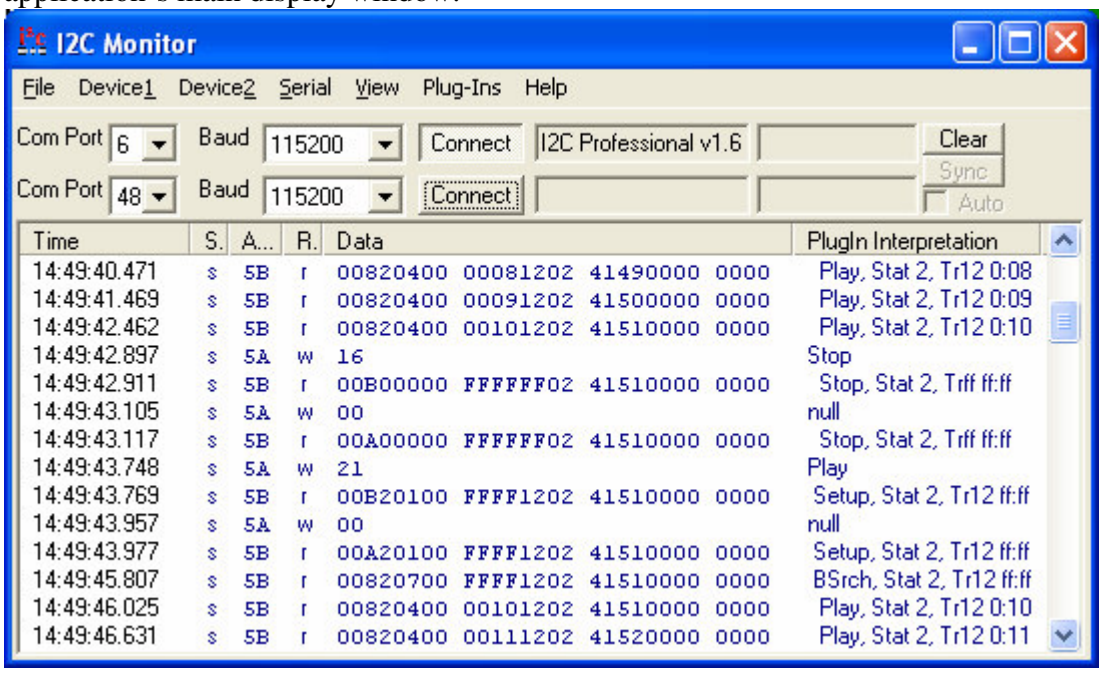
Plug-In modules for the AVIT I²C Monitor Application

The I²C Monitor application can be extended to provide message decoding for specific devices, extra filtering or alternative display formats. This achieved by building plug-in code modules that are automatically loaded by the I²C Monitor application and are passed I²C and serial messages.
Two types of plug-in are available. A View Plug-in is a dialog window that can be used to interpret messages and display them in suitable MFC controls. It can also provide advanced filtering of messages for the application's main display window.



Interpreter plug-ins do not have their own window but are passed the received messages by the application and they return a string that is displayed in an extra column of the application's main display window.



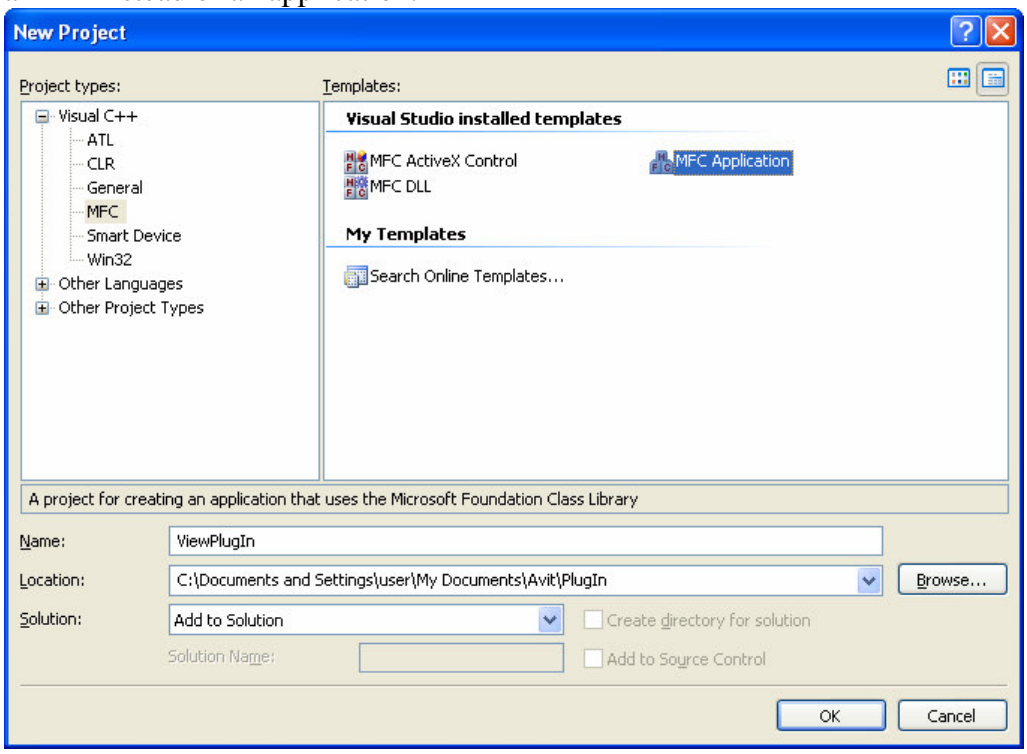
Running the supplied sample Plug-Ins

This document comes with a set of sample code for each type of plug-in. These should be unzipped to a suitable working folder and the project file opened in Visual Studio. A build of the project should succeed and place the DLL file in the same folder as the I^2C Monitor application (C:\Progam Files\AVIT Research\I2C Monitor). When the Monitor application runs the DLLs should be listed under the Plug-Ins menu where they can be enabled and disabled.

Creating a View Plug-In using Visual Studio 2005

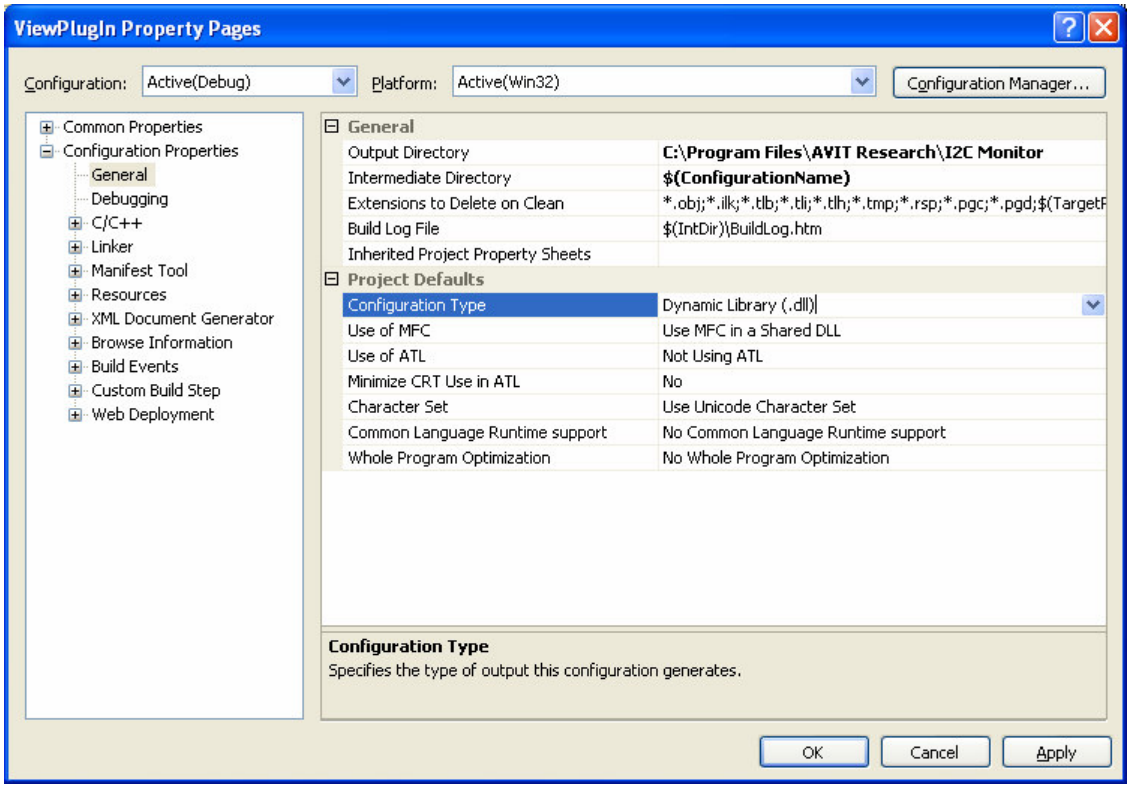
Create a new project

For a View Plug-in the type of project should be an MFC Application. This project type will create the dialog object automatically and later on in the process we will change it to produce a DLL instead of an application.

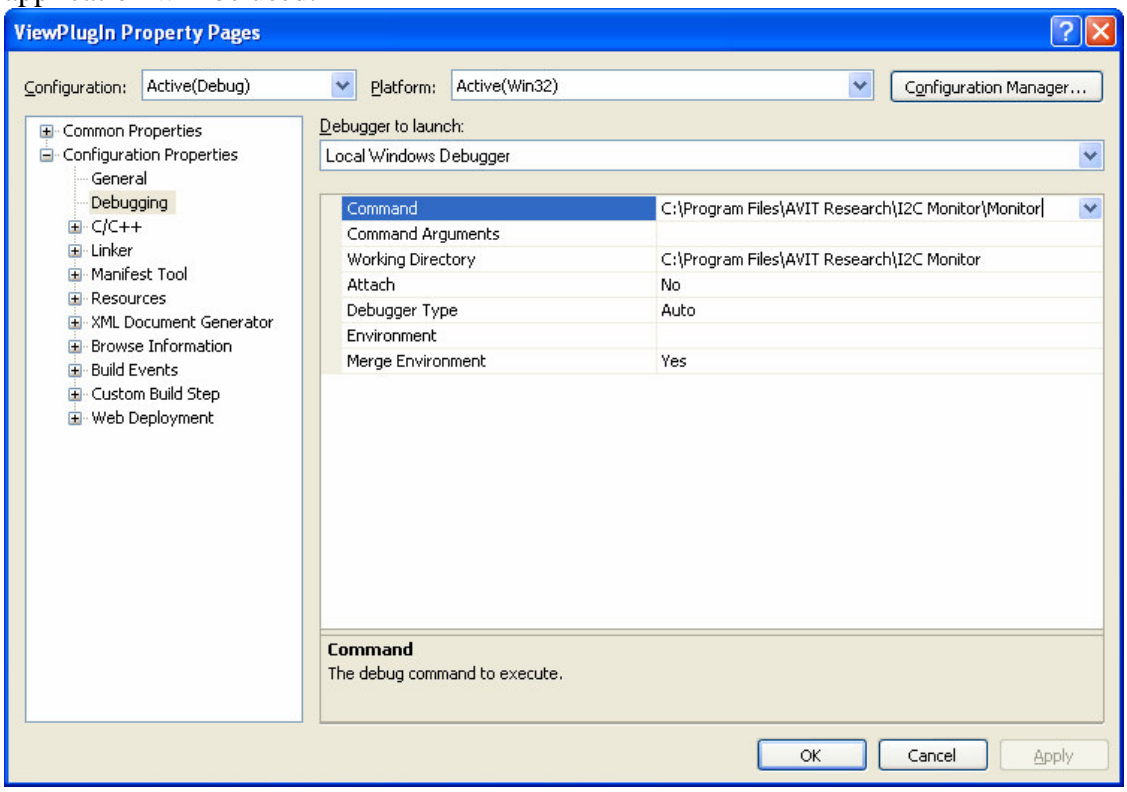


In the setup wizard, select the type as dialog based application. All other settings should be fine at their defaults.

Once created select the project properties and change the application type to “Dynamic Library (.dll)”. Change the Output Directory to the same folder as the application e.g. “C:\Program Files\AVIT Research\I2C Monitor”.



Also, under the debugging tab set the command and working directory so that the I^2C Monitor application will be used:



Modify the standard files to add the Dll functions.

ViewPlugIn.h

Add prototypes for “ShowCustomWindow” and “ProcessI2cMessage”.

```
class CViewPlugInApp : public CWinApp
{
public:
        CViewPlugInApp();

// Overrides
        public:
        virtual BOOL InitInstance();

// Implementation

        DECLARE_MESSAGE_MAP()
};

extern CViewPlugInApp theApp;

▶▶▶extern "C" __declspec(dllexport) void ShowCustomWindow(CWnd * pWnd);
▶▶▶extern "C" __declspec(dllexport) int ProcessI2cMessage(int Interface, int Address, unsigned
char * Data, int DataLen);
```

ViewPlugIn.cpp

Change InitInstance so it doesn’t create the window. Delete the lines:

```
        CViewPlugInDlg dlg;
        m_pMainWnd = &dlg;
        INT_PTR nResponse = dlg.DoModal();
        if (nResponse == IDOK)
        {
                // TODO: Place code here to handle when the dialog is
                //  dismissed with OK
        }
        else if (nResponse == IDCANCEL)
        {
                // TODO: Place code here to handle when the dialog is
                //  dismissed with Cancel
        }
```

And replace with:

```
        m_pMainWnd = NULL;
```

Change the last line of the function to read:

```
        return TRUE;
```

This is because we want the window to be a child of the Monitor application so we need the call from the application to pass a handle to it’s window that we can use as the parent of the dialog window we create.

ViewPlugInDlg.h

Add prototypes for the “ShowCustomWindow” and “ProcessI2cMessage” methods.

```
// CCanViewDlg dialog
class CViewPlugInDlg : public CDialog
{
// Construction
public:
        CViewPlugInDlg(CWnd* pParent = NULL); // standard constructor
▶▶▶     void ShowCustomWindow(CWnd * pWnd);
▶▶▶     int ProcessI2cMessage(int Interface, int Address, unsigned char * Data, int DataLen);
```

ViewPlugInDlg.cpp

Add Dll functions and the calls to the equivalent methods in the Dialog object.

```
extern "C" __declspec(dllexport) void ShowCustomWindow(CWnd * pWnd)
{
	((CViewPlugInDlg *)theApp.m_pMainWnd)->ShowCustomWindow(pWnd);
}

extern "C" __declspec(dllexport) int ProcessI2cMessage(int Interface, int Address, unsigned
char * Data, int DataLen)
{
	return ((CViewPlugInDlg *)theApp.m_pMainWnd)->ProcessI2cMessage(Interface, Address,
Data, DataLen);
}
```

Add the method for the ShowWindow call:

```
void CViewPlugInDlg::ShowCustomWindow(CWnd * pWnd)
{
	AFX_MANAGE_STATE(AfxGetStaticModuleState());
	if(theApp.m_pMainWnd == NULL)
	{
		CViewPlugInDlg * dlg = new CViewPlugInDlg(pWnd);
		dlg->Create(IDD_DIALOG1, pWnd);
		dlg->OnInitDialog();
		theApp.m_pMainWnd = dlg;
	}
	if(theApp.m_pMainWnd)
	{
		theApp.m_pMainWnd->ShowWindow(SW_RESTORE);
	}
}
```

This creates the dialog window object if it doesn't already exist and uses the handle to the parent window that is passed into the function as the initialisation parameter. It then makes the window visible. Code could be inserted here to restore the position of the window from registry settings.

Add the method for the "ProcessI2cMessage" call:

```
int CViewPlugInDlg::ProcessI2cMessage(int Interface, int Address, unsigned char * SerData, int
DataLen)
{
	AFX_MANAGE_STATE(AfxGetStaticModuleState());

	if((Interface == 0) || (Interface == 1))	// I2c interfaces
	{
		if(Address == 0x5A)
		{
			return -1;		// Don't show message in Monitor app
						// or pass message to other plug-ins
		}
		else if(Address == 0x5B)
		{
			return 1;		// Don't pass message to other plug-ins
		}
	}
	else if((Interface == 2) || (Interface == 3))		// Serial interfaces
	{
		// Address is zero for serial interfaces
	}

	return 0;
}
```

This is where the code should be placed for interpreting the data and writing it to the custom window. A return value of zero indicates that the message wasn't processed by the plug-in and it should be passed to the other plug-ins or displayed in the Monitor list display. A return value of 1 indicates that the Monitor should not pass the message to other plug-ins but should still show it in the list display. A return value of -1 indicates that the message should not be passed onto the other plug-ins and it should not be shown in the Monitor list display. Using

these return values the plug-in can perform more sophisticated filtering than the Monitor achieves normally.

Dialog Resource

The dialog window "Application Window" property should be changed to FALSE so that the plug-in window does not create its own entry on the windows task bar.

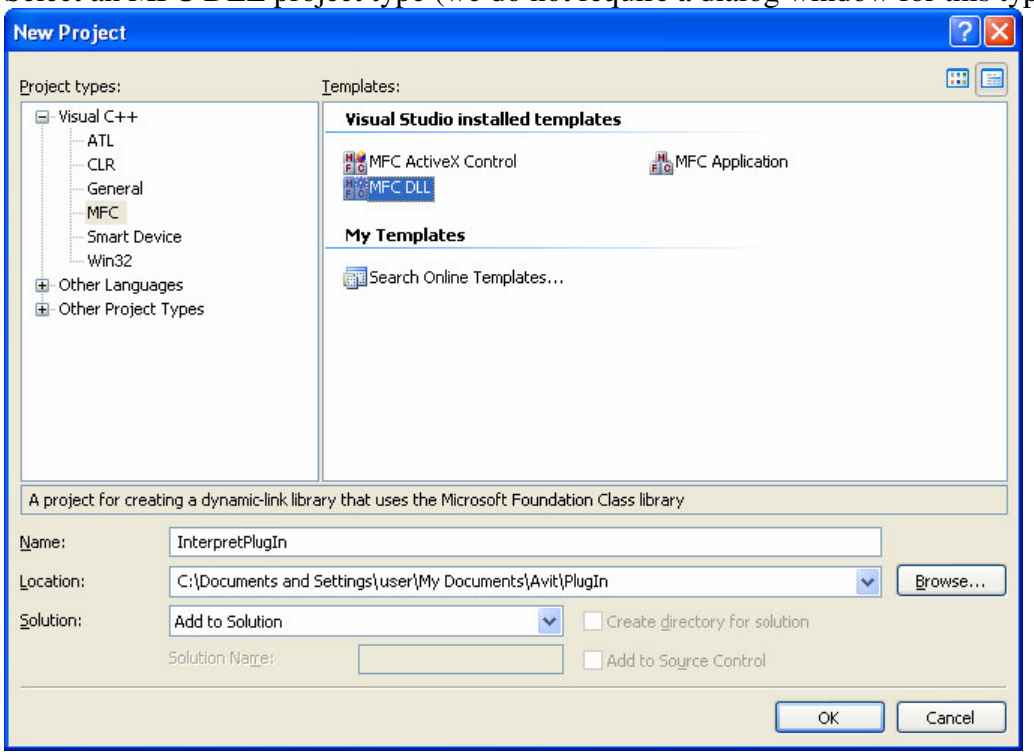
Build and Run the Plug-In

When the plug-in is built it will place the DLL file in the same directory as the I^2C Monitor application. Selecting RUN will run the application and allow the plug-in to be debugged.

Creating an Interpreter Plug-In using Visual Studio 2005

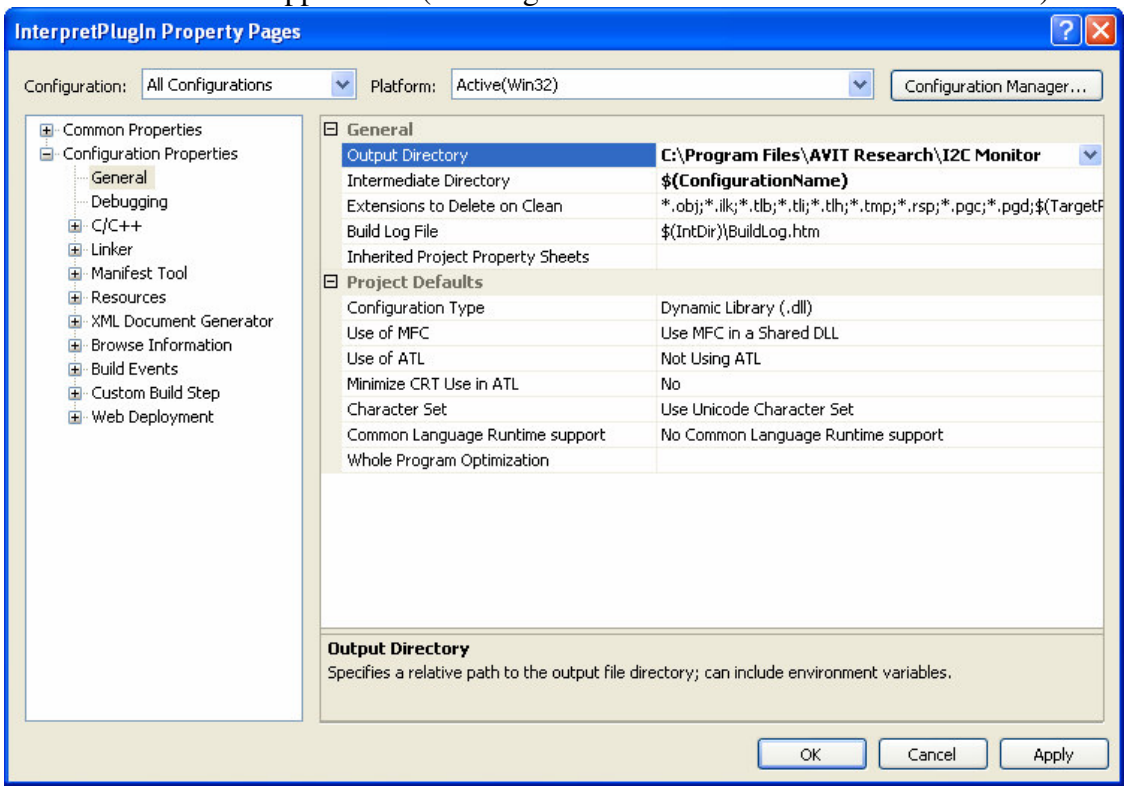
Create a new project

Select an MFC DLL project type (we do not require a dialog window for this type of plug-in).

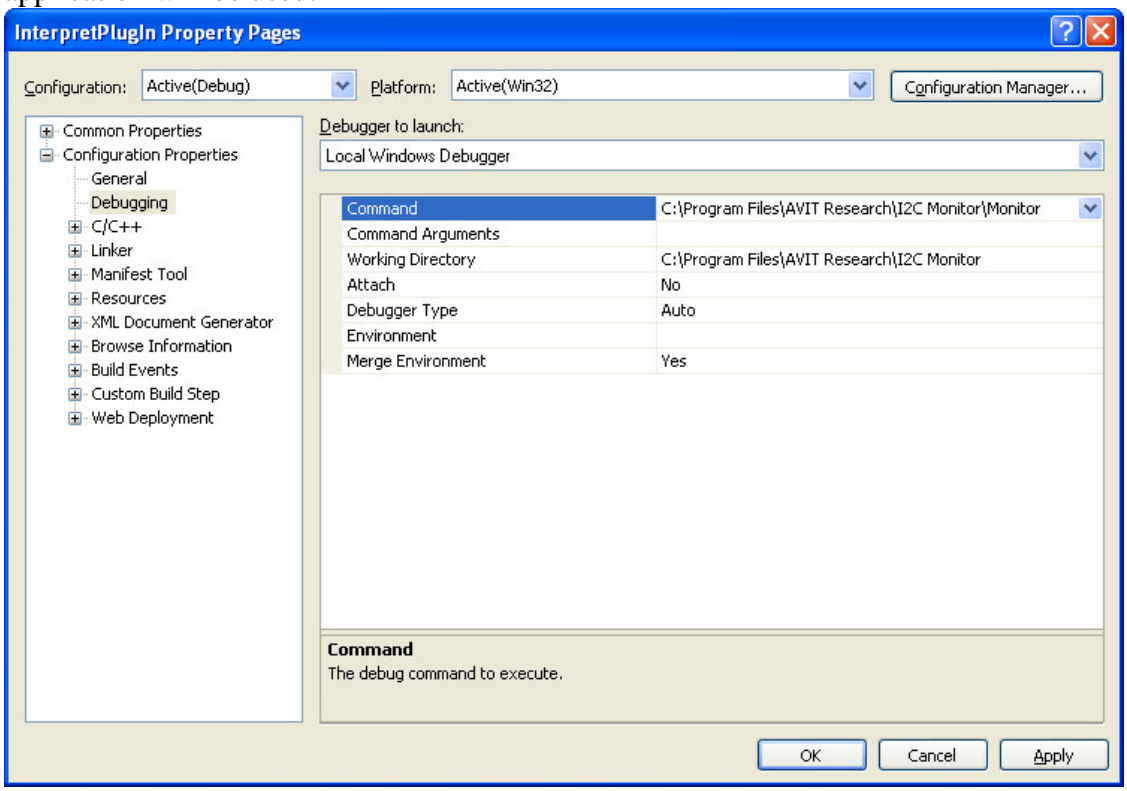


Use the default options of the wizard.

Once created, change the project “Output Directory” property to place the output DLL in the same location as the application (i.e. Program Files\AVIT Research\I2C Monitor):



Also, under the debugging tab set the command and working directory so that the I²C Monitor application will be used:



Modify the standard files to add the Interpreter function

InterpretPlugIn.h

Add prototype for the Interpret method.

```
class CInterpretPlugInApp : public CWinApp
{
public:
        CInterpretPlugInApp();
▶▶▶     int Interpret(int Interface, int Address, unsigned char * Data,
                                    int DataLen, char * RetString, int RetStringLen);
```

InterpretPlugIn.cpp

Add code to implement the Interpret method. The Interface parameter has values 0 and 1 for the two I²C interfaces. The string that is produced by the Interpret method is returned in the location pointed by the RetString parameter which can take up to RetStringLen characters (including the null terminator). Returning a value of 1 indicates that a string is being returned and other interpreter DLLs should not be passed the message. Returning a value of zero indicates that the message should be passed to other interpreter plug-ins. Returning a value of -1 will prevent the message from being shown in the Monitor application list display.

```
int CInterpretPlugInApp::Interpret(int Interface, int Address,
                                    unsigned char * Data, int DataLen,
                                    char * RetString, int RetStringLen)
{
        if(RetString && (DataLen > 0))
        {
                if(Address == 0xC0)
                {
                        strcpy_s(RetString, RetStringLen, "Write to EEPROM");
                        return 1;       // Returning a string
                }
                else if(Address == 0xC1)
                {
```

```
                        strcpy_s(RetString, RetStringLen, "Read from EEPROM");
                        return 1;       // Returning a string
                }
        }
        return 0;               // Not processed
}
```

InterpretPlugIn.def

Add the method to the EXPORTS list.

```
EXPORTS
▶▶▶     Interpret
```

Build and Run the Plug-In

When the plug-in is built it will place the DLL file in the same directory as the I^2C Monitor application. Selecting RUN will run the application and allow the plug-in to be debugged.