TYX Corporation

Productivity Enhancement Systems

Reference	TYX_0051_1
Revision	1.1
Document	MFCClientForTpsServer.doc
Date	January 21, 2003

MFC Client for TpsServer

The purpose of this document is to write a client for the TpsServer.

Knowledge of MFC and COM is recommended, but not required. For those that have questions about MFC and/or COM, I would recommend that they refer to literature that addresses those issues since this document will not attempt to teach you either one even if it will provide a lot of details on how to make this client sample work.

The usage of the TpsServer will be asynchronous.

It will allow the client to access the TpsServer functionality, and it will also process the notifications from the TpsServer.

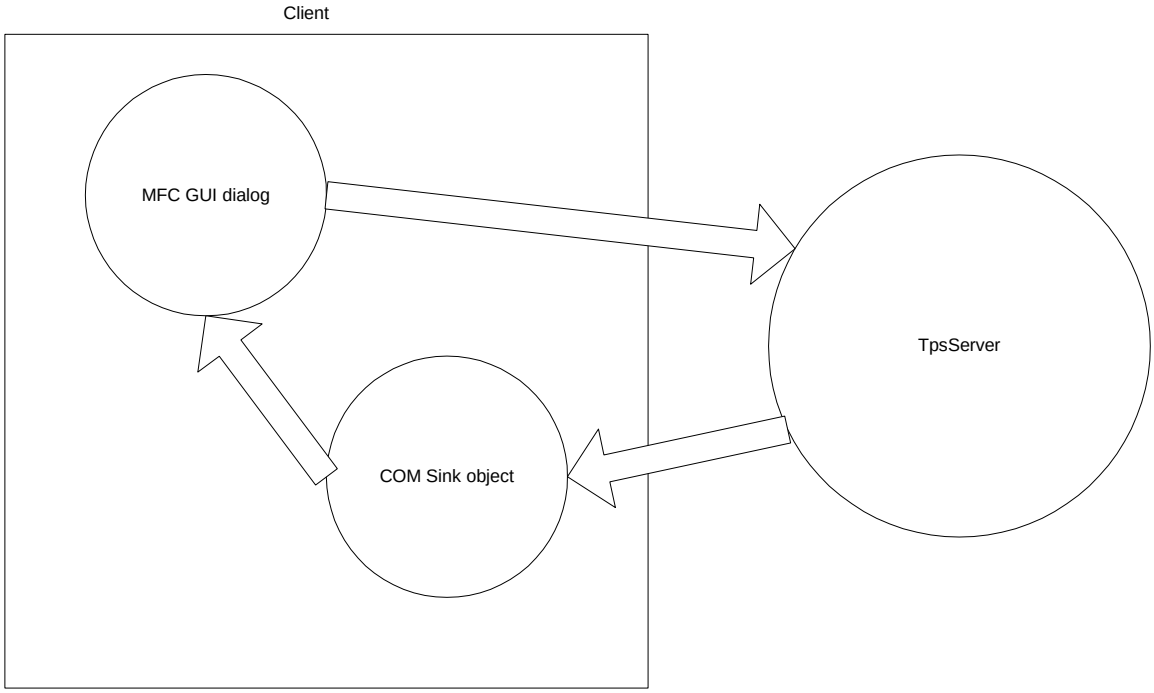
The client will include an MFC GUI.

The commands to the Wrts will go directly to the TpsServer.

The notification that comes from the TpsServer will go through a COM Sink client.

Notes:

- It would be possible for the Wrts to notify the MFC GUI directly. This architecture is however considered to be the correct way to access a dialog.
- This MFC dialog is a sample. You may change this sample to meet your requirements. You will be however likely to reuse the COM object.



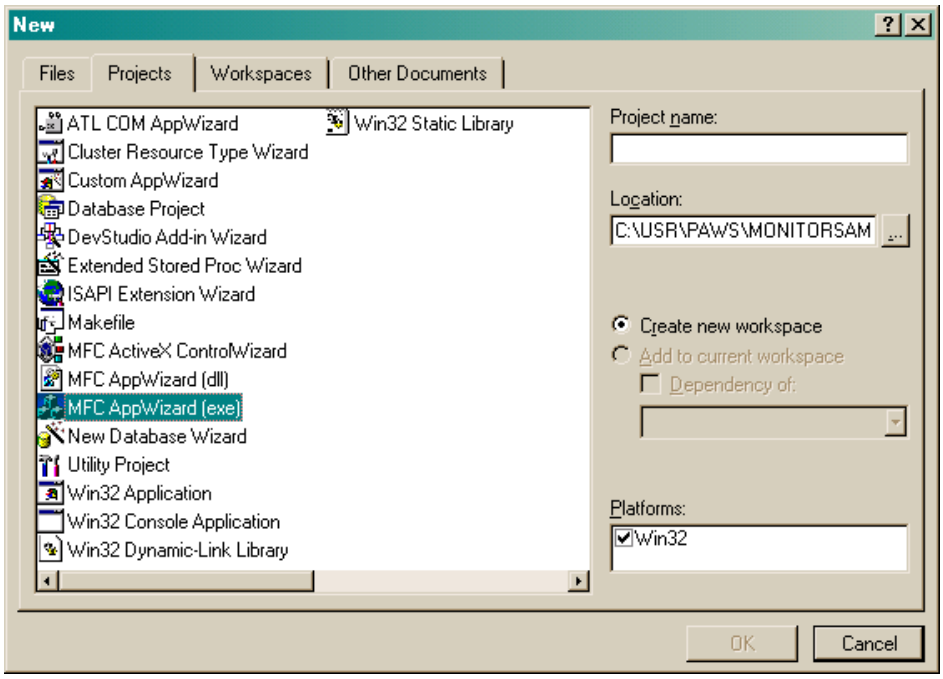
We will first generate the dialog and allow the MFC GUI dialog buttons to control the TpsServer.

As we generate the COM Sink object to deal with the notifications, we will add the functionality that will allow for the COM Sink object to be generated and for the connectivities to be made.

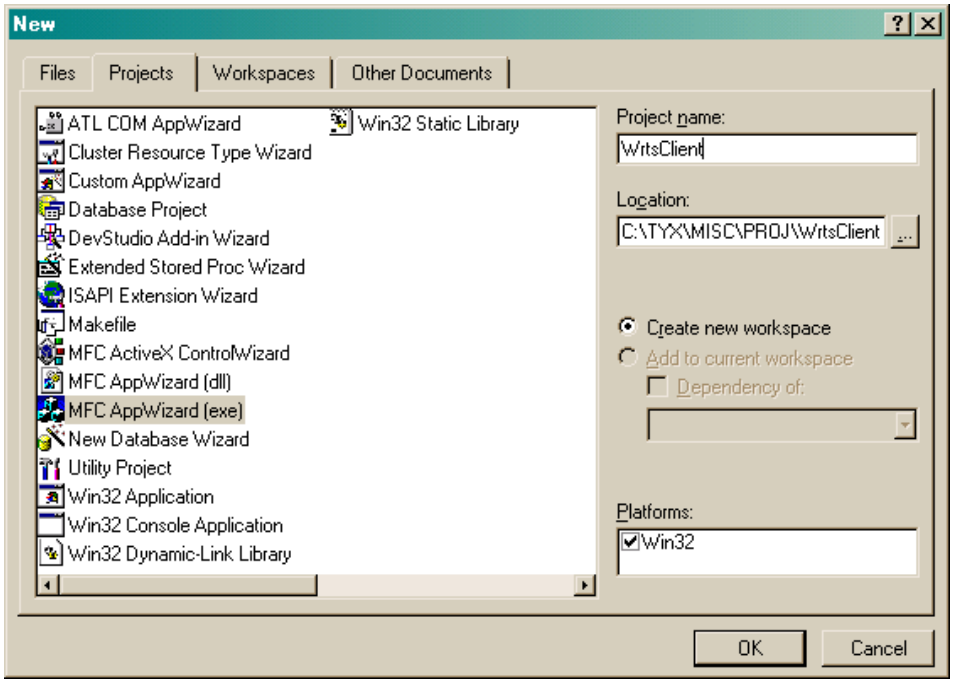
1 Creating the MFC GUI dialog

1.1 Adding Buttons

You will want to create a new **MFC AppWizard (exe)**.

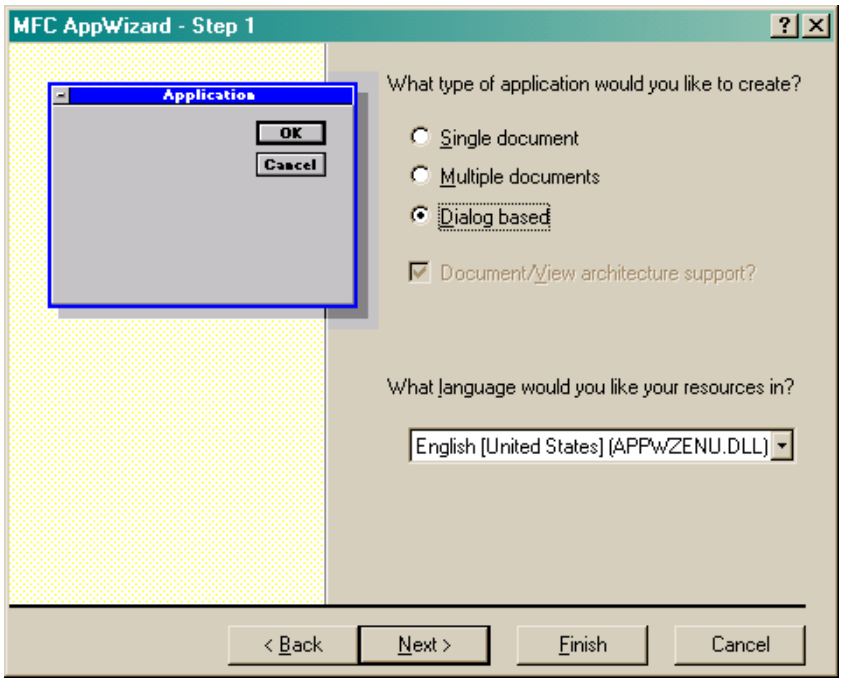


The name will be **WrtsClient**



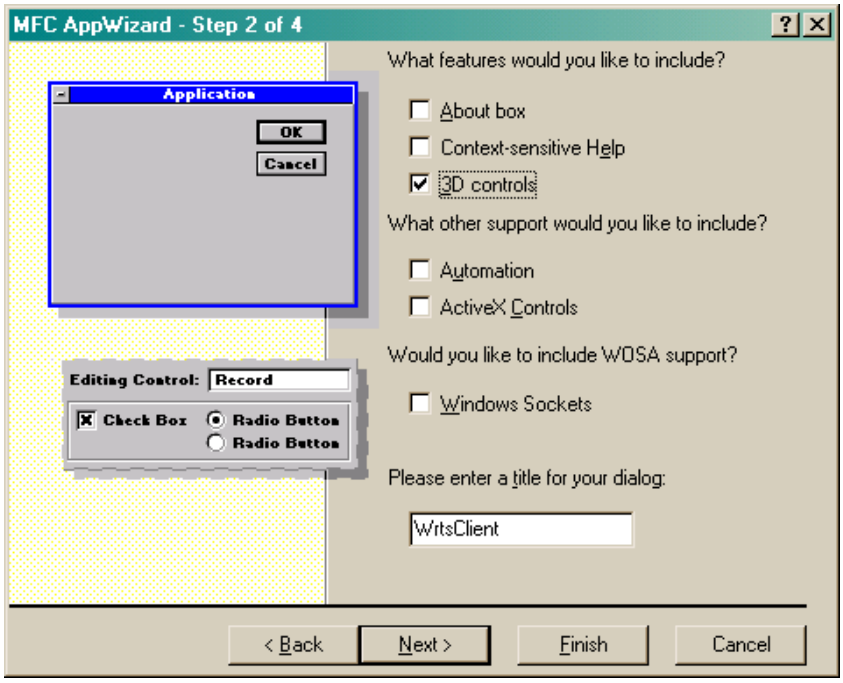
Click on **OK**.

Select **Dialog based**



Click on **Next >**.

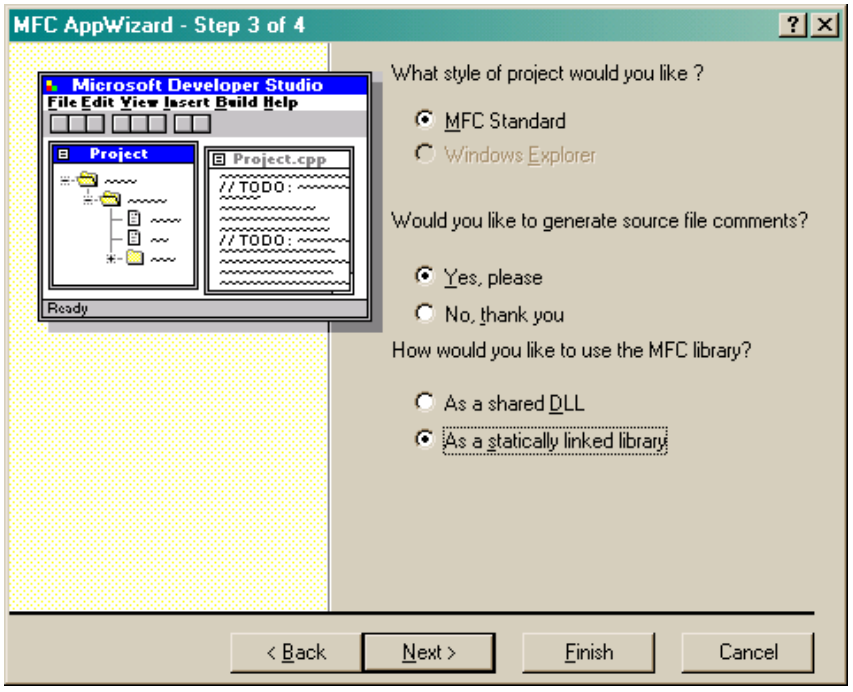
Unselect **Active X Controls** and **About box**.



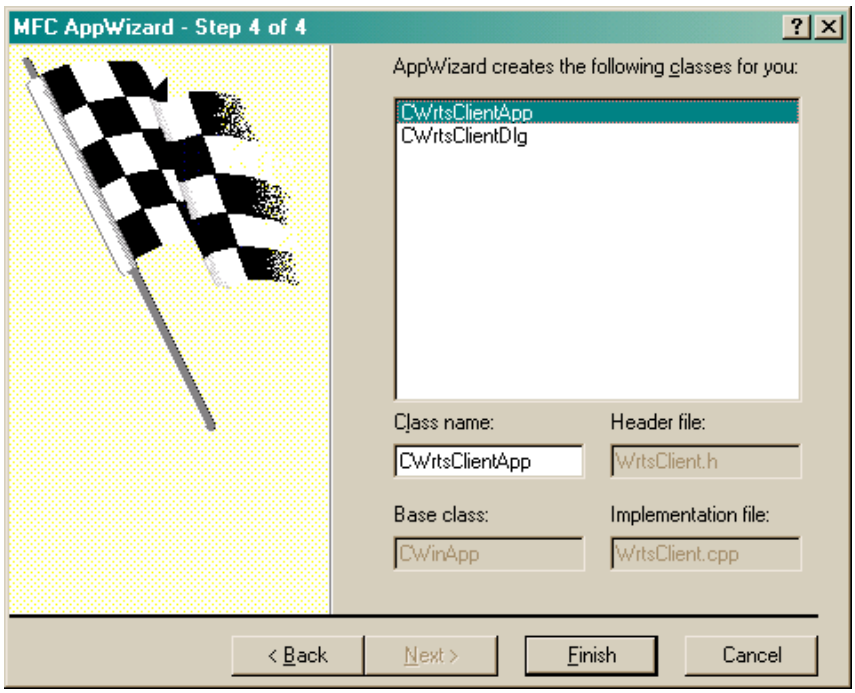
And then press on **Next >**.

Include MFC statically in order to be less dependent upon the OS environment. This would be a necessary step for usage of the client on an embedded system. For a complete OS, you may be able to use MFC dynamically in order to generate a smaller exe.

In our case, we will select **As a statically linked library**.



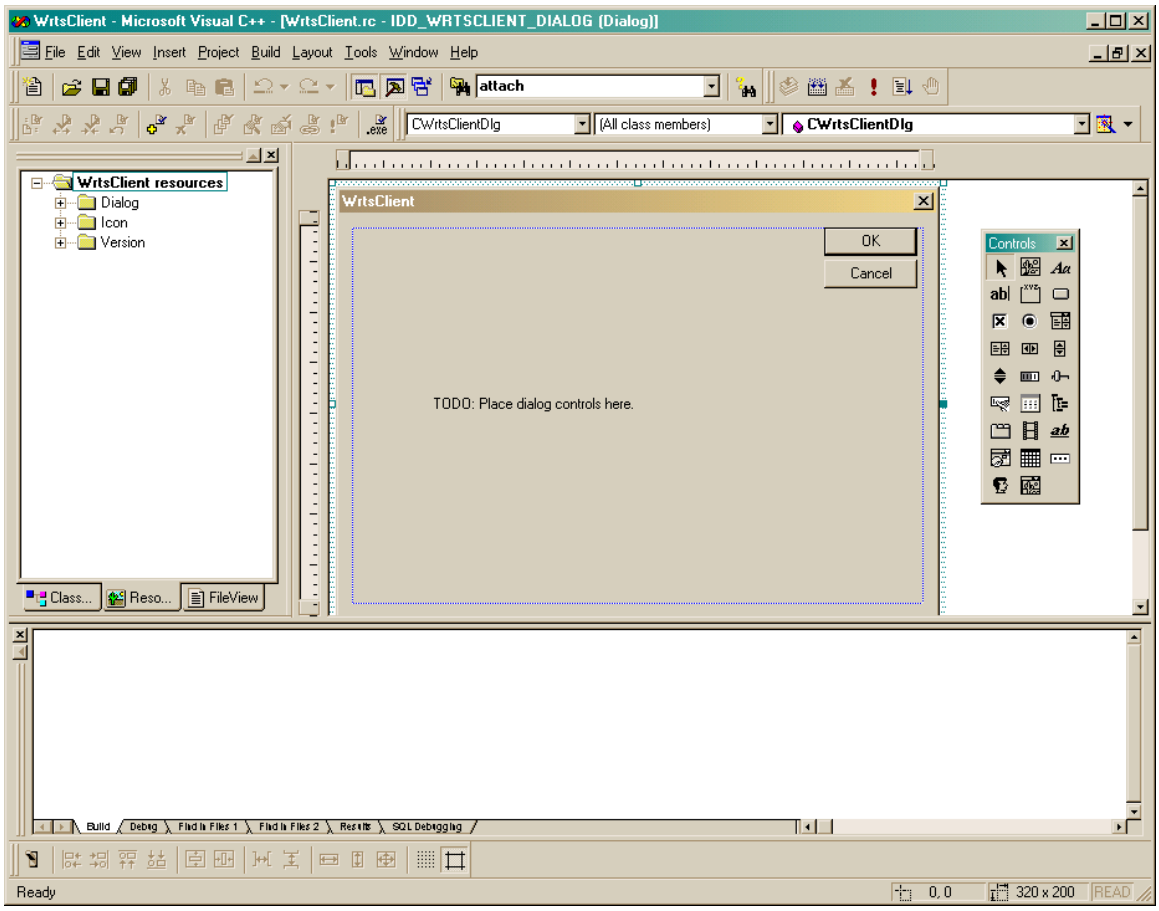
We can press **Next >** and accept the default class name.



Press **Finish**.

Click **OK** on the **New Project Information** window that follows.

You should have access to the following window:

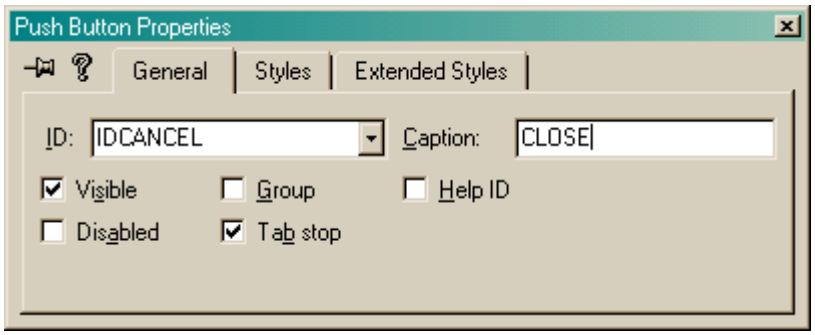


Our first step is to verify that the code builds and runs.

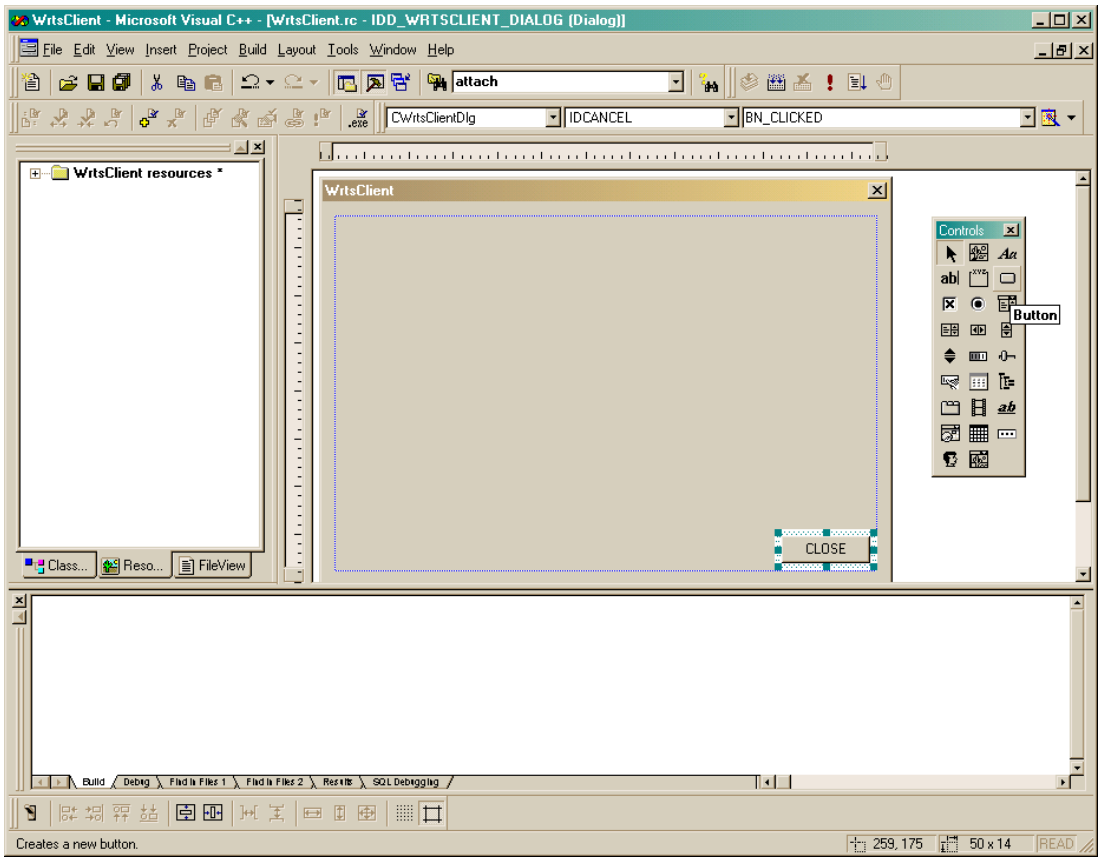
The purpose at this point is to delete the unnecessary buttons and add the ones that we want to support.

- Select the **OK** button by left clicking on it in the dialog box, and press delete.
- Select **TODO: …** by left clicking on it in the dialog box, and press delete.
- We will move the **Cancel** button. To rename the caption, right click on the **Cancel** button and select **Properties**. Change the caption to **CLOSE** and close

the window. It this window, you could also change the ID if you wanted. We will leave the current one for **CLOSE**.



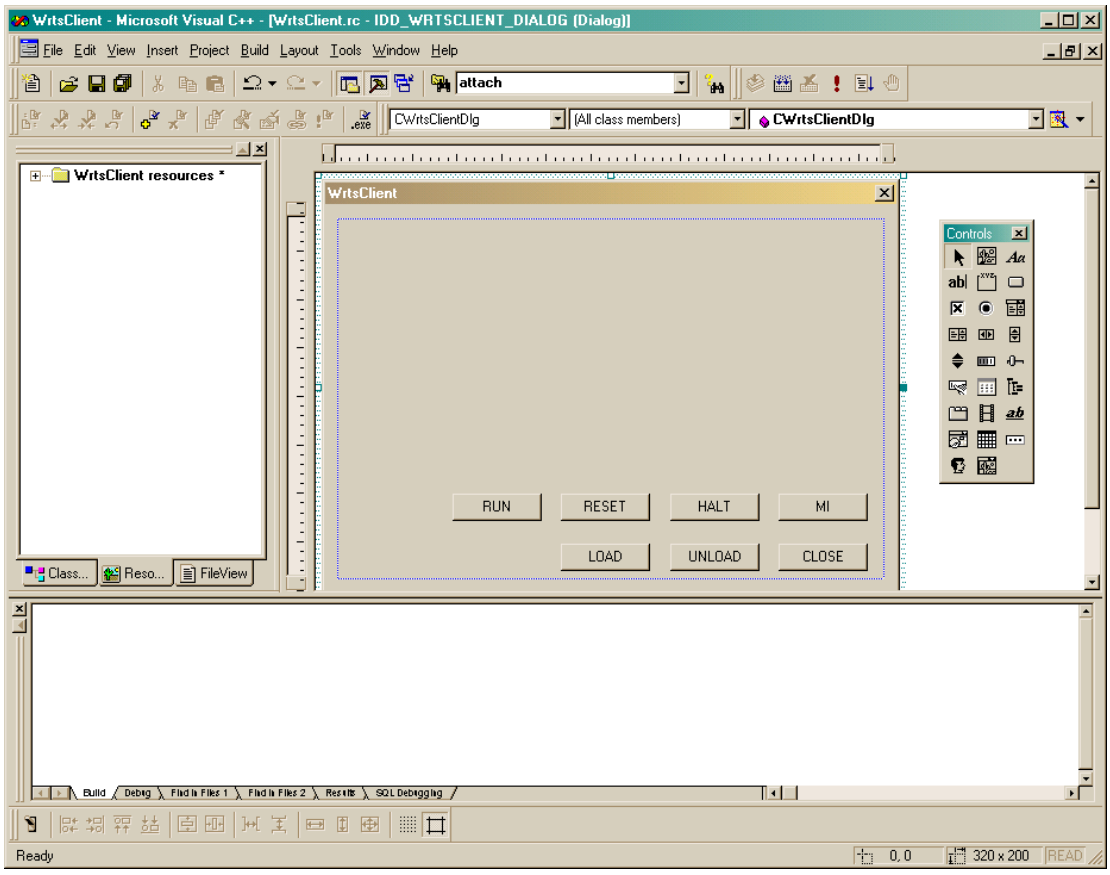
- We now want to add some buttons with the following **ID:** and the **Caption:**



- Select the button from the toolbar and drag and drop a button onto the dialog window. Copy and paste this button 5 time and change the properties of those buttons to match the content of the following table.

ID:	Caption:
IDC_RUN	RUN
IDC_RESET	RESET
IDC_HALT	HALT
IDC_MI	MI
IDC_LOAD	LOAD
IDC_UNLOAD	UNLOAD

- Align the buttons as needed.
- You should have a dialog box that may look like this one.



Make sure that you can build without errors and run the exe.

We will now add the functionality to the dialog box.

1.2 Connecting to the TpsServer

The purpose of what we want to do now is to load the TpsServer when you open the client. We will want to unload when we close the client.

For this, we will add some code that you will want to add, no matter what your client looks like:

- In order to give visibility throughout the code, you will want to add the following at the bottom of the **StdAfx.h** file. The new code is black and the existing code is gray:

```
#include <afxwin.h>         // MFC core and standard components

#include <afxext.h>         // MFC extensions

#include <afxdtctl.h>            // MFC support for Internet Explorer 4
Common Controls

#ifndef _AFX_NO_AFXCMN_SUPPORT

#include <afxcmn.h>              // MFC support for Windows Common
Controls

#endif // _AFX_NO_AFXCMN_SUPPORT



// New Lines added by TYX

#import "C:\usr\tyx\com\RtsAx.dll" raw_interfaces_only \

       raw_native_types, named_guids rename_namespace("RTSAX")

#include <atlbase.h>

void handleError(HRESULT hr);



//{{AFX_INSERT_LOCATION}}

// Microsoft Visual C++ will insert additional declarations immediately before the previous line.
```

Please note that the path to the **RtsAx.dll** should be the one that you have on your computer. The **#include** and the **#import** are necessary. The **handleError** will contain a function that will handle errors. The existence of this function is not vital, but the function that we propose and it's content are highly recommended.

- The code for the **handleError** will be placed in the **StdAfx.cpp** file in order to be accessed from anywhere:

```
#include "stdafx.h"


void handleError(HRESULT hr)
{
    if (SUCCEEDED(hr)) return;

    USES_CONVERSION;
    CComPtr<IErrorInfo> pErrorInfo;
    CString errMsg;
    if (::GetErrorInfo(0, &pErrorInfo) == S_OK)
    {
            CComBSTR bstrDscr;
            if (SUCCEEDED(pErrorInfo->GetDescription(&bstrDscr)))
                    errMsg.Format(_T("COM Error: 0x%08X %s"), hr, OLE2T(bstrDscr));
            else
                    errMsg.Format(_T("COM Error: 0x%08X"), hr);
    }
    else
    {
            // ::GetErrorInfo may return S_FALSE
            TCHAR* pszMsg = NULL;
            DWORD nChars = ::FormatMessage(
```

```
                            FORMAT_MESSAGE_ALLOCATE_BUFFER |
FORMAT_MESSAGE_FROM_SYSTEM,

                            NULL, hr, MAKELANGID(LANG_NEUTRAL, SUBLANG_DEFAULT),

                            (LPTSTR)&pszMsg, 0, NULL);


            if (nChars)

                    errMsg.Format(_T("COM Error: 0x%08X %s"), hr, pszMsg);

            else

                    errMsg.Format(_T("COM Error: 0x%08X"), hr);

            // Free the buffer.

            if (pszMsg != NULL)

                    ::LocalFree((HLOCAL)pszMsg);

    }

    // Display the com error:

    ::MessageBox(NULL, errMsg, _T("Error"), MB_OK);

}
```

- In the **WrtsClientDlg.h** file, you will want to create a smart pointer by adding the following code:

```
            // Generated message map functions

            //{{AFX_MSG(CWrtsClientDlg)

            virtual BOOL OnInitDialog();

            afx_msg void OnPaint();

            afx_msg HCURSOR OnQueryDragIcon();

            //}}AFX_MSG

            DECLARE_MESSAGE_MAP()
```

```
        CComPtr<RTSAX::ITpsServerEx> m_pTpsServerEx;
};
```

- Now we can and want to CoCreate an instance of the TpsServer in the **WrtsClientDlg.cpp OnInitDialog** method. We also want to Attach:

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

// 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
        // create TpsServer
        HRESULT hr = m_pTpsServerEx.CoCreateInstance(RTSAX::CLSID_TpsServer);
        if (FAILED(hr)) { handleError(hr); return FALSE; }

        // attach tpsserver to wrts
        hr = m_pTpsServerEx->Attach();
        if (FAILED(hr)) { handleError(hr); return FALSE; }

        return TRUE;  // return TRUE  unless you set the focus to a control
}
```

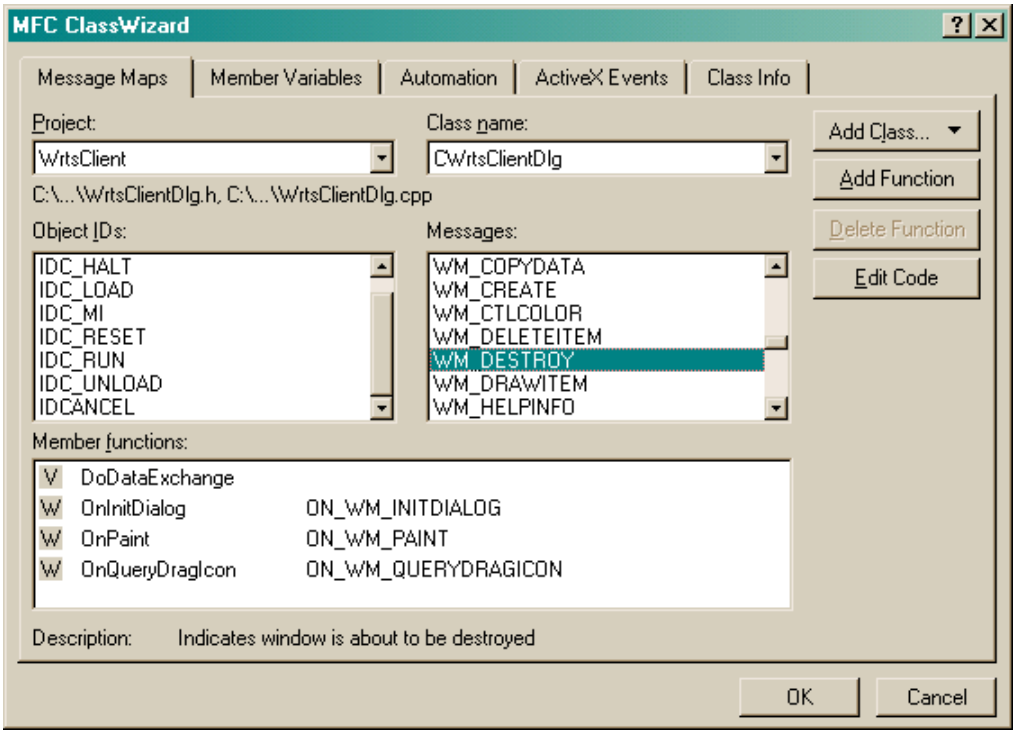
CoCreateInstance creates a TpsServer object.

Attach() is a method that will create an instance of the Wrts locally.

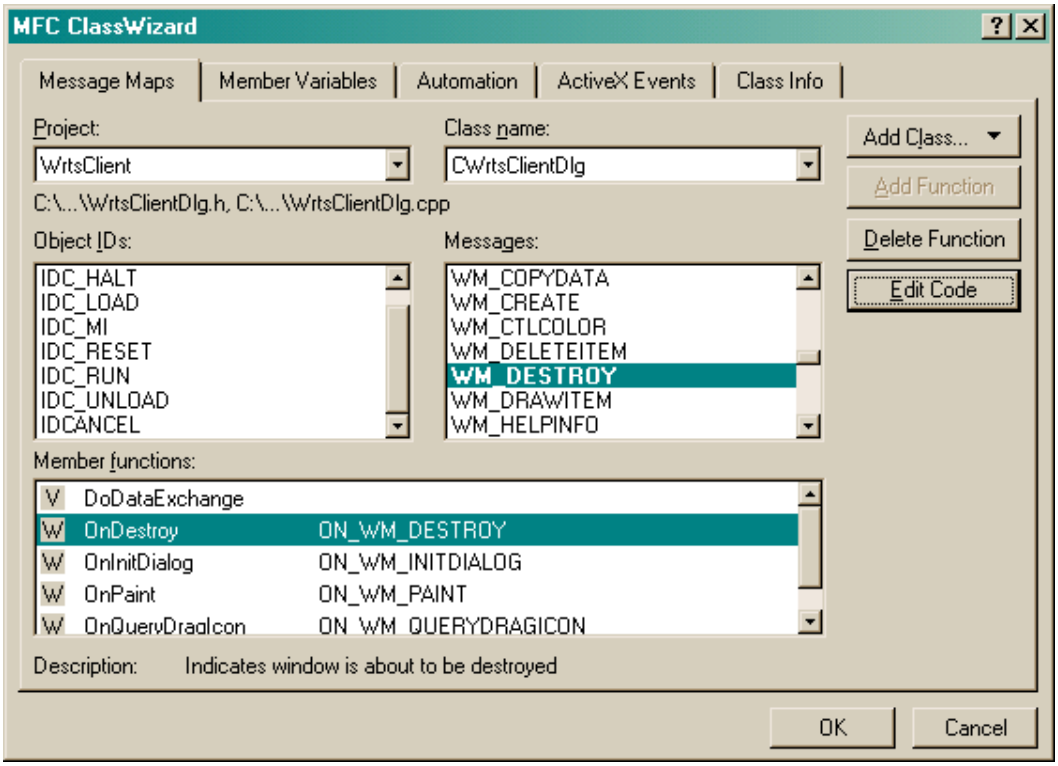
Note: If you wish to create an instance of the Wrts on a remote computer, you will want to pass on the IP address of that computer as an parameter for Attach. Here is an example of such call: `m_pTpsServerEx->Attach(CComVariant("192.168.100.164"));`

Please note that you will need to set the proper privileges on the computers in order to achieve this.

- Now we want to remember to release as the dialog gets destroyed. We will go to **View** and select **ClassWizard**.



- For the **CwrtsClientDlg** class name, select the **WM_DESTROY** and press **Add Function**. This will lead to the following window.



- Now we have a function name called **OnDestroy** where we can add the necessary code to Release the Wrts instance. In that function located in **WrtsClientDlg.cpp**, we want to add the following code:

```
void CWrtsClientDlg::OnDestroy()
{
	CDialog::OnDestroy();

	// TODO: Add your message handler code here
	m_pTpsServerEx.Release();
}
```

- Now we need to add CoInitialize and CoUninitialize. We could do this manually, but this will be added by the wizard (to add a Simple ATL/COM object) in the next step, so we would have to delete it before seeing it added by the wizard. So,

at this point, we will start with the next step which involves the TpsServer notification to the client.

Summary:

Here, you have seen code associated to 2 separate issues:

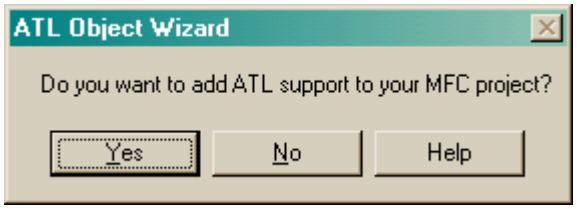
- The code associated to this specific GUI sample. This code will change as a function of your own GUI and may be entirely replaced.
- The code that you should have no matter what your GUI looks like that enables the client to load the TpsServer, to Attach and Release the Wrts instance. This code should be implement in your GUI no matter what. Please note that you can easily attach the Wrts when you initiate the GUI and you do not need a special button in your GUI to accomplish this task. You will want to import and include the proper files in order to gain access to the methods.

2 The sink object:

At this point, you will want to save your project and copy it. The wizard to insert an ATL/COM object may crash and corrupt your project. If this happens, you will want to delete the project and reuse a copy of the saved project and try it again.

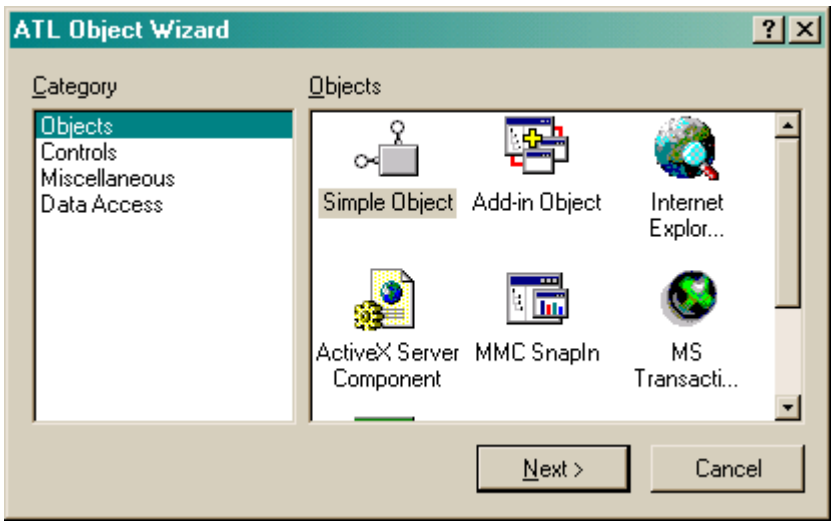
2.1 Creating the object:

- Close the project.
- Delete the files that are in your project subfolder with the extensions: **ncb**, **opt**, **plg**, **aps** and **clw** and the **Debug** folder.
- Open the project by double clicking on the file with the **dsw** extension.
- Go to **Insert** and select **New ATL Object**.

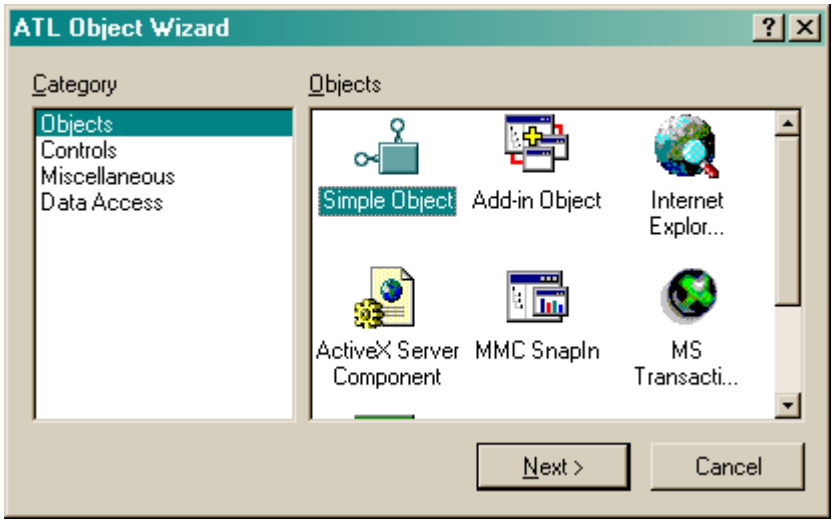


- Click on **YES**.
- You will be likely to see the following window:

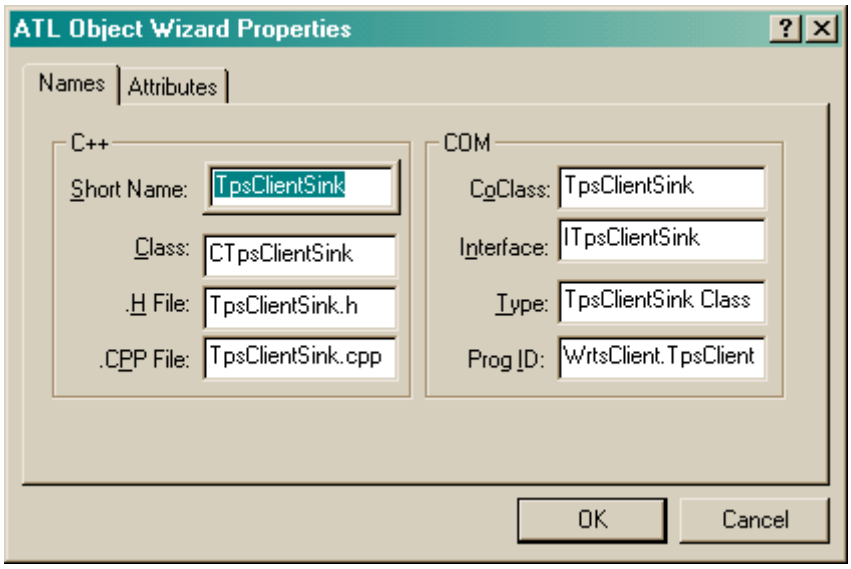
- If you are lucky, you will see the following window directly. You can then skip the steps until you see a picture of this window again:



- In the event that you see the error window, click on **OK**.
- Go into **Build** and select **Clean**.
- Go into **File** and select **Save Workspace**.
- Close the MSVC Studio.
- Delete the files with the extension **ncb, aps** and **opt** and the **Debug** folder.
- Double click on the file with the **dsw** extension.
- Go to **Insert** and select **New ATL Object**.
- Now you should see the following window:

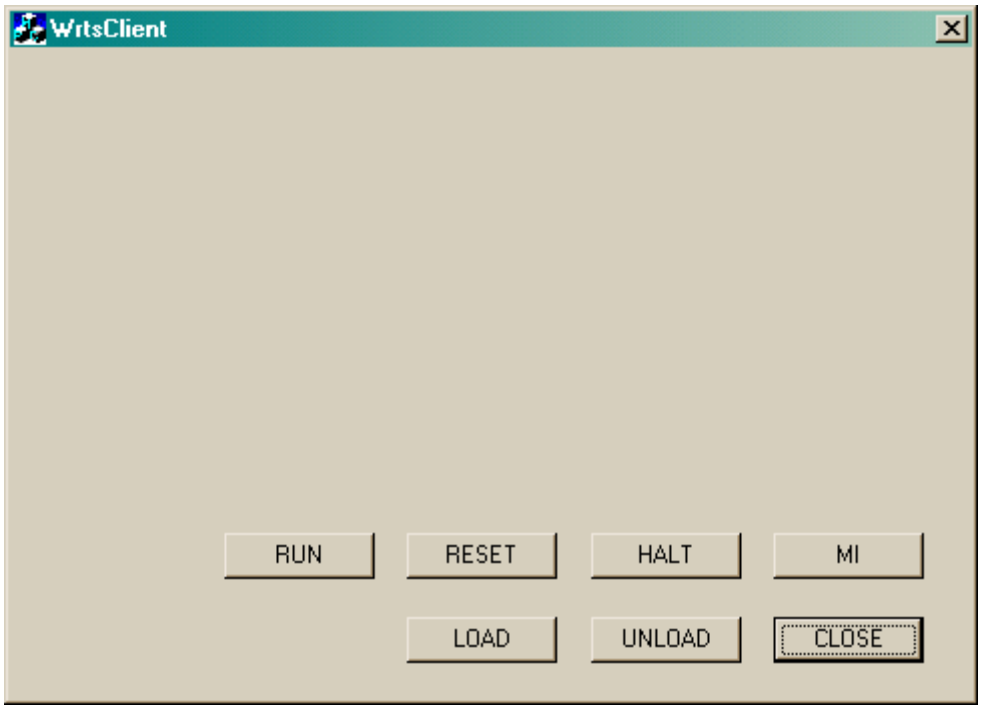


- Select **Simple Object** and then on **Next**.
- You need to add the name for the object. In the **Short Name** edit box, we will add **TpsClientSink**.



- In the **Attributes** tabs, we will want to select **SupportISupportErrorInfo** and then click on **OK**.
- At this point the project should build without errors.

Note: Once build, we can run the project. You will see the dialog window:



If you go into the task manager, you will see that the Wrts is running in the background.

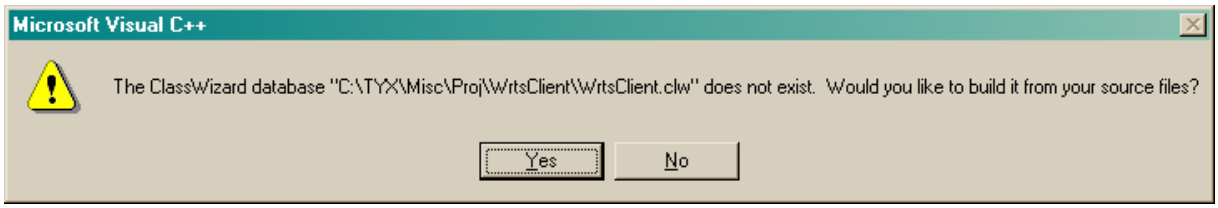
Once you have press **CLOSE**, you will see the Wrts being removed from the list in the Task Manager.

Summary:

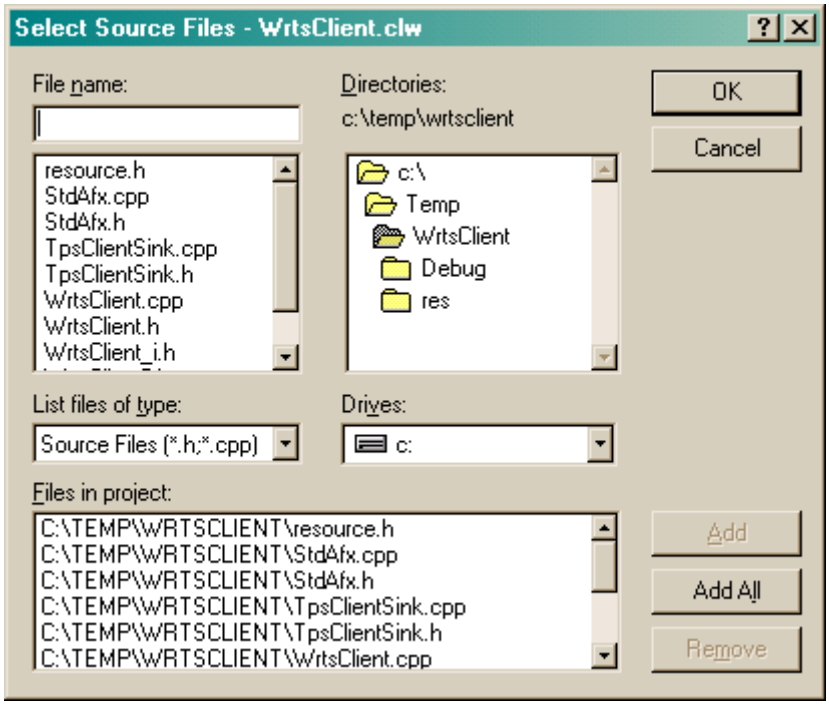
Adding **CoInitialize** and **CoUninitialize** can be done manually, but using the wizard to insert an ATL/Com object, you can make your life easier. Inserting this object will be needed later in this document anyway.

2.2 Adding functionality to the Buttons:

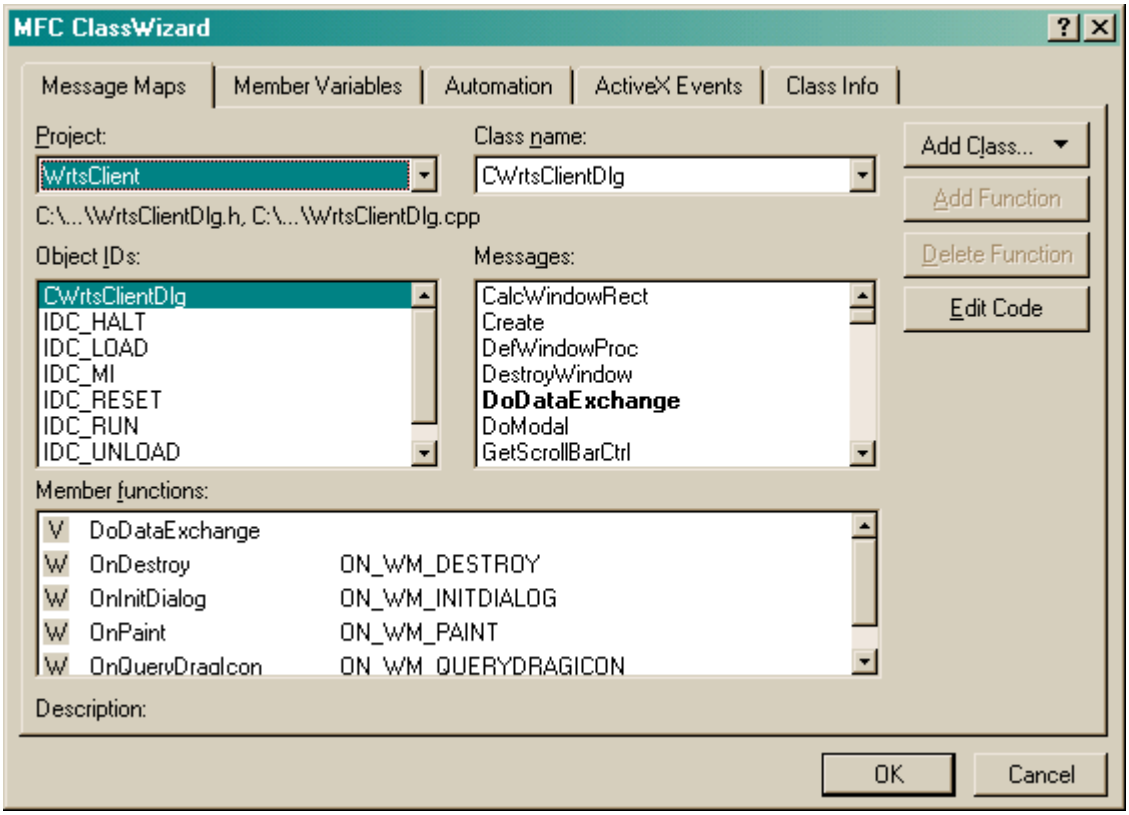
Go to **View** and select **Class Wizard**. You will see the following window



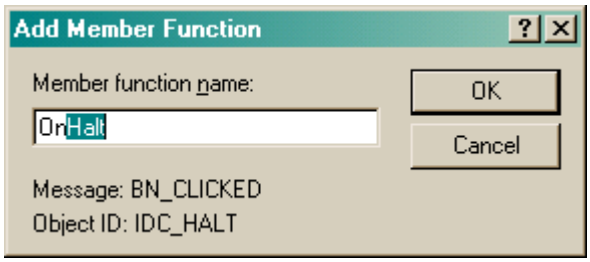
- Press on **Yes** and **OK** on the following window.



- You now have access to the **Class Wizard** window:



- Select **IDC_HALT** in the list of Object Ids and then **BN_CLICKED** and press **Add Function**.
- Accept the default name by pressing **OK**



- Do the same for **IDC_LOAD**, **IDC_MI**, **IDC_RESET**, **IDC_RUN**, **IDC_UNLOAD** and then click **OK** on the **MFC ClassWizard**.
- Those functions are added in the **WrtsClientDlg.cpp** file.
- We now want to add the code that is associated to the actions represented by the content of the caption.
- In the **OnHalt** method, you will want to add the following code:

```
void CWrtsClientDlg::OnHalt()
{
	// TODO: Add your control notification handler code here
	HRESULT hr = m_pTpsServerEx->Halt();
	if (FAILED(hr))
		handleError(hr);
}
```

- Do the same for the other methods associated to the other buttons:

```
void CWrtsClientDlg::OnLoad()
{
	// TODO: Add your control notification handler code here
	HRESULT hr = m_pTpsServerEx->Load(CComBSTR(""));
	if (FAILED(hr))
		handleError(hr);
}

void CWrtsClientDlg::OnMi()
{
	// TODO: Add your control notification handler code here
	HRESULT hr = m_pTpsServerEx->ManualIntervention();
	if (FAILED(hr))
		handleError(hr);
}
```

```
void CWrtsClientDlg::OnReset()
{
	// TODO: Add your control notification handler code here
	HRESULT hr = m_pTpsServerEx->Reset();
	if (FAILED(hr))
		handleError(hr);
}

void CWrtsClientDlg::OnRun()
{
	// TODO: Add your control notification handler code here
	HRESULT hr = m_pTpsServerEx->Run();
	if (FAILED(hr))
		handleError(hr);
}

void CWrtsClientDlg::OnUnload()
{
	// TODO: Add your control notification handler code here
	HRESULT hr = m_pTpsServerEx->Unload();
	if (FAILED(hr))
		handleError(hr);
}
```

You will want to add those methods where it makes sense for you in your own GUI.

- At this point, you can build the project. Once built, you can run it. Press **LOAD** and select a project on your computer. For the purpose of this test, you may want

to use a project that does nothing more than generate beeps since the Wrts will only run in the background and you won't be able to see any display at this point in time.

- Press Run and the TPS will be run.
- Press **CLOSE**.

➔ At this point, we have demonstrated how to connect directly with the TpsServer. The next step will allow for the client to catch all notifications from the TpsServer.

Notes:

- The buttons do not have access to the status of the Wrts so it is not possible at this point to disable the buttons that the user should not press (such as **RUN** before **LOAD**. Please note however, that if you press **RUN** before **LOAD**, an error will be displayed thanks to **handleError**).
- You can also include the name and path of the PAW project in the **Load** method. Here is an example:

```
m_pTpsServerEx->Load(CComBSTR("C:\\usr\\paws\\test.paw"));
```

Please note how each backslash is replaced by 2 backslashes inside the string.

Summary:

At this point in time, we are done accessing the methods that controls the Wrts directly. If you do not care to catch the notifications sent by the TpsServer, you are done.

The locations where the methods depend on the GUI and those methods can be moved.

3 Catching the notifications sent by the TpsServer:

This is the second step.

It has been initiated in step 2.1.

Now we will do the following:

- We will first worry about how the sink object is created by working on the sink class. This is something that you will need regardless of what your GUI looks like.

- Then we will add the property and the methods that allow the sink object to communicate with the dialog class. This step is also necessary regardless of what your GUI looks like. The sink object simply passes on the notifications from the TpsServer to your dialog class.

- We will finally handle the notifications in the dialog class. How those notifications are handled depend on your GUI.

3.1 The sink class and connecting the client to the TpsServer:

- In the TpsClientSink.h, add the following code:

```
class ATL_NO_VTABLE CTpsClientSink :
    public CComObjectRootEx<CComSingleThreadModel>,
    public CComCoClass<CTpsClientSink, &CLSID_TpsClientSink>,
    public ISupportErrorInfo,
    public IDispatchImpl<ITpsClientSink, &IID_ITpsClientSink, &LIBID_WrtsClientLib>,
    public IDispEventImpl<1,
                          CTpsClientSink,
                          &RTSAX::DIID__IRtsBaseEvents,
                          &RTSAX::LIBID_RTSAX, 1, 0>
{
```

Please note the “**,**” at the end of the line preceeding **IDisEventImpl**.

I. The first parameter is the ID used. We start with 1. This ID will be used as a parameter later in the document.

II. The second argument is the name of the CoClass. This depends on the name that you used. In our case, we will use **CtpsClientSink**.

III. The third is the connection point to the interface. This parameter should be fixed.

IV. The forth is associated to the library. This parameter should be fixed.

V. The last two are associated to the version of the TpsServer. These parameters should remain **1** and **0** until further notice.

- We will now add the information for the mapping in the same file below the above code:

```
BEGIN_COM_MAP(CTpsClientSink)
	COM_INTERFACE_ENTRY(ITpsClientSink)
	COM_INTERFACE_ENTRY(IDispatch)
	COM_INTERFACE_ENTRY(ISupportErrorInfo)
END_COM_MAP()

BEGIN_SINK_MAP(CTpsClientSink)
	SINK_ENTRY_EX(1, RTSAX::DIID__IRtsBaseEvents, 0x1, OnRtsTps)
	SINK_ENTRY_EX(1, RTSAX::DIID__IRtsBaseEvents, 0x5, OnRtsState)
	SINK_ENTRY_EX(1, RTSAX::DIID__IRtsBaseEvents, 0x9, OnRtsMiEnable)
	SINK_ENTRY_EX(1, RTSAX::DIID__IRtsBaseEvents, 0xa, OnRtsOutput)
	SINK_ENTRY_EX(1, RTSAX::DIID__IRtsBaseEvents, 0xb, OnRtsDisplay)
	SINK_ENTRY_EX(1, RTSAX::DIID__IRtsBaseEvents, 0xc, OnRtsInfo)
	SINK_ENTRY_EX(1, RTSAX::DIID__IRtsBaseEvents, 0xd, OnRtsWarning)
	SINK_ENTRY_EX(1, RTSAX::DIID__IRtsBaseEvents, 0xe, OnRtsError)
```

```
END_SINK_MAP()


// ISupportsErrorInfo
	STDMETHOD(InterfaceSupportsErrorInfo)(REFIID riid);
```

Please note that the first parameter is the same value as the first parameter for **IdispEventImpl** just above.

- We need to copy the notification in the same file just below the code above.

```
// ISupportsErrorInfo
	STDMETHOD(InterfaceSupportsErrorInfo)(REFIID riid);


	void __stdcall OnRtsState(long lState)
	{
		if (m_ClientHWnd)
			PostMessage((HWND)m_ClientHWnd, WM_USER + 1, eOnRtsState, lState);
	}
	void __stdcall OnRtsTps(BSTR pTps)
	{
		// extract the TPS name
		m_Tps = pTps;
		if (m_ClientHWnd)
			PostMessage((HWND)m_ClientHWnd, WM_USER + 1,
						eOnRtsTps, (long)(BSTR)m_Tps);
	}
	void __stdcall OnRtsMiEnable(VARIANT_BOOL varBool)
	{
		if (m_ClientHWnd)
```

```
                PostMessage((HWND)m_ClientHWnd, WM_USER + 1,
                                eOnRtsMiEnable, (long)(BOOL)(varBool !=
VARIANT_FALSE));
    }
    void __stdcall OnRtsOutput(BSTR pMsg)
    {
            if (m_ClientHWnd)
                    SendMessage((HWND)m_ClientHWnd, WM_USER + 1,
                                        eOnRtsOutput, (long)pMsg);
    }
    void __stdcall OnRtsDisplay(BSTR pMsg)
    {
            if (m_ClientHWnd)
                    SendMessage((HWND)m_ClientHWnd, WM_USER + 1,
                                        eOnRtsDisplay, (long)pMsg);
    }
    void __stdcall OnRtsInfo(BSTR pMsg)
    {
            if (m_ClientHWnd)
                    SendMessage((HWND)m_ClientHWnd, WM_USER + 1,
                                        eOnRtsInfo, (long)pMsg);
    }
    void __stdcall OnRtsWarning(BSTR pMsg)
    {
            if (m_ClientHWnd)
                    SendMessage((HWND)m_ClientHWnd, WM_USER + 1,
                                        eOnRtsWarning, (long)pMsg);
    }
    void __stdcall OnRtsError(BSTR pMsg)
```

```
    {
            if (m_ClientHWnd)
                    SendMessage((HWND)m_ClientHWnd, WM_USER + 1,
                                            eOnRtsError, (long)pMsg);
    }


protected:
    HWND            m_ClientHWnd;
    CComBSTR        m_Tps;


// ITpsClientSink
public:
};
```

- Now we want to add the enum for the third parameter in the **SendMessage** above. We will do this in **StdAfx.h**:

```
    extern CDemoSekasTpsClientModule _Module;
    #include <atlcom.h>

    enum EnumOnRtsEvent
    {
            eOnRtsTps = 0,
            eOnRtsState,
            eOnRtsMiEnable,
            eOnRtsOutput,
            eOnRtsDisplay,
            eOnRtsInfo,
```

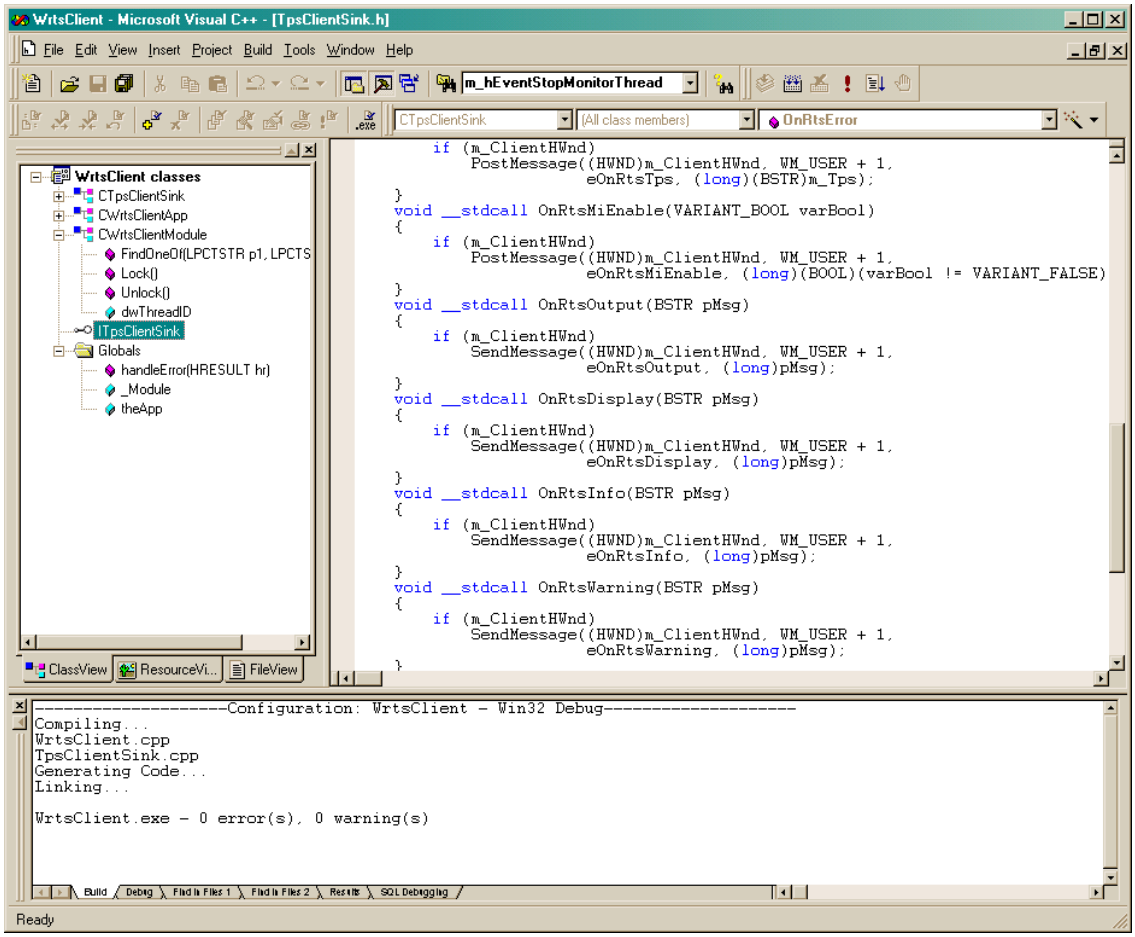
```
        eOnRtsWarning,

        eOnRtsError,

};



//{{AFX_INSERT_LOCATION}}

// Microsoft Visual C++ will insert additional declarations immediately

// before the previous line.



#endif //
!defined(AFX_STDAFX_H__203EBD2C_7A3A_42FE_B018_12F5404E61EE__INCLUDED_)
```

- At this point, the project should build without errors.

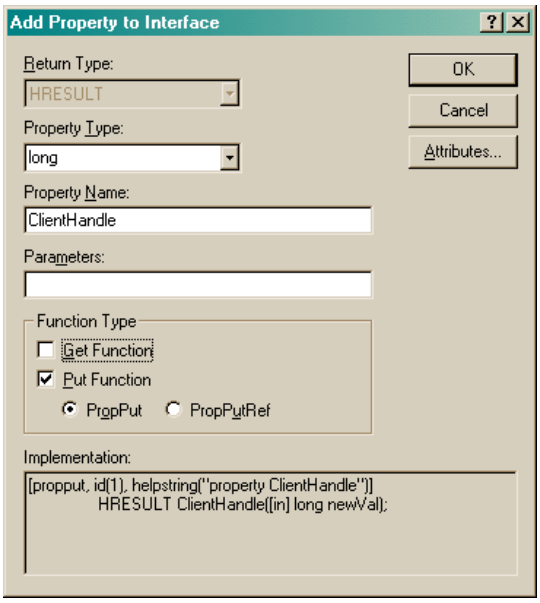
- We still need to add the property and the methods. The property will allow to set the member variable **m_ClientHWnd** of the sink object to the dialog handle. The member variable **m_ClientHWnd** will then allow the sink object to access the buttons and the edit windows that we will implement later. The methods will allow the sink object to establish and break the connection with the TpsServer. We will see this in the explanation below.

3.2 Adding the Property and the Methods:

- In the project window, select the **Class View** tab and right click on the **ITpsClientSink** Interface for the **CWrtsClientModule** and select **Add Property**.



- In that **Add Property to Interface** window, select **long** for **Property Type**, **ClientHandle** for **Property Name**. Unselect **GetFunction**.



- Press on **OK**.
- In the **TpsClientSink.cpp**, add the line of code that will allow to set the member variable **m_ClientHWnd** of the sink object to equal the handle for the dialog.

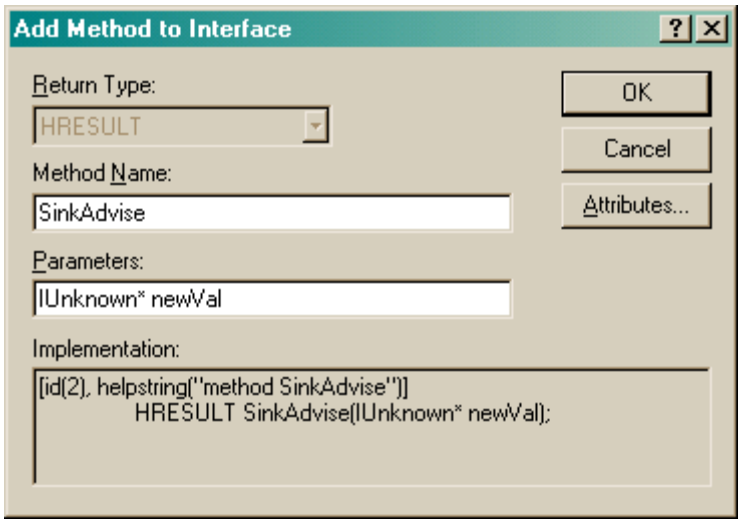
```
STDMETHODIMP CTpsClientSink::put_ClientHandle(long newVal)

{

      AFX_MANAGE_STATE(AfxGetStaticModuleState())


      // TODO: Add your implementation code here

      m_ClientHWnd = (HWND)newVal;

      return S_OK;

}
```

- At this point, we will also initialize that member variable in the constructor located in **TpsClientSink.h** to be NULL.

```
public:
```

```
CTpsClientSink()

{

	m_ClientHWnd = NULL;

}
```

- For the first method, right click once again on the Interface and select **Add Method**. Enter **SinkAdvise** for the **Method Name**. Add **Iunknown* newVal** for **Parameters**.

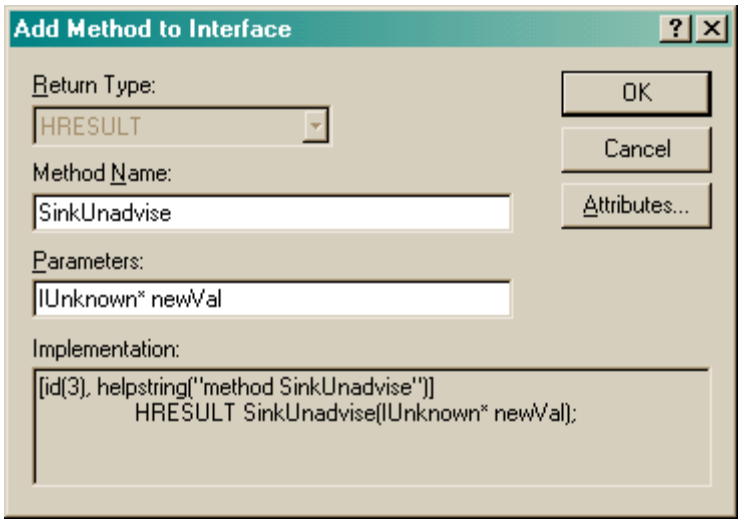


- In **TpsClientSink.cpp**, modify the return value from **return S_OK;** to the following line below

```
STDMETHODIMP CTpsClientSink::SinkAdvise(IUnkown *newVal)

{

	AFX_MANAGE_STATE(AfxGetStaticModuleState())


	// TODO: Add your implementation code here
```

```
        return DispEventAdvise(newVal);
}
```

- For the second method, right click once again on the Interface and select **Add Method**. Enter **SinkUnadvise** for the **Method Name**. Add **IUnknown* newVal** for **Parameters**.



- In **TpsClientSink.cpp**, modify the return value from **return S_OK;** to the following line below

```
STDMETHODIMP CTpsClientSink::SinkUnadvise(IUnknown *newVal)
{
        AFX_MANAGE_STATE(AfxGetStaticModuleState())

        // TODO: Add your implementation code here
        return DispEventUnadvise(newVal);
}
```

- At this point, we are done with sink class.
- We are going to declare a smart pointer in the dialog class that will point to the sink object. In the **WrtsClientDlg.h**, we will add

```
        //}}AFX_MSG

        DECLARE_MESSAGE_MAP()


        CComPtr<RTSAX::ITpsServerEx> m_pTpsServerEx;

        CComPtr<ITpsClientSink> m_pSink;

    };
```

- We are first going to add some code in **OnInitdialog** located in **WrtsClientDlg.cpp**, that you will need to add regardless of the aspect of your GUI:

```
BOOL CWrtsClientDlg::OnInitDialog()

{

    CDialog::OnInitDialog();


    // 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

    // create tps server

    HRESULT hr = m_pTpsServerEx.CoCreateInstance(RTSAX::CLSID_TpsServer);
```

```
    if (FAILED(hr)) { handleError(hr); return FALSE; }

    // put tps server in asynchronous mode
    hr = m_pTpsServerEx->put_Synchronous(VARIANT_FALSE);
    if (FAILED(hr)) { handleError(hr); return FALSE; }

    // create the sink object
    hr = m_pSink.CoCreateInstance(CLSID_TpsClientSink);
    if (FAILED(hr)) { handleError(hr); return FALSE; }

    // pass the window handle to the sink object
    hr = m_pSink->put_ClientHandle((long)m_hWnd);
    if (FAILED(hr)) { handleError(hr); return FALSE; }

    // set up the connection point
    hr = m_pSink->SinkAdvise(m_pTpsServerEx);
    if (FAILED(hr)) { handleError(hr); return FALSE; }

    // attach tpsserver to wrts
    hr = m_pTpsServerEx->Attach();
    if (FAILED(hr)) { handleError(hr); return FALSE; }

    return TRUE;  // return TRUE  unless you set the focus to a control
}
```

The added code has the following purpose:

I. **put_Synchronous** puts the tps in asynchronous mode.

II. **CoCreateInstance** creates the sink object.

III. **put_ClientHandle** passes the dialog handle to the sink object.

IV. **SinkAdvise** connects the sink object to the TpsServer connection point..

- Similarly as we added the **UnAdvise** in the **OnInitdialog**, we now need to add the code for **Unadvise** in **OnDestroy**. We want to do this before we release the TpsServer. We also want to release the sink object before releasing the TpsServer object.

```
void CWrtsClientDlg::OnDestroy()
{
	CDialog::OnDestroy();

	// TODO: Add your message handler code here
	if (m_pSink && m_pTpsServerEx)
		m_pSink->SinkUnadvise(m_pTpsServerEx);
	m_pSink.Release();
	m_pTpsServerEx.Release();
}
```

- The project should build without errors. If you are having problems linking, close the MSVC studio saving the project, and reload project. Doing a rebuild all at this point should be successful.

3.3 Handling the notifications:

- Now we can add the code in the dialog. The location of this code is specific to this sample and may be modified in order to match the functionality of your own GUI.
- We will address the three following actions:

I. We will add a few Edit boxes in the dialog window

II. Prepare the reception of notifications.

III. We will add the code.

IV. We will add the mapping.

3.3.1 Adding Edit boxes in the dialog

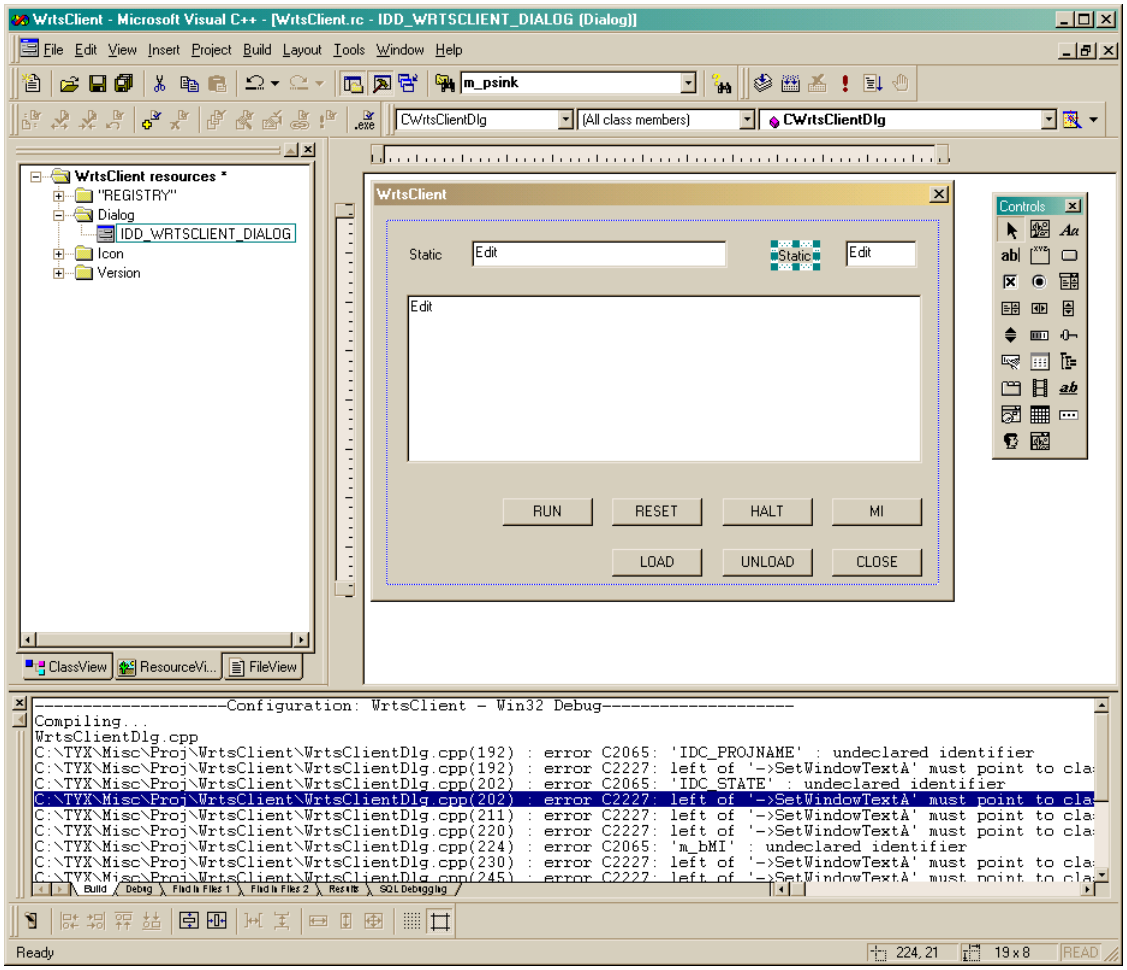
The purpose of doing this is to add three read-only edit boxes.

I. One to display the messages intended for the Wrts display. We will not parse the string in order to handle the invisible characters intended to handle the ASCII codes for coloring the font, background… It will nevertheless still display the text that is intended to be displayed.

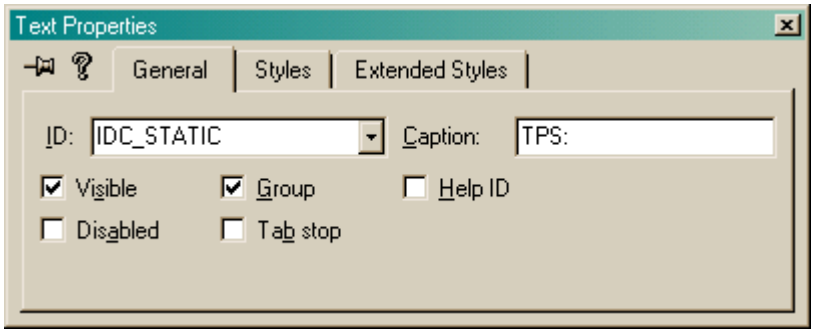
II. The second Edit will display the project name.

III. The last one will indicate the state of the Wrts.

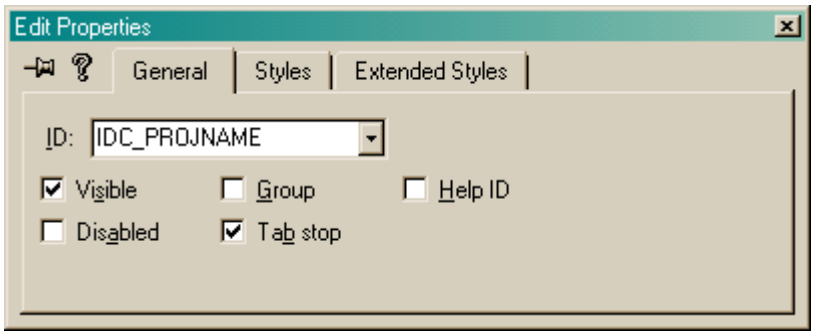
Add three edit boxes and two static boxes as defined below:



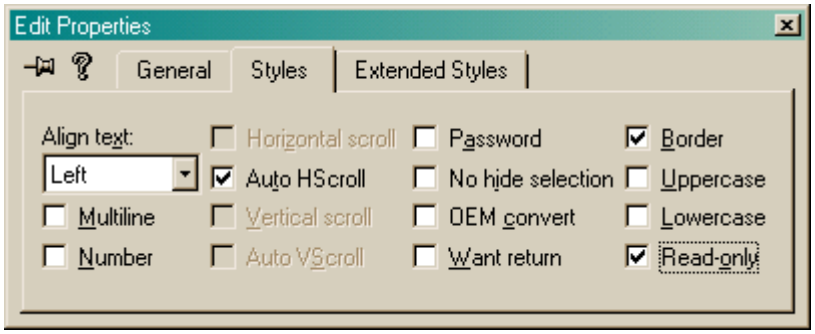
- Select the top left static box, and right click on it.
- Change the **Caption** to **TPS:** as shown below.



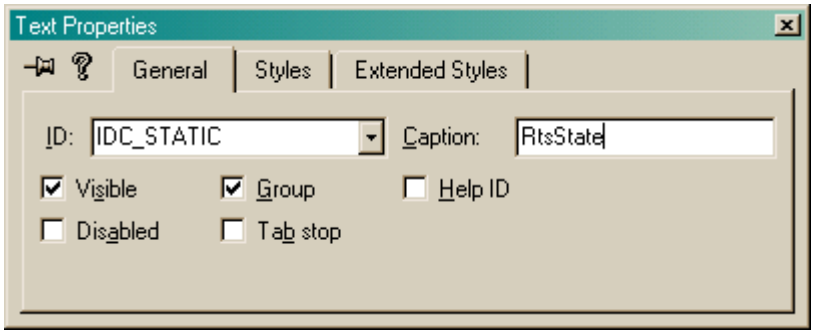
- Select the Edit box next to it and right click on it. Change the **ID** to **IDC_PROJNAME**



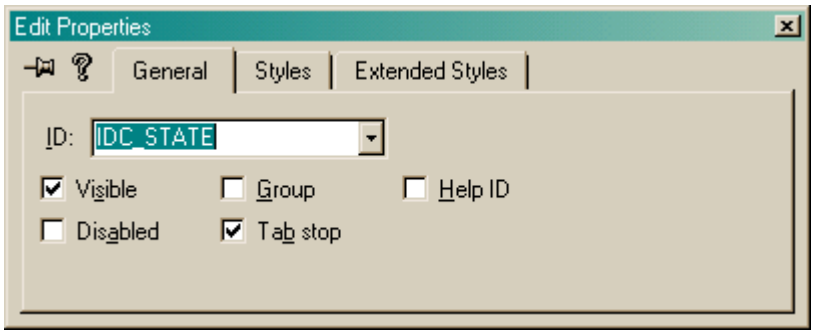
- In the **Styles** tab, select **Read only**.



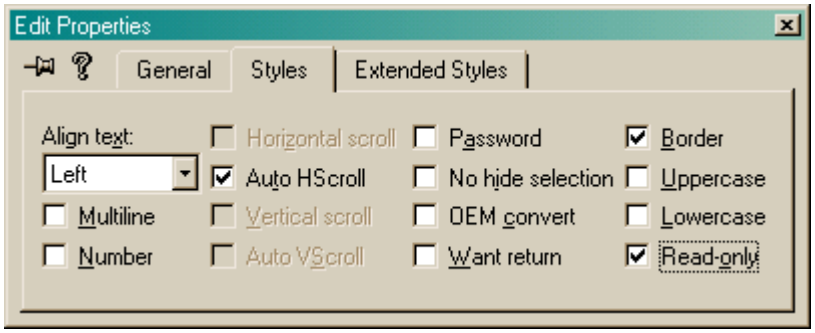
- In the second static box, change the **Caption** to **RtsState**



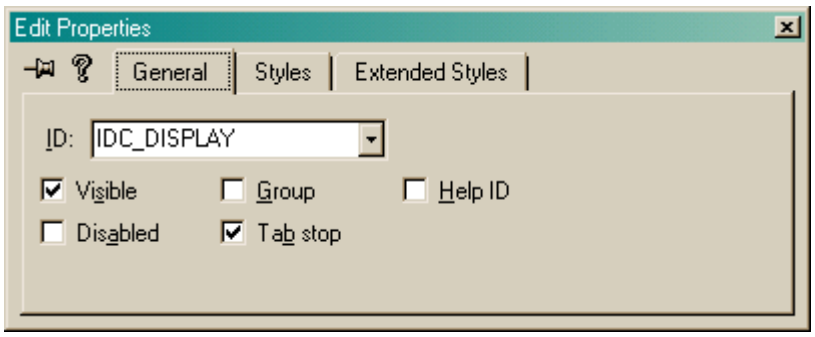
- In the Edit box next to it, change the **ID** to **IDC_STATE**



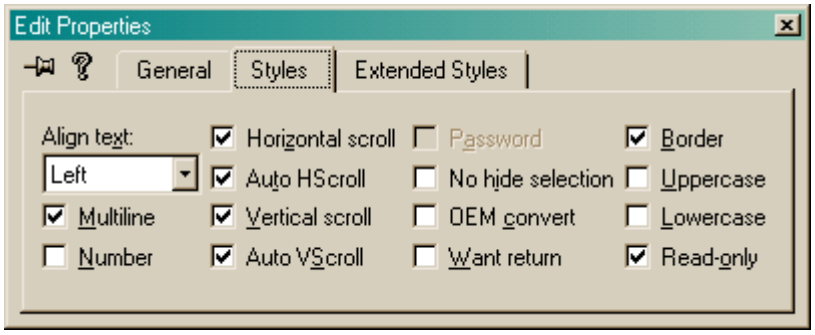
- And change select **Read only**.



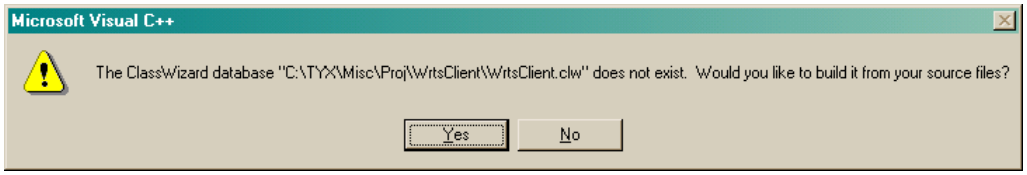
- In the last Edit box, the largest, where the output text will be displayed, change the **ID** to **IDC_DISPLAY**



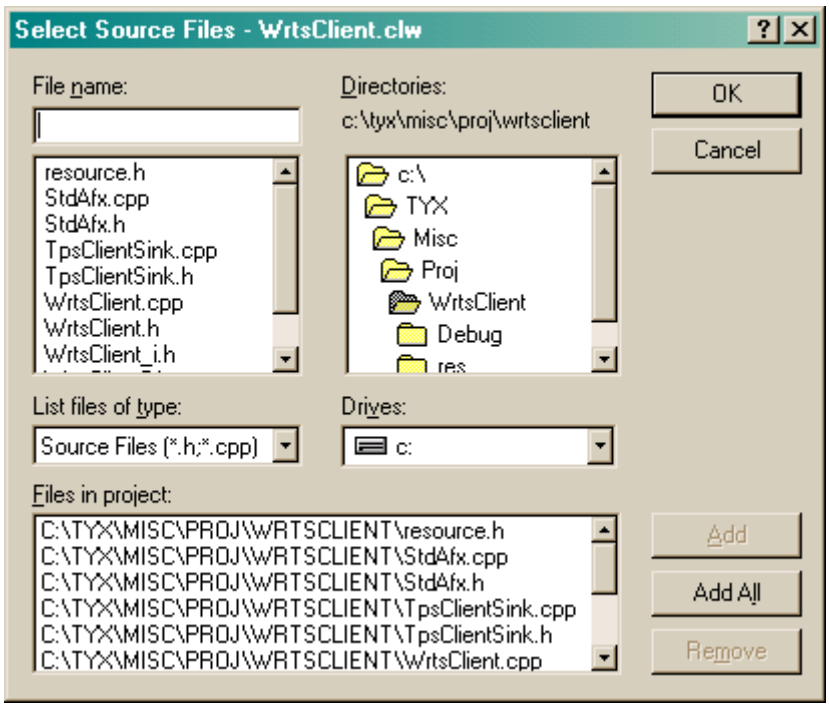
- Select **Multine** and then **Horizontal scroll**, **Vertical scroll**, **Auto VScroll** and **Read Only**.



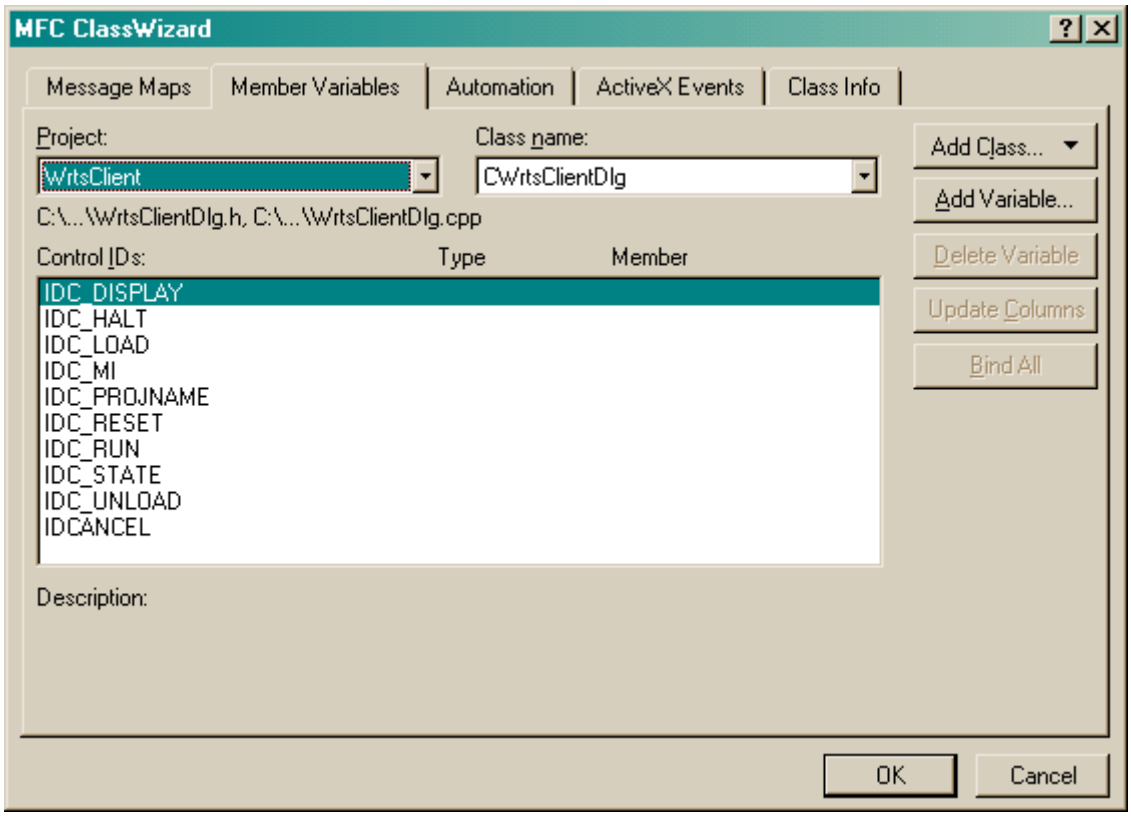
- Now we want to add a variable associated to the Edit box that is used as display. Go to **View** and select **Class Wizard**. If you don't see the **MFC ClassWizard** window directly, you will see the following window:



- Click on **Yes** and **OK** on the following window

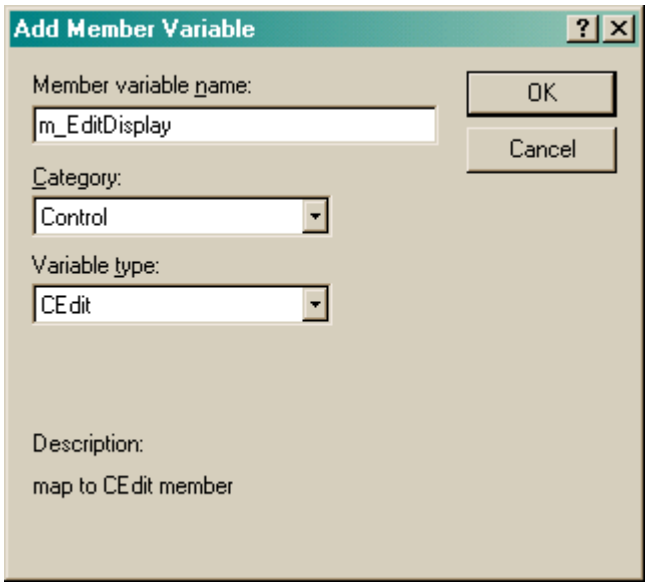


- Select the **Member Variables** tab and select **IDC_DISPLAY**.



- Press on **Add Variable**.
- In the **Add Member Variable** window, Change the **Category** to **Control**. The **Variable type** should change to **CEdit**.

  Change the **Member variable name** to **m_EditDisplay**.



- Click on **OK** and **OK** on the **MFC ClassWizard** window.

3.3.2 Preparing the reception of notifications

- In **WrtsClientDlg.h**, we will add a method that will handle the incoming messages from the sink object.

```
// Generated message map functions
```

```
    //{{AFX_MSG(CWrtsClientDlg)
    virtual BOOL OnInitDialog();
    afx_msg void OnPaint();
    afx_msg HCURSOR OnQueryDragIcon();
    afx_msg void OnDestroy();
    afx_msg void OnHalt();
    afx_msg void OnLoad();
    afx_msg void OnMi();
    afx_msg void OnReset();
    afx_msg void OnRun();
    afx_msg void OnUnload();
    //}}AFX_MSG
    LRESULT OnComMessage(WPARAM wParam, LPARAM lParam);
    DECLARE_MESSAGE_MAP()


    CComPtr<RTSAX::ITpsServerEx> m_pTpsServerEx;
    CComPtr<ITpsClientSink> m_pSink;
};
```

3.3.3 Adding the code

- We will add a small variable to keep track of the **MI** button. It will enable to keep track of whether it needs to be pressed, regardless of whether we press on **HALT** and then **RUN**. This variable and associated functionality makes sense in the context of this sample, but not necessarily in the context of another GUI. We will call this member variable **m_bMI**. It will be declared in the WrtsClientDlg.h class

```
LRESULT OnComMessage(WPARAM wParam, LPARAM lParam);
```

```
	DECLARE_MESSAGE_MAP()


	CComPtr<RTSAX::ITpsServerEx> m_pTpsServerEx;
	CComPtr<ITpsClientSink> m_pSink;


	BOOL m_bMI;
};
```

- We will initialize this variable in the **OnInitDialog** function to **FALSE**

```
	// attach tpsserver to wrts
	hr = m_pTpsServerEx->Attach();
	if (FAILED(hr)) { handleError(hr); return FALSE; }


	m_bMI = FALSE;
	return TRUE;  // return TRUE  unless you set the focus to a
control
}
```

- We will also set it to **FALSE** in **OnMi**

```
void CWrtsClientDlg::OnMi()
{
	// TODO: Add your control notification handler code here
	m_bMI = FALSE;
	HRESULT hr = m_pTpsServerEx->ManualIntervention();
```

```
        if (FAILED(hr))
            handleError(hr);
    }
```

- It will also be handled **OnComMessage** function below.
- In the we will add **#include <limits.h>** at the top of **WrtsClientDlg.cpp** in order to handle some of the code associated to the **Display** edit box in the **OnComMessage** function.

```
// WrtsClientDlg.cpp : implementation file
//

#include "stdafx.h"
#include "WrtsClient.h"
#include "WrtsClientDlg.h"
#include <limits.h>

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif
```

- At the end of **WrtsClientDlg.cpp**, we will add the code associated to the definition above:

```
LRESULT CWrtsClientDlg::OnComMessage(WPARAM wParam, LPARAM lParam)
```

```
{
        USES_CONVERSION;
        TRY
        {
                switch((EnumOnRtsEvent)wParam)
                {
                        case eOnRtsTps:
                                GetDlgItem(IDC_PROJNAME)-
>SetWindowText(OLE2T((BSTR)lParam));
                                break;
                        case eOnRtsMiEnable:
                                GetDlgItem(IDC_MI)->EnableWindow(lParam);
                                break;
                        case eOnRtsState:
                                // control the state of buttons
                                switch(lParam)
                                {
                                case 0:// RTS_STATE_CLOSED
                                        GetDlgItem(IDC_STATE)->SetWindowText(_T("CLOSED"));
                                        GetDlgItem(IDC_RUN)->EnableWindow(FALSE);
                                        GetDlgItem(IDC_RESET)->EnableWindow(FALSE);
                                        GetDlgItem(IDC_HALT)->EnableWindow(FALSE);
                                        GetDlgItem(IDC_MI)->EnableWindow(FALSE);
                                        GetDlgItem(IDC_LOAD)->EnableWindow(TRUE);
                                        GetDlgItem(IDC_UNLOAD)->EnableWindow(FALSE);
                                        break;
                                case 1:// RTS_STATE_READY
                                        GetDlgItem(IDC_STATE)->SetWindowText(_T("READY"));
                                        GetDlgItem(IDC_RUN)->EnableWindow(TRUE);
```

```
        GetDlgItem(IDC_RESET)->EnableWindow(TRUE);
        GetDlgItem(IDC_HALT)->EnableWindow(FALSE);
        GetDlgItem(IDC_MI)->EnableWindow(FALSE);
        GetDlgItem(IDC_LOAD)->EnableWindow(FALSE);
        GetDlgItem(IDC_UNLOAD)->EnableWindow(TRUE);
        break;
    case 2:// RTS_STATE_RUNNING
        GetDlgItem(IDC_STATE)->SetWindowText(_T("RUNNING"));
        GetDlgItem(IDC_RUN)->EnableWindow(FALSE);
        GetDlgItem(IDC_RESET)->EnableWindow(TRUE);
        GetDlgItem(IDC_HALT)->EnableWindow(TRUE);
        GetDlgItem(IDC_MI)->EnableWindow(m_bMI);
        GetDlgItem(IDC_LOAD)->EnableWindow(FALSE);
        GetDlgItem(IDC_UNLOAD)->EnableWindow(TRUE);
        break;
    case 3:// RTS_STATE_HALTED
        {
        GetDlgItem(IDC_STATE)->SetWindowText(_T("HALTED"));
            GetDlgItem(IDC_RUN)->EnableWindow(TRUE);
            GetDlgItem(IDC_RESET)->EnableWindow(TRUE);
            GetDlgItem(IDC_HALT)->EnableWindow(FALSE);

            // save the MI state
            CWnd* pMI = GetDlgItem(IDC_MI);
            m_bMI = pMI->IsWindowEnabled();
            pMI->EnableWindow(FALSE);

            GetDlgItem(IDC_LOAD)->EnableWindow(FALSE);
```

```
                                        GetDlgItem(IDC_UNLOAD)->EnableWindow(TRUE);
                                }
                                break;
                        case 4:// RTS_STATE_FINISH
                                GetDlgItem(IDC_STATE)->SetWindowText(_T("FINISH"));
                                GetDlgItem(IDC_RUN)->EnableWindow(TRUE);
                                GetDlgItem(IDC_RESET)->EnableWindow(TRUE);
                                GetDlgItem(IDC_HALT)->EnableWindow(FALSE);
                                GetDlgItem(IDC_MI)->EnableWindow(FALSE);
                                GetDlgItem(IDC_LOAD)->EnableWindow(FALSE);
                                GetDlgItem(IDC_UNLOAD)->EnableWindow(TRUE);
                                break;
                        }
                        break;
                case eOnRtsOutput:
                case eOnRtsDisplay:
                case eOnRtsInfo:
                case eOnRtsWarning:
                case eOnRtsError:
                        m_EditDisplay.SetSel(LONG_MAX, LONG_MAX);
                        m_EditDisplay.ReplaceSel(OLE2T((BSTR)lParam));
                        break;
        }
        return 0;
}
CATCH(COleDispatchException, e)
{
        if (e->m_strDescription.IsEmpty())
```

```
                handleError(e->m_scError);
        else
        {
                CString errMsg;
                errMsg.Format("COM Error from %s: %s",
                        e->m_strSource,
                        e->m_strDescription);
                // Display the com error:
                ::MessageBox(NULL, errMsg, _T("Error"), MB_OK);
        }
        return 0;
    }
    AND_CATCH_ALL(e)
    {
        handleError(E_FAIL);
        return 0;
    }
    END_CATCH_ALL
}
```

3.3.4 Adding the map

- In the **WrtsClientDlg.cpp**, we will add the message map for the **OnComMessage**.

```
BEGIN_MESSAGE_MAP(CWrtsClientDlg, CDialog)
        //{{AFX_MSG_MAP(CWrtsClientDlg)
        ON_WM_PAINT()
```

```
        ON_WM_QUERYDRAGICON()
    ON_WM_DESTROY()
        ON_BN_CLICKED(IDC_HALT, OnHalt)
        ON_BN_CLICKED(IDC_LOAD, OnLoad)
        ON_BN_CLICKED(IDC_MI, OnMi)
        ON_BN_CLICKED(IDC_RESET, OnReset)
        ON_BN_CLICKED(IDC_RUN, OnRun)
        ON_BN_CLICKED(IDC_UNLOAD, OnUnload)
        //}}AFX_MSG_MAP
        ON_MESSAGE(WM_USER + 1, OnComMessage)
END_MESSAGE_MAP()
```

At this point, the project should build without errors and you should be able to run the client.

This should conclude the making of the Wrts client.