TYX CORPORATION

Productivity Enhancement Systems

Reference	TYX_0051_7
Revision	1.1
Document	PawsIODLLHowTo.doc
Date	April 7, 2003

Binary I\O dll resource with MFC tutorial

This document will help with the making of a Binary input-output DLL ATL/COM interface with MFC.

Studio version used: 1.20.0

Requirements: Studio 1.10.x or above. Backward and forward compatibility between the COM build with one version of the Studio library and other Studio versions is not guarantied.

Introduction: First we will address the issue of what to do with the dll that we wish to link to the Atlas with a Binary IO and MFC. Other documents will address the issue of using the support of Text IO and Binary IO resources with an exe without the usage of GUI within the COM environment.
Then we will see the Atlas and what it takes with a 416 environment to make use of it.

Dll versus Exe:

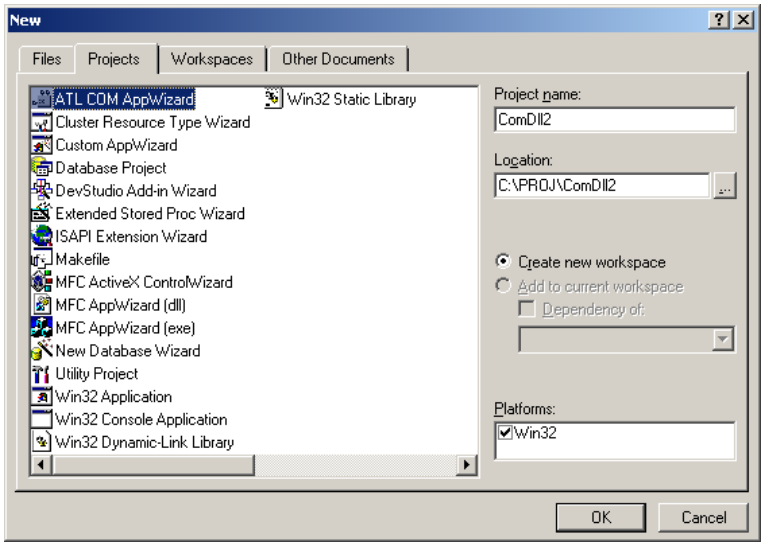
1. **Advantage of Dlls versus Exe:**
   - **The Dll uses less resources than the Exe.**
   - **The Dll can easily remain on top of the Wrts.**
   - **The accelerators are directly passed on to the Wrts without additional code. When the Exe has the focus, it will need some code in order to pass on the accelerator destined for the Wrts.**
   - **The message loop for painting the GUI is taken care of by the Wrts. For the Exe, the code needs to be placed into the Exe which may be done by the wizard to some extent.**
   - **The Dll will be slightly faster than the Exe. This will however only make a difference for repeated transfer of a large amount of data.**
2. **Advantage of the Exe versus Dll:**
   - **The Exe is more isolated than the Dll and if it crashes, it will not affect the Wrts. This may be an issue if the Exe links to code that is not stable.**
   - **The Exe can be moved in front or behind the Wrts at will.**
   - **The Exe can be run remotely.**

1 1 When generating a dll with MSVC 6++

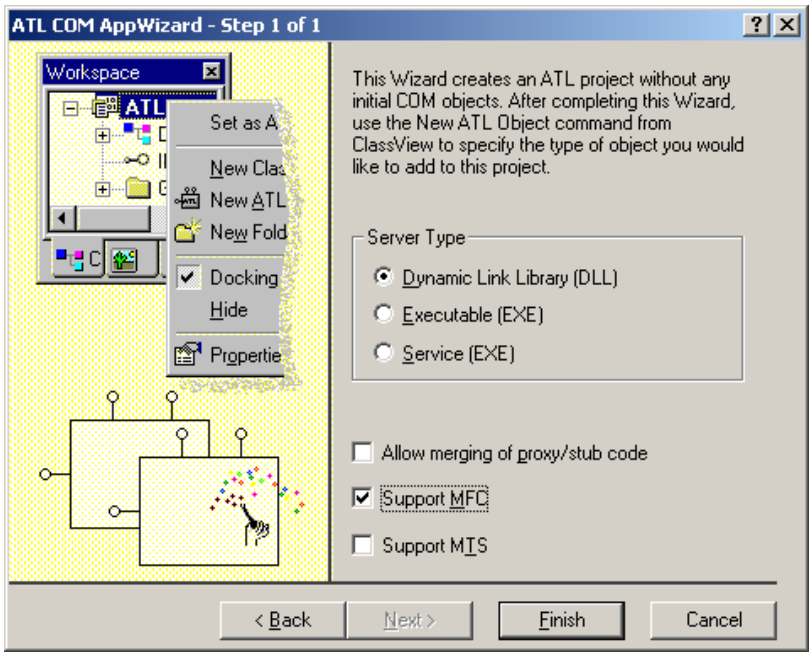
Combining ATL and MFC is not totally straightforward with the wizards. It is easier to deal with fixing the ATL problems on top of a MFC project than the other way around so we are going to do just that:
First start an MFC project and then try to add the ATL to it.

1.1 1.1 Starting a simple ATL resource with MFC support, dialog based with one toggle button:

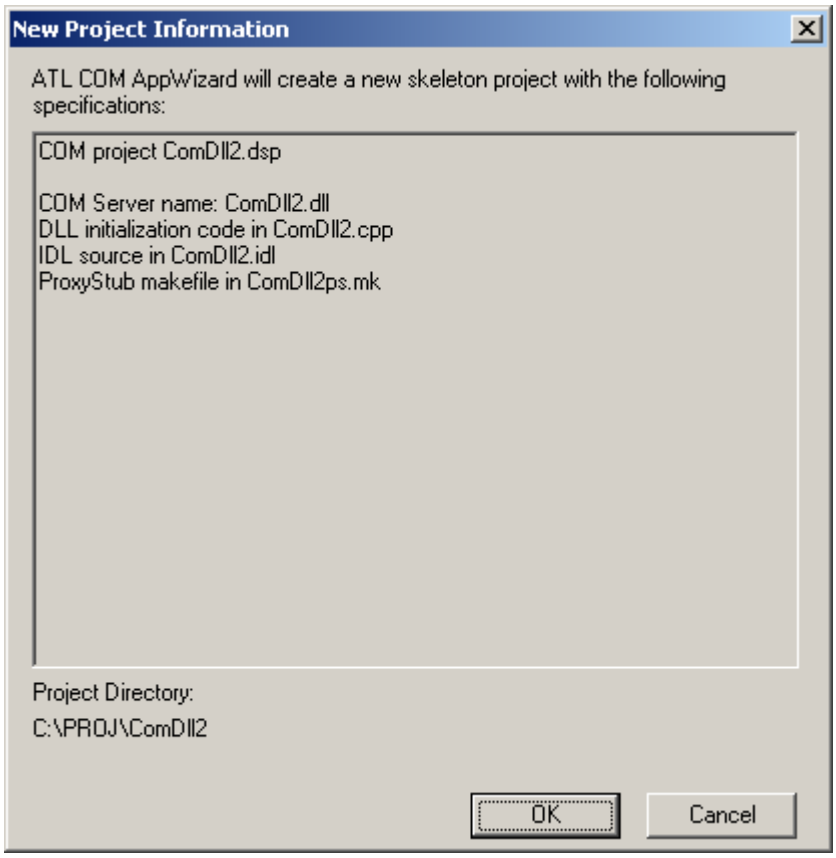
- From **File/New**…, select an **ATL COM Application** and select a project name **ComDll2** as seen below:



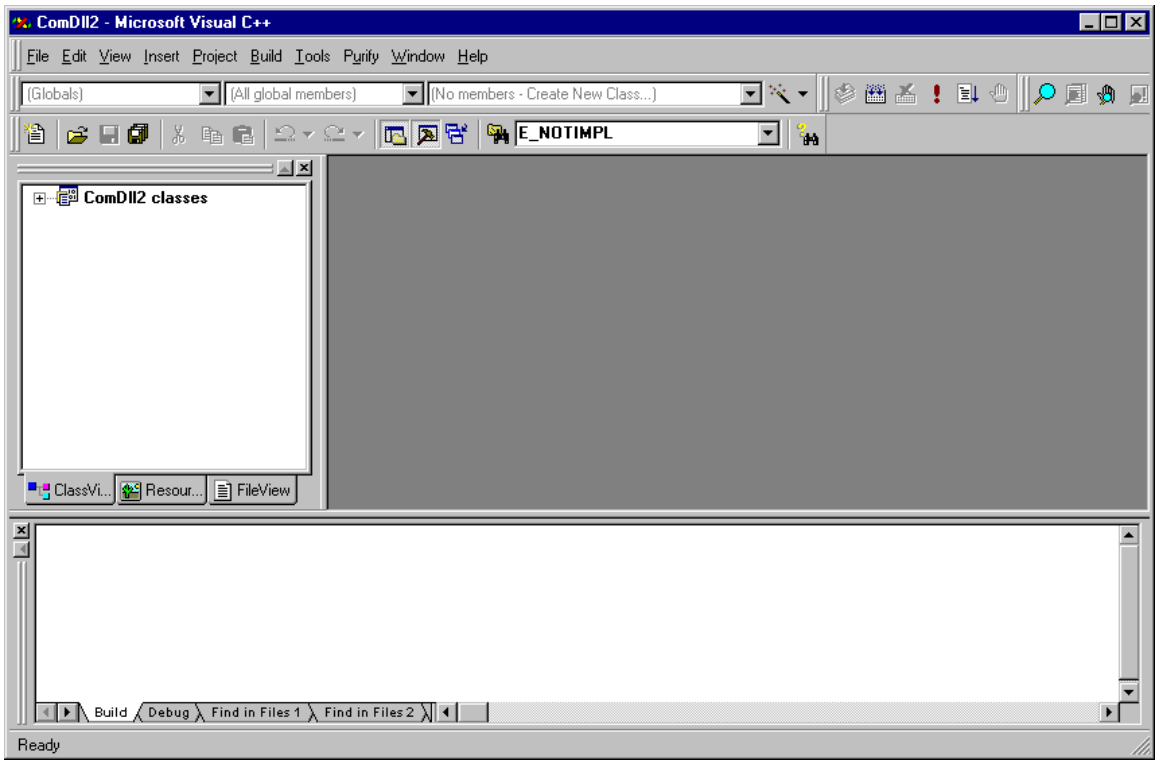
- Click on **OK**
- Select **Dynamic Link Library (DLL)** and **Support MFC**.



- Click on **Finish**.
- Click **OK** on the following window:

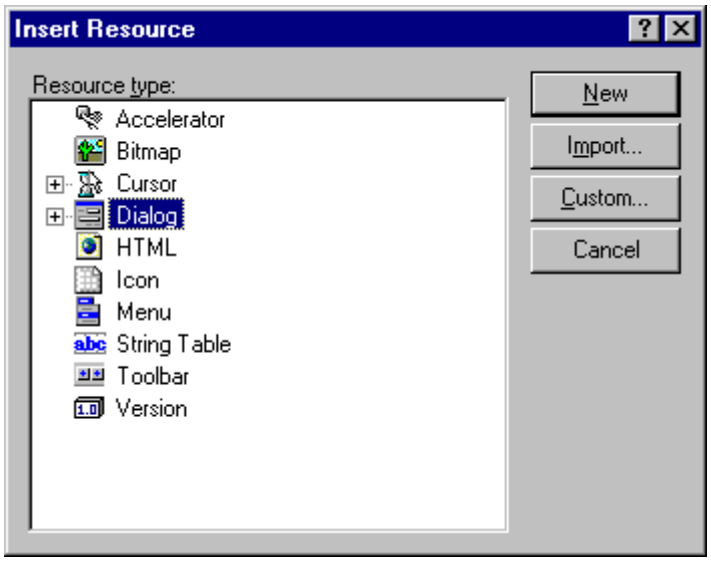


- If we look at the image below, we can see what the project should look like.

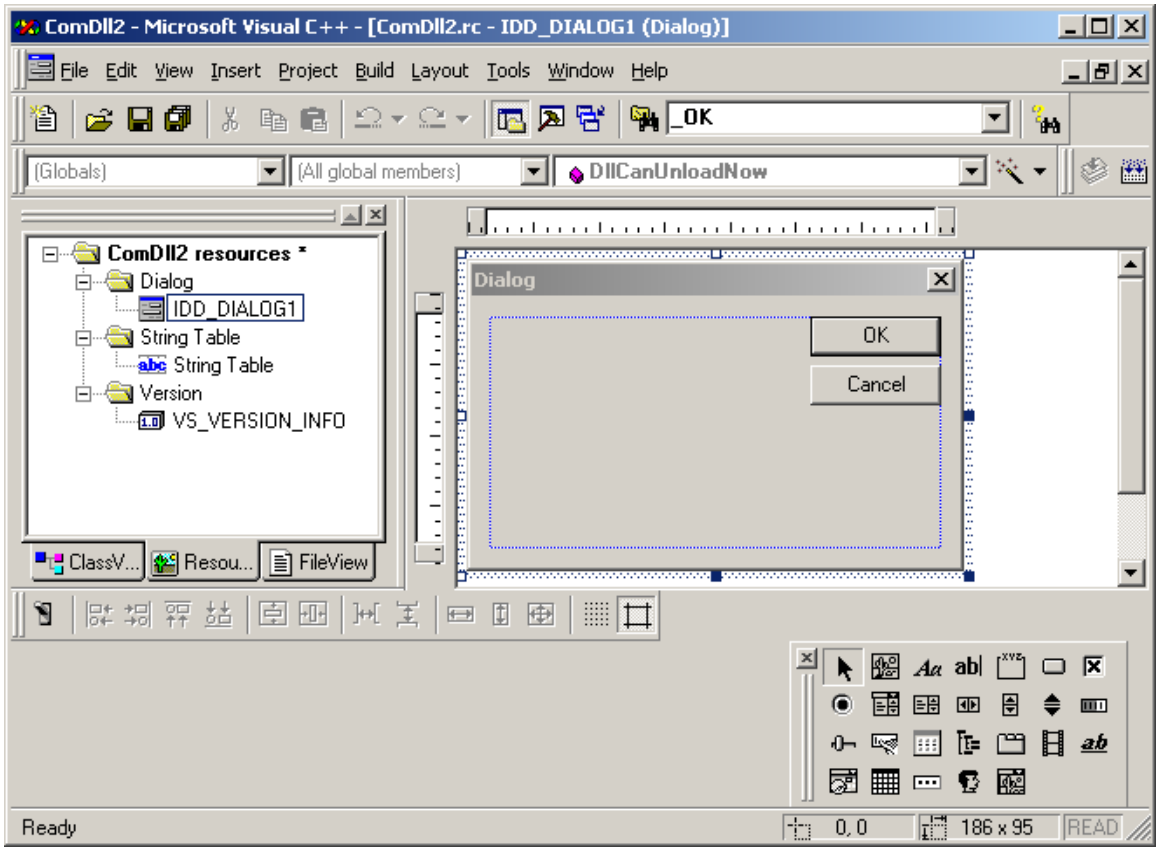


- Go into the Workspace and select the **ResourceView** Tab.

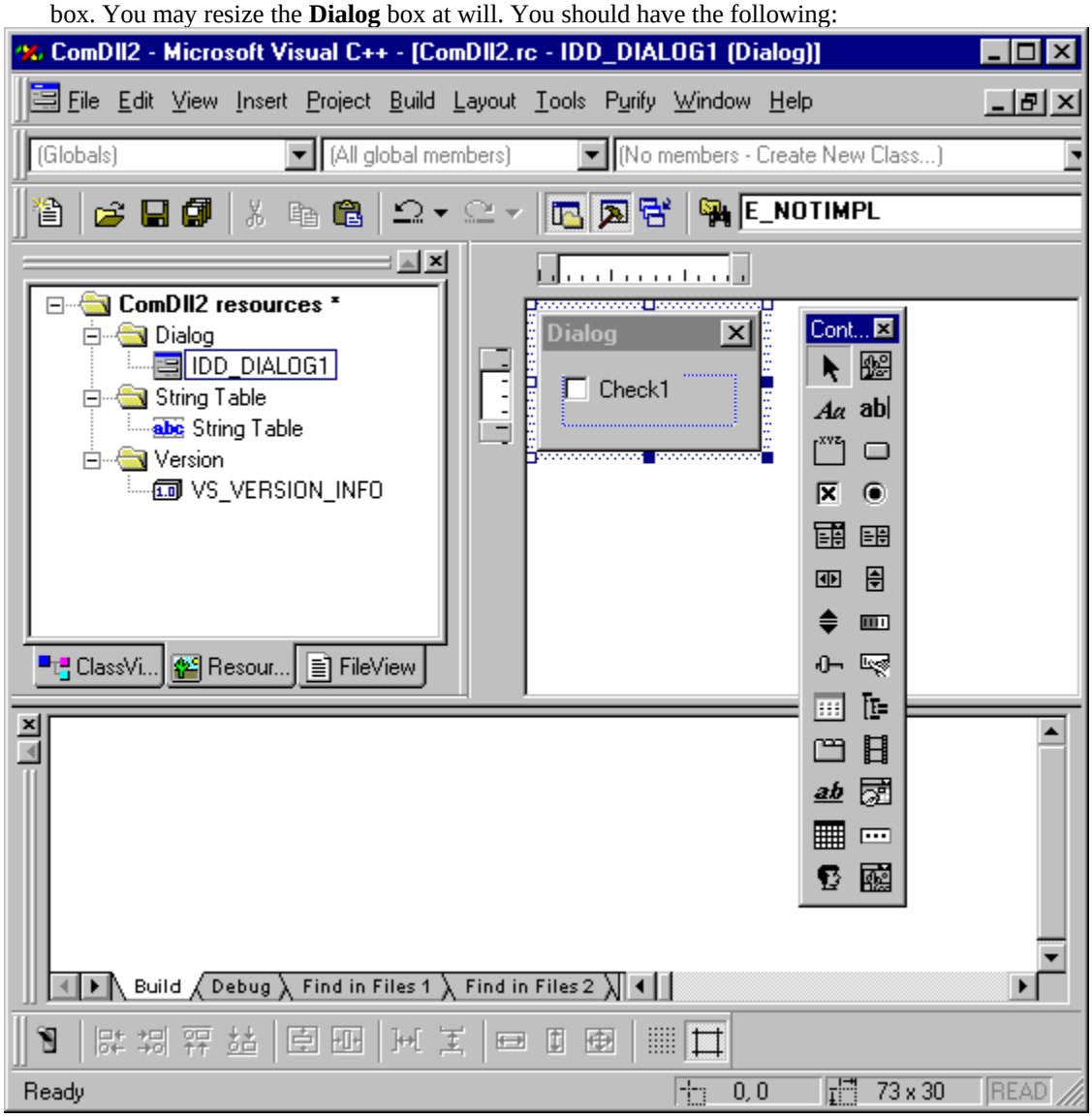
- Right click on **ComDll2 resources** and select insert:



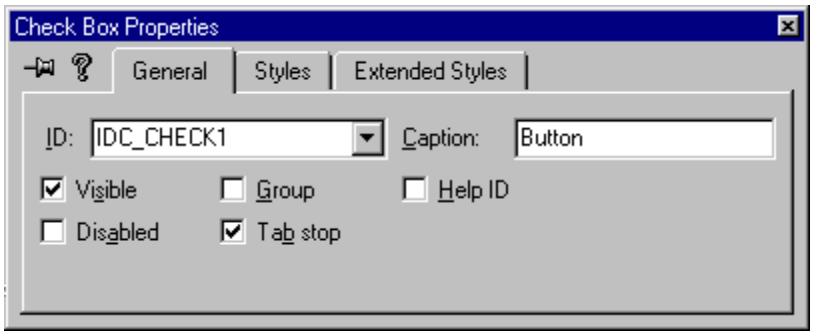
- Click on **Dialog** and select **New**.
- You will now have a the following window:



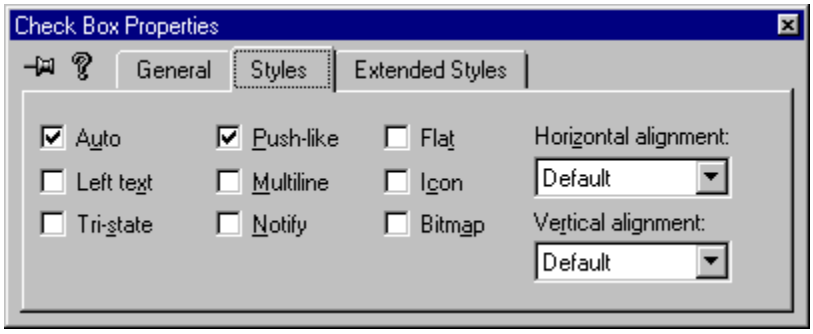
- Left click on the the **OK** and press **delete**. Left click on the **Cancel** button and press **delete**.
- In the **Controls** window, select the **check box** (square item with an **x** in it). Left click on it and drag it onto the **Dialog** box. You may resize the **Dialog** box at will. You should have the following:



- Right click on the checkbox and select the **Properties**. You will see the windows below and change the caption to Button.

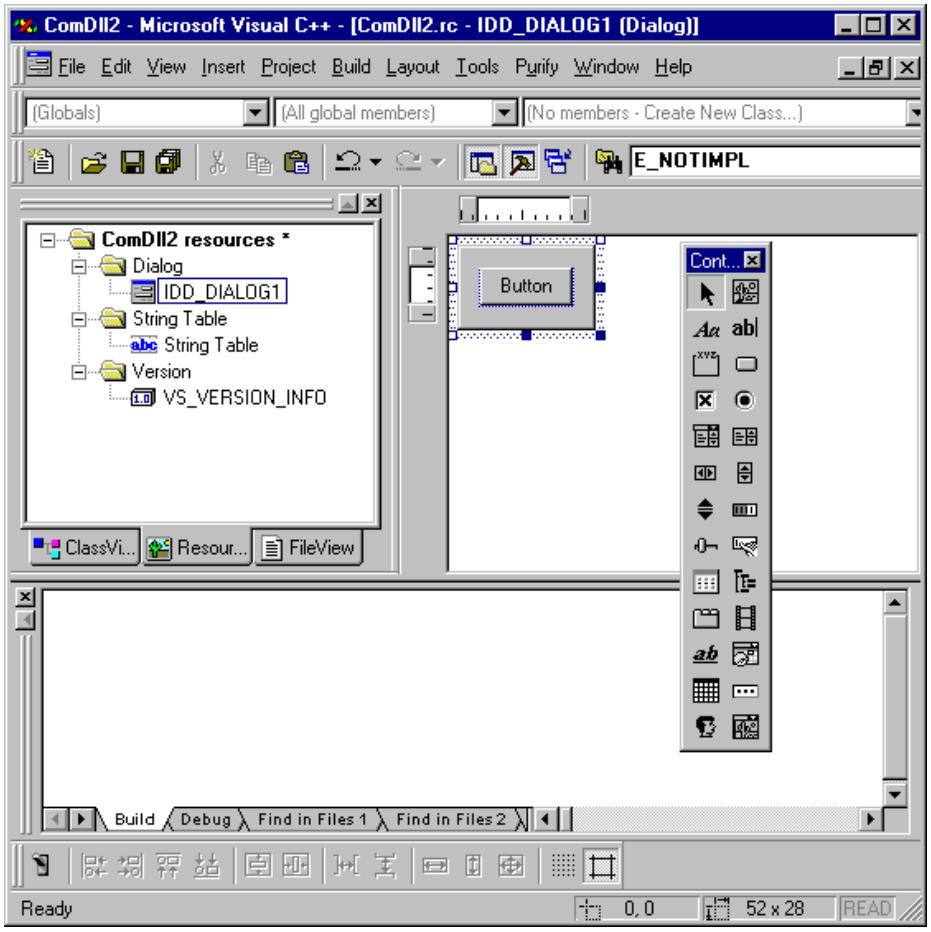


- In the **Style** tab, select **Push-like** check box.

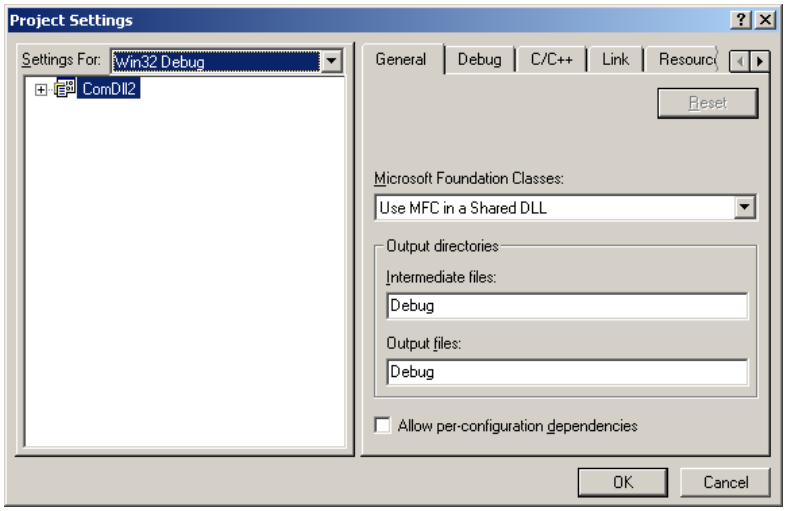


- In the Extended Styles, select the options that you see fit.
- Then right click on the dialog box and select **Properties**. In **Style**, uncheck the **title bar** and close that window:

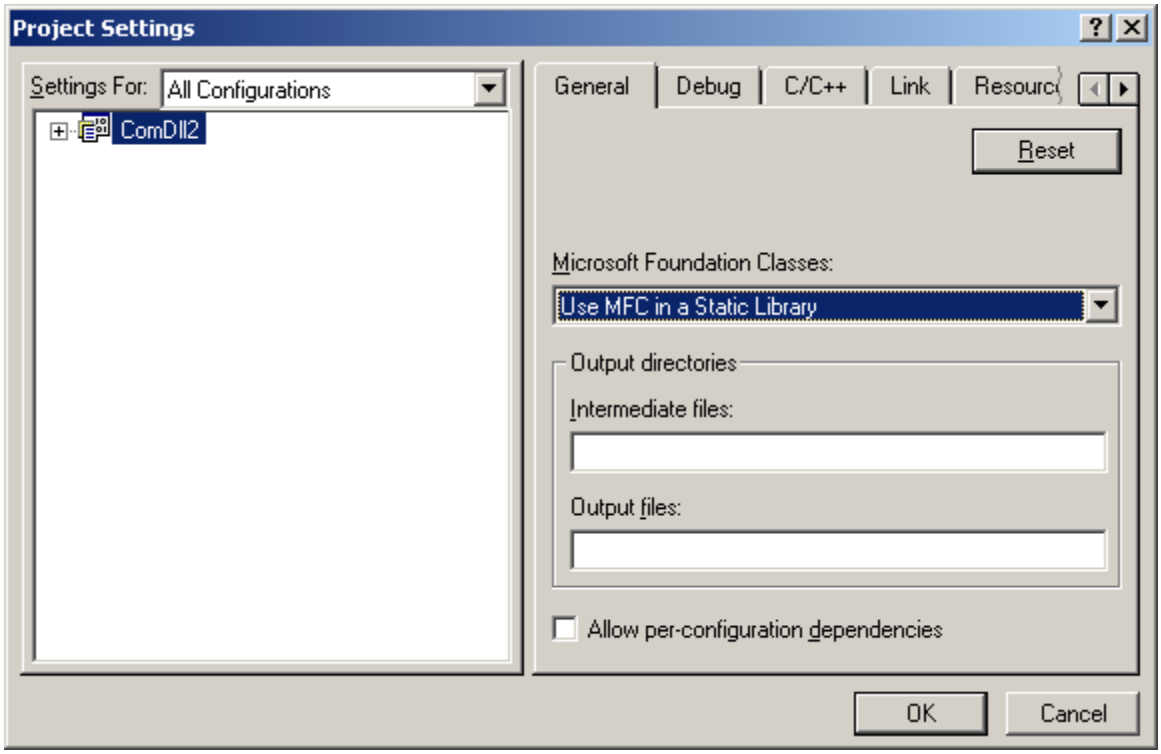
- You will see now have the following project:



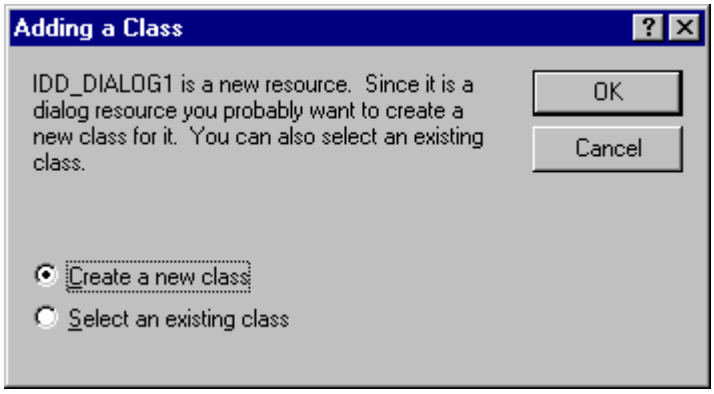
- Go into **Project/Settings…** and you'll see the following window



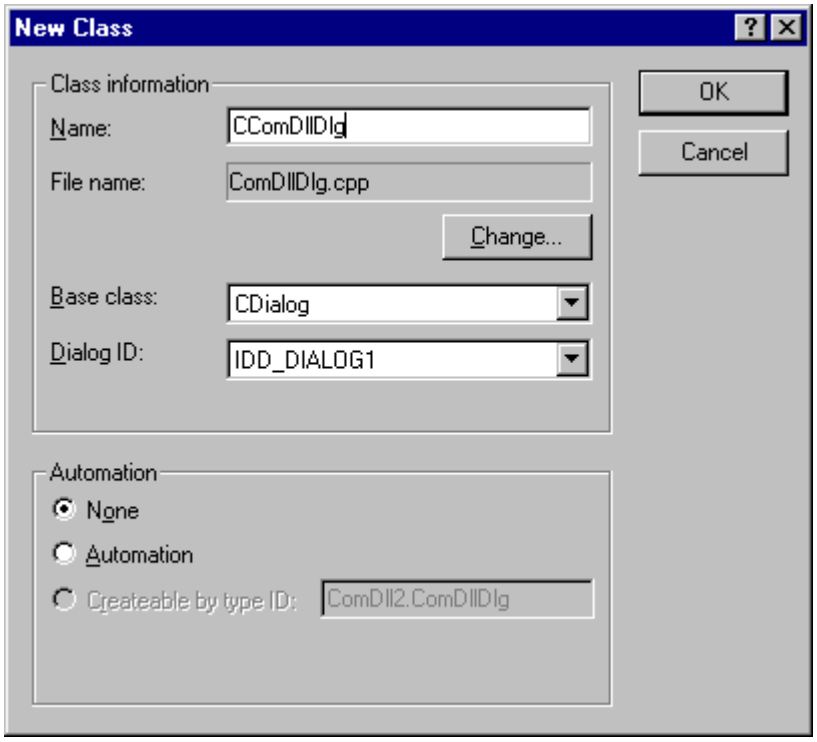
- Select **All configurations** for **Settings For:** and **Use MFC in a Static Library** for **Microsoft Foundation Classes:** as seen below.



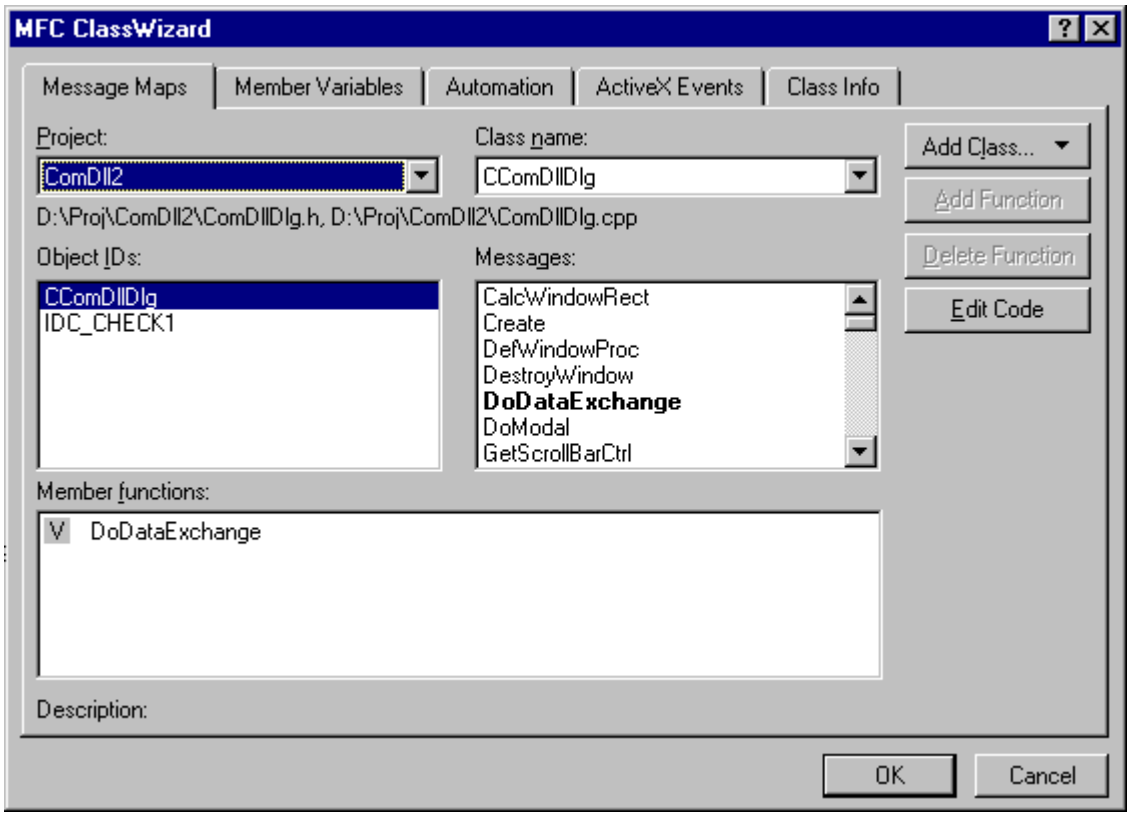
- Click on **OK**.
- Build your project by going into **Build/Build ComDll2.dll** or press **F7**. Your project should build without errors.
- Now, you can write click on the dialog box window and select **Class Wizard**. You will see the following window and click **OK** by accepting the default:



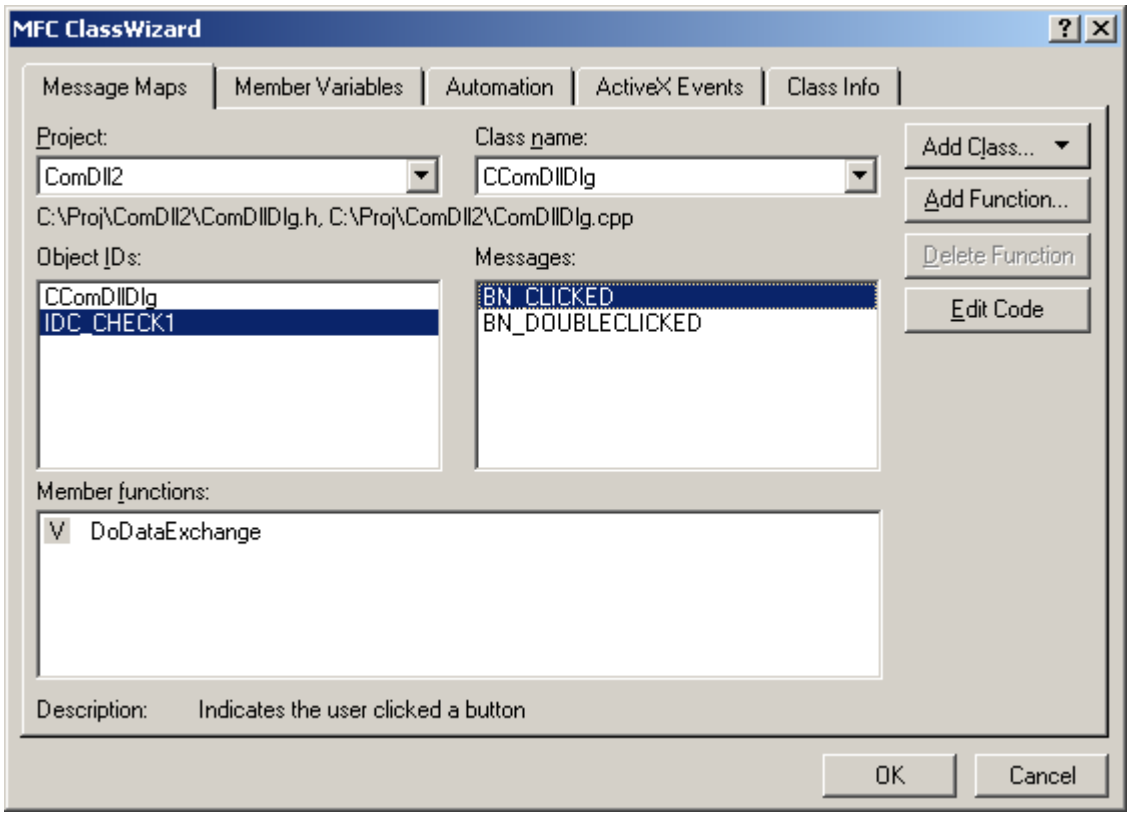
- You will now be asked to chose a name for the new Dialog class and type in **CComDllDlg**:



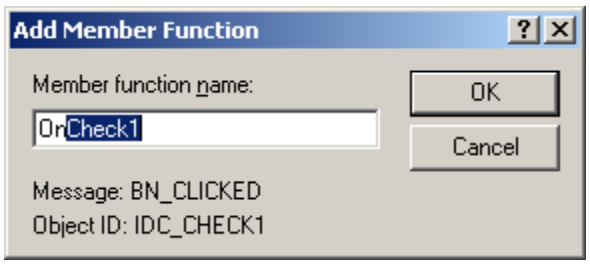
- • Press on **OK**
- • You will now see the following Window:



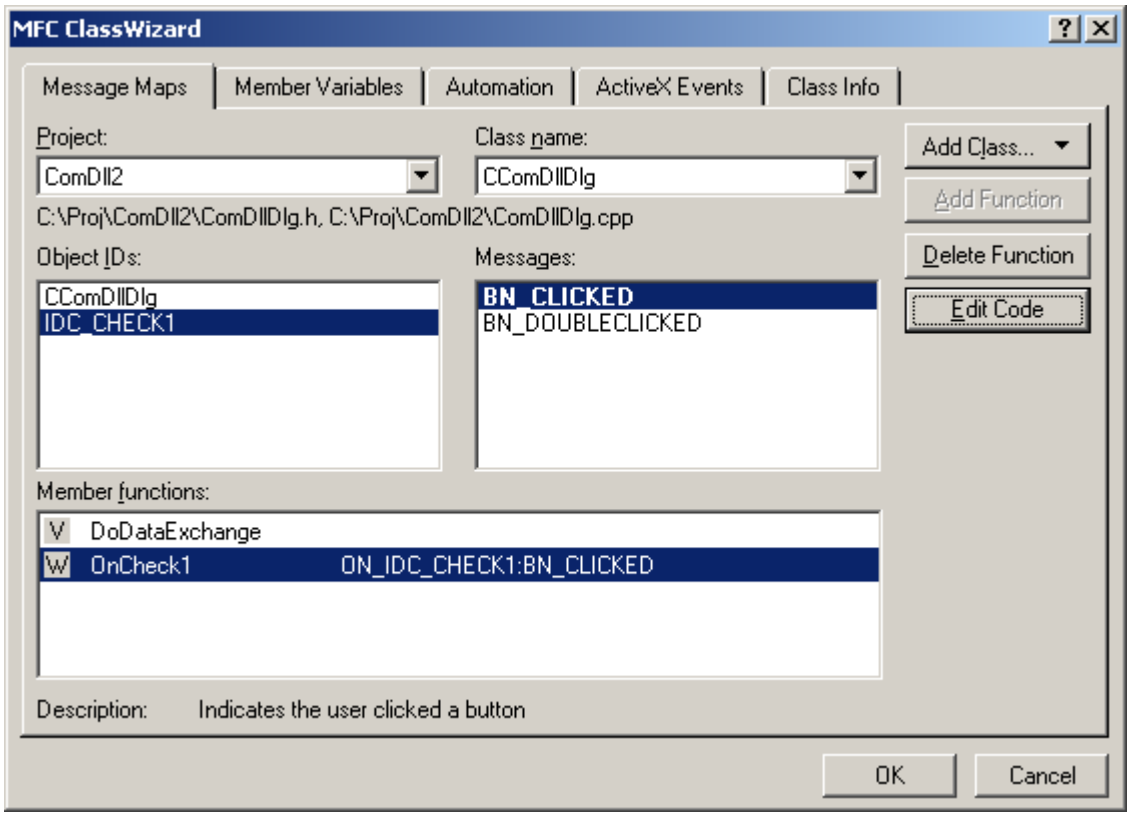
- This is where you may add any functionality related to clicking on the window. We will in our case only be interested in what happens when we click on the button. No implementation of that functionality is necessary in our case.
  In the event that you wished to implement it, you will have to select **IDC_CHECK1** on the **Object Ids** window.
  Then you will click on **BN_CLICKED** in the **Messages:** Window.



- Click on the **Add Function** button. A **Add Member Function** will open.



- Click on **OK** after allowing the default name to be chosen. Now you have allowed the wizard to add the code to your project.

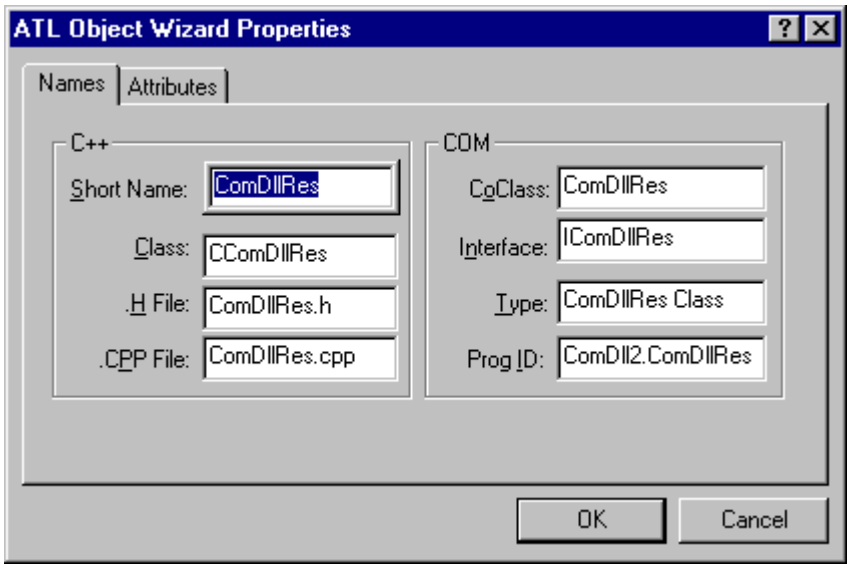


- Click on **OK**.

1.2 1.2 Now, we will add the ATL environment:

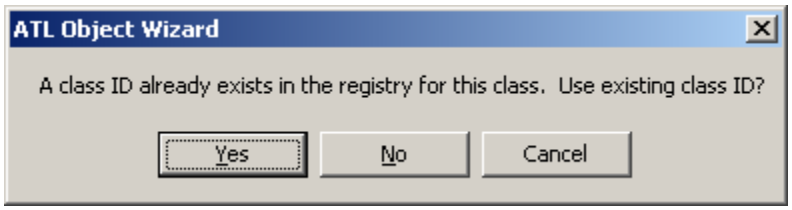
- Go to **Insert** and select **New ATL Object**.
- Now you should see the following window:

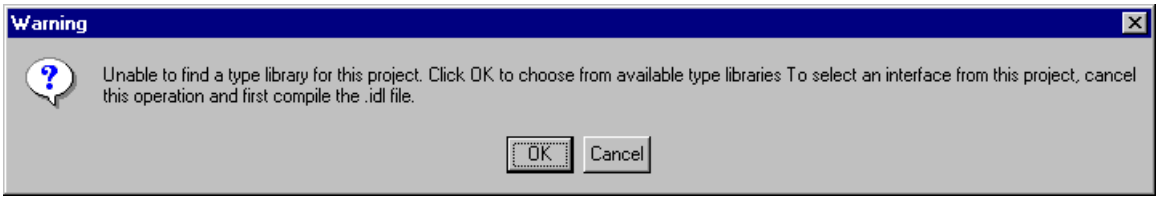
- Select **Simple Object** as seen below and click on **Next**.
- You will then see the following window. In the **Short name:** box, enter the name of the ATL object that you wish to create. The other boxes will fill in by themselves. In this case, we chose the name **ComDllRes**. Don't chose "**IOResource**" because it's already used by the system and it will prevent you from building the COM exe.
  Also, take note of the name in **Prog ID**: In this case **ComDll2.ComDllRes** (dll name followed by the short name). This will be used as an entry in the Wrts options.



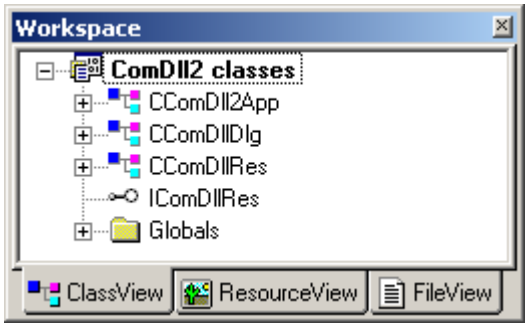
- In the list of attributes, as shown below, all defaults can be acceptable. You may want to add the option of **Support IsupportErrorInfo** as seen below. These options make more sense for those that are familiar with COM.

- Click on **OK**.
- If you get the following window:



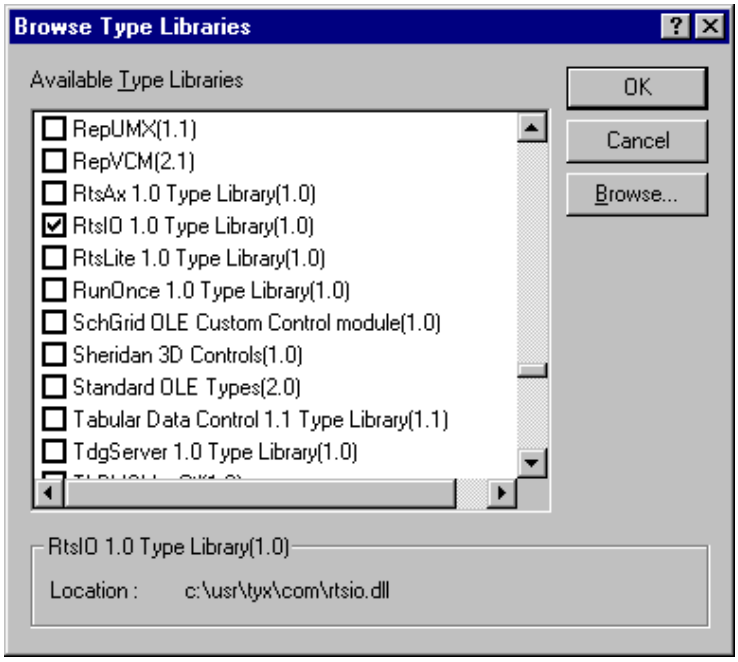


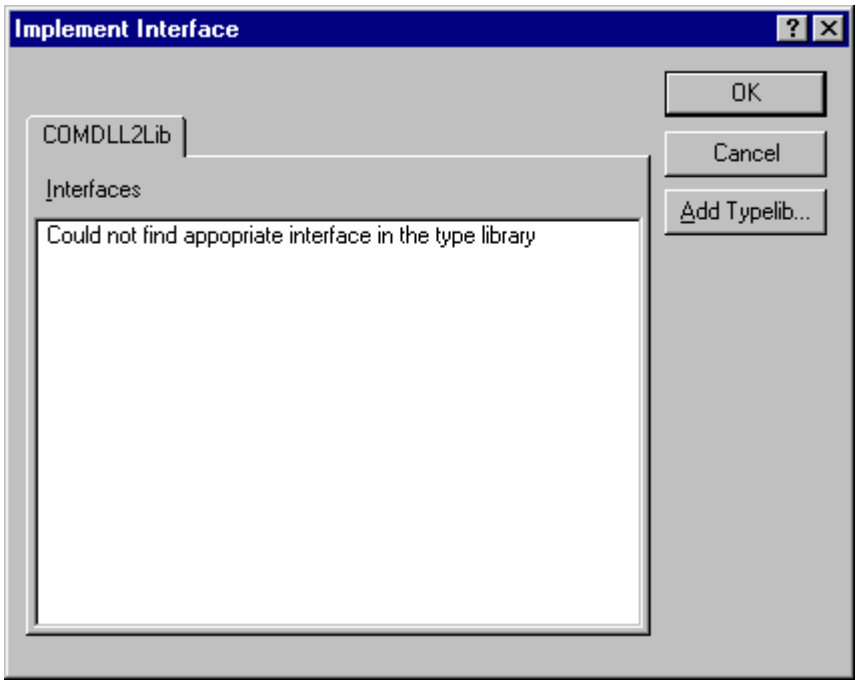
- Click on **Yes**. If not, just proceed.
- Now we are back to the Studio.
- Now you need to link the project to an interface, which in this case relates to the Wrts capabilities. In the workspace, you should right-click on the Class that starts with **C<short name**>, where short name is from the **ATL Object Wizard Properties**. In our case **CComDllRes** as seen below.



- After right clicking on **CComDllRes**, Select **Implement interface…**
- You will get the following message and then click on **OK**.

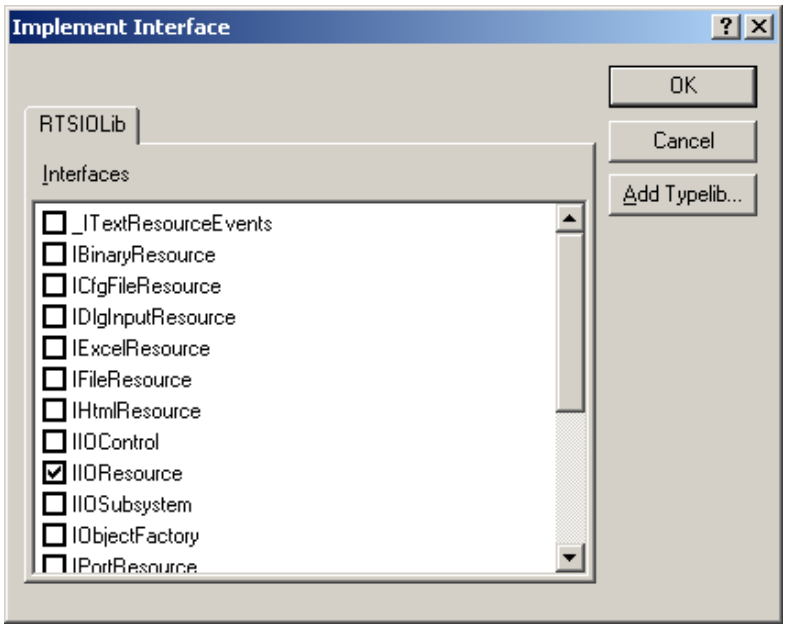
- In the event that you see the following image instead of the previous one, press **AddTypelib…**



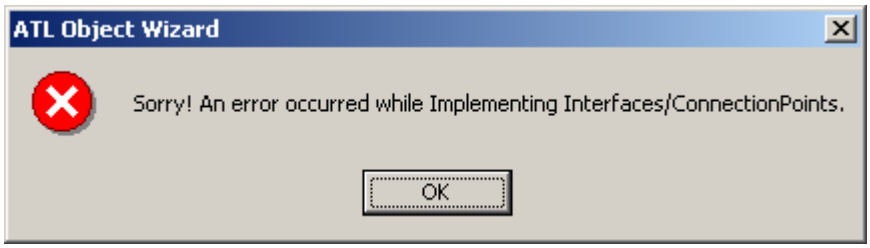


- Select the **RtsIO** for Wrts input-output library and then click on **OK**. If you do not see **RtsIO**, it's because you are using a version of the TYX Stutio that is older than 1.10.x.

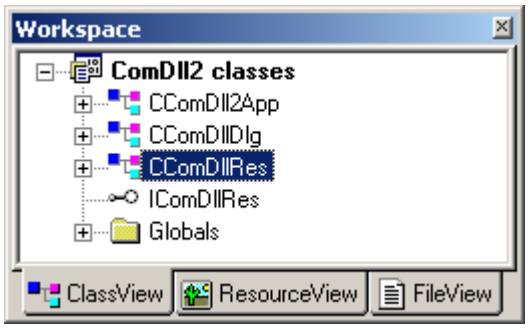
- In the following window, you need to check **IIOResouce** and then click on **OK**. You may chose to select at this time both **IIOResource** and **IBinaryResource**, but it may cause problems.



- If you get an error message:



- Close the project. Delete the **Debug** folder, and any of the files that may be present with the following extensions: **ncb**, **opt**, **plg**, **aps** and **clw** and try again.
- You need to repeat this for **IBinaryResource** (for text IO resources) if you haven't done so already, by repeating the steps above starting at right-clicking on **CComDllRes**.
- If you are still having problems trying to add the interfaces, you can make sure that the **ComDllRes.h** contains the content of the file below.
- Now you will need to modify the source of what has been made available to you. In our case, double click on **CComDllRes**, or **C<short name>** as shown below:




```
// ComDllRes.h : Declaration of the CComDllRes

#ifndef __COMDLLRES_H_
#define __COMDLLRES_H_

#include "resource.h"       // main symbols
#import "c:\usr\tyx\com\rtsio.dll" raw_interfaces_only, raw_native_types, no_namespace, named_guids

/////////////////////////////////////////////////////////////////////////////
// CComDllRes
class ATL_NO_VTABLE CComDllRes :
        public CComObjectRootEx<CComSingleThreadModel>,
        public CComCoClass<CComDllRes, &CLSID_ComDllRes>,
        public ISupportErrorInfo,
        public IDispatchImpl<IComDllRes, &IID_IComDllRes, &LIBID_COMDLL2Lib>,
        public IBinaryResource,
        public IIOResource
{
public:
        CComDllRes()
        {
        }

DECLARE_REGISTRY_RESOURCEID(IDR_COMDLLRES)

DECLARE_PROTECT_FINAL_CONSTRUCT()

BEGIN_COM_MAP(CComDllRes)
        COM_INTERFACE_ENTRY(IComDllRes)
        COM_INTERFACE_ENTRY(IDispatch)
        COM_INTERFACE_ENTRY(ISupportErrorInfo)
        COM_INTERFACE_ENTRY(IBinaryResource)
        COM_INTERFACE_ENTRY(IIOResource)
END_COM_MAP()

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

// IComDllRes
public:
// IBinaryResource
        STDMETHOD(Read)(LONG lType, VARIANT * pVal)
        {
                if (pVal == NULL)
                        return E_POINTER;

                return E_NOTIMPL;
        }
        STDMETHOD(Write)(LONG lType, VARIANT val)
```

```
	{
		return E_NOTIMPL;
	}
// IIOResource
	STDMETHOD(Open)(BSTR bstrName, LONG lMode)
	{
		return E_NOTIMPL;
	}
	STDMETHOD(Close)()
	{
		return E_NOTIMPL;
	}
	STDMETHOD(Flush)()
	{
		return E_NOTIMPL;
	}
	STDMETHOD(Abort)()
	{
		return E_NOTIMPL;
	}
	STDMETHOD(Seek)(LONG lOffset, SHORT sOrigin)
	{
		return E_NOTIMPL;
	}
	STDMETHOD(get_name)(BSTR * pVal)
	{
		if (pVal == NULL)
			return E_POINTER;

		return E_NOTIMPL;
	}
	STDMETHOD(get_Mode)(LONG * pVal)
	{
		if (pVal == NULL)
			return E_POINTER;

		return E_NOTIMPL;
	}
	STDMETHOD(get_Size)(LONG * pVal)
	{
		if (pVal == NULL)
			return E_POINTER;

		return E_NOTIMPL;
	}
	STDMETHOD(get_Position)(LONG * pVal)
	{
		if (pVal == NULL)
			return E_POINTER;

		return E_NOTIMPL;
	}
	STDMETHOD(get_Eof)(VARIANT_BOOL * pVal)
	{
		if (pVal == NULL)
			return E_POINTER;

		return E_NOTIMPL;
	}
	STDMETHOD(get_State)(LONG * pVal)
	{
		if (pVal == NULL)
			return E_POINTER;
```

```
                return E_NOTIMPL;
        }
};

#endif //__COMDLLRES_H_
```

For complete information on each one of those functions, please refer to the TYX website or the updated online help.

- What you will want to do first, is to change the exit code for all of those functions, from **E_NOTIMPL** which is the return for "Error, not implemented", to **S_OK** for an ok return value.
  - For example: For the **Open** and **Close** functions, you will get:

```
STDMETHOD(Open)(BSTR bstrName, LONG lMode)
{
        return S_OK;  // Changed
}
STDMETHOD(Close)()
{
        return S_OK;  // Changed
}
```

Those particular functions are called by the Wrts upon loading and unloading the dll with the Wrts. This is where you would put the code that you wish to see executed when the Wrts loads and unloads an Atlas program.

- You will also want to delete the following line on line 16 or 17:

```
public IIOResource
```

in order to avoid compilation problems. Make sure that you also delete the comma at the end of line 16 if `public IIOResource` is on line 17.

- You will also want to return an end of file true statement in the get_Eof method:

```
STDMETHOD(get_Eof)(VARIANT_BOOL * pVal)
{
        if (pVal == NULL)
                return E_POINTER;
        *pVal = VARIANT_TRUE;                // Added
        return S_OK;                         // Changed
}
```

- At the end of the **CComDllRes** class definition, you need to make the dialog a member variable so that it can be accessible throughout the resource. One place to put it would be on line 116.

CComDllDlg theDlg;

For the above to build properly, you then need to include the header file that has the definition for the class above. On line 7, you should now add this include:

#include "ComDllDlg.h"

- The code below is where the binary input and output are being handled. Some code has been added in order to achieve the goal that we have set ourselves in this example, which is to pass on and retrieve an integer. Typically, you will implement your own code as a function of your own requirements:

```
// IBinaryResource
        STDMETHOD(Read)(LONG lType, VARIANT * pVal)
        {
```

```
        if (pVal == NULL)
                return E_POINTER;

        pVal->vt = VT_I4;                                               // Added
        pVal->lVal = (int)theDlg.IsDlgButtonChecked(IDC_CHECK1);        // Added

        return S_OK;                                                    // Changed
}
STDMETHOD(Write)(LONG lType, VARIANT val)
{
        if(val.lVal == 1)                                               // Added
                theDlg.CheckDlgButton(IDC_CHECK1, BST_CHECKED);         // Added
        else if (val.lVal == 0)                                         // Added
                theDlg.CheckDlgButton(IDC_CHECK1, BST_UNCHECKED);       // Added
        else                                                            // Added
                return E_FAIL;                                          // Added

        return S_OK;                                                    // Changed
}
```

At this point, if you want to refer to the dialog object, you need to use **theDlg**. That will give you access to the methods associated to the dialog Class.
pVal->vt will set the type of the variant to an integer. The following line will retrieve the information whether the button is checked or not and set the variant equal to it.
In **Write**, we retrieve the dialog object for the window the exact same way, by using **theDlg**. The following lines check the button as a function of what is passed on by the Atlas.
There are naturally different ways to do the same thing, and this is only one of them.

The Output on the other hand will display the content of the output string from the Atlas.

- If you wish to see the window appear when loading an Atlas project and disappear when unloading it, you need to go into the **ComDllRes.h** file and:

Change the code for the **Open** and **Close** function to the following code:

```
STDMETHOD(Open)(BSTR bstrName, LONG lMode)
{
        AFX_MANAGE_STATE(AfxGetStaticModuleState());            // Added
        BOOL ret = theDlg.Create(IDD_DIALOG1, NULL);            // Added
        if (!ret)                                               // Added
                AfxMessageBox("Error creating TM_Button Dialog"); // Added
        theDlg.ShowWindow(SW_SHOW);                             // Added

        return S_OK;                                            // Changed
}
STDMETHOD(Close)()
{
        theDlg.DestroyWindow();                                 // Added

        return S_OK;                                            // Changed
}
```

For open, it is a good idea to use the **AFX_MANAGE_STATE** command for clean code. This is necessary in the event that you use MFC as a shared library and not a static one.
Then, the object is created and then displayed. Open is called when the Altas program is loaded.
Close is called when the Atlas program is unloaded. For clean code, you want to destroy the object at this point.

- You are now ready to build the exe with the option **Build/Build**, by using the icon bar, or by pressing **F7**.
- **Note for advanced users:** If you are not an advanced COM user, you may skip this note.

If you want to make use of more than one IO resource, such as text and binary, you may have a compilation problem unless you correct some code.

In the file **<Short Name>.h** from the **ATL Object Wizard Properties**, **ComDllRes.h** in our case, in the COM_MAP section, you may have an uncertainty about the mapping between the IIOResource and the resource that you want to use. In our case, you will have the code below:

```
BEGIN_COM_MAP(CComDllRes)
    COM_INTERFACE_ENTRY(IComDllRes)
    COM_INTERFACE_ENTRY(IDispatch)
    COM_INTERFACE_ENTRY(ISupportErrorInfo)
    COM_INTERFACE_ENTRY(IBinaryResource)
    COM_INTERFACE_ENTRY(IIOResource)
END_COM_MAP()
```

If you had another resource such as text, you will have to tell IIOResouce which resource to use in order to satisfy the compiler. To do that, you will need to change the **COM_INTERFACE_ENTRY** for **IIOResource** to the line below.

```
BEGIN_COM_MAP(CComDllRes)
    COM_INTERFACE_ENTRY(IComDllRes)
    COM_INTERFACE_ENTRY(IDispatch)
    COM_INTERFACE_ENTRY(ISupportErrorInfo)
    COM_INTERFACE_ENTRY2(IIOResource, IBinaryResource)  // Necessary to point to the
                                                        // desired class since there are two of them (Text and Binary).
    COM_INTERFACE_ENTRY(ITextResource)
    COM_INTERFACE_ENTRY(IBinaryResource)
END_COM_MAP()
```

That will ensure that your code will build properly, and you may then make use of both IO resource types.

This is the following code in **ComDllRes.h** that will do that.

1.3 1.3 What is wrong and how to fix it?

1.3.1 1.3.1 How to move the ComDll2.dll?

You will need to reregister the exe manually.
To do this, you need to go into a DOS window, move up to the subdirectory where the Proj6.exe is and execute the following command:

regSvr32 ComDll2.dll or in any other case **regSvr32 <your project>.dll**

Please note that for Win98, regsvr32 is not accessible from anywhere. You will need to add the path to Regsvr32 and if necessary the path for the dll also.
For NT, regsvr32 can be invoked from anywhere.

To unregister the Dll, you will want to use:

regSvr32 /u ComDll2.dll

2 2 What to do in the TYX Studio?

- At this point, you will need to have an Atlas program that makes use of that COM resource and to setup the Wrts in order to link that Atlas IO device to you COM dll.

2.1 2.1 Sample for Atlas 416:

Note: The version of the Atlas 416 has to be high enough to include the **INPUT** and **OUTPUT** commands. Earlier versions of 416 Atlas do not include those commands.

```
 001000 BEGIN, ATLAS PROGRAM 'BUTTONS'                                  $
 001010 REQUIRE, 'TM_BUTTON', I-O DEVICE,
            CAPABILITY,
            FILE-SIZE 80 WORDS                                          $
C                                                                       $
 001020 DECLARE, INTEGER, STORE, 'TM1'                                  $
 001025 DECLARE, INTEGER, STORE, 'I'                                    $
 001030 DECLARE, INTEGER, STORE, 'TMSTATUS'                             $
C                                                                       $
E010000 DISPLAY, MESSAGE, *********** THE START ***************
                                                                        $
        DISPLAY, MESSAGE, *********** OUTPUT ******************
                                                                        $
 010010 CALCULATE, 'TM1' = 1                                            $
 010020 OUTPUT, USING 'TM_BUTTON', ('TM1')                              $
        DISPLAY, MESSAGE, 'BUTTON ON'
        $
        WAIT FOR, 1 SEC BEFORE STEP 010030                              $
 010030 CALCULATE, 'TM1' = 0                                            $
 010040 OUTPUT, USING 'TM_BUTTON', ('TM1')                              $
        DISPLAY, MESSAGE, 'BUTTON OFF'
        $
        WAIT FOR, 1 SEC BEFORE STEP 010050                              $
 010050 CALCULATE, 'TM1' = 1                                            $
 010060 OUTPUT, USING 'TM_BUTTON', ('TM1')                              $
        DISPLAY, MESSAGE, 'BUTTON ON'
        $
        WAIT FOR, 1 SEC BEFORE STEP 010070                              $
C                                                                       $
 010070 CALCULATE, 'TM1' = 1                                            $
 010080 OUTPUT, USING 'TM_BUTTON', ('TM1')                              $
        DISPLAY, MESSAGE, *********** INPUT ******************
                                                                        $
        FOR, 'I' = 1 THRU 5 BY 1, THEN                                  $
 010090         INPUT, USING 'TM_BUTTON', 'TMSTATUS'                    $
 010100         DISPLAY, MESSAGE, TMSTATUS -$
                DISPLAY, RESULT, 'I'                                    $
                DISPLAY, MESSAGE, /5 - = $
 010110         DISPLAY, RESULT, 'TMSTATUS'                             $
 010120         DISPLAY, MESSAGE,  -Press Manual intervention to continue-
                                                                        $
                WAIT FOR, MANUAL INTERVENTION                           $
        END, FOR                                                        $
 010190 DISPLAY, MESSAGE, *********** THE END ******************
                                                                        $
 010200 TERMINATE, ATLAS PROGRAM 'BUTTONS'                              $
```

This atlas will simply demonstrate two things:
It will output updated status of the button from checked to unchecked and vice-versa, and it will retrieve the status of the button in the Atlas via a Binary IO resource.

You can note that for IEEE416 Atlas, you may use the same COM resource for input and output only with newer versions of 416 and only with a Binary IO.
You will need to compile this Atlas and have it ready to run.

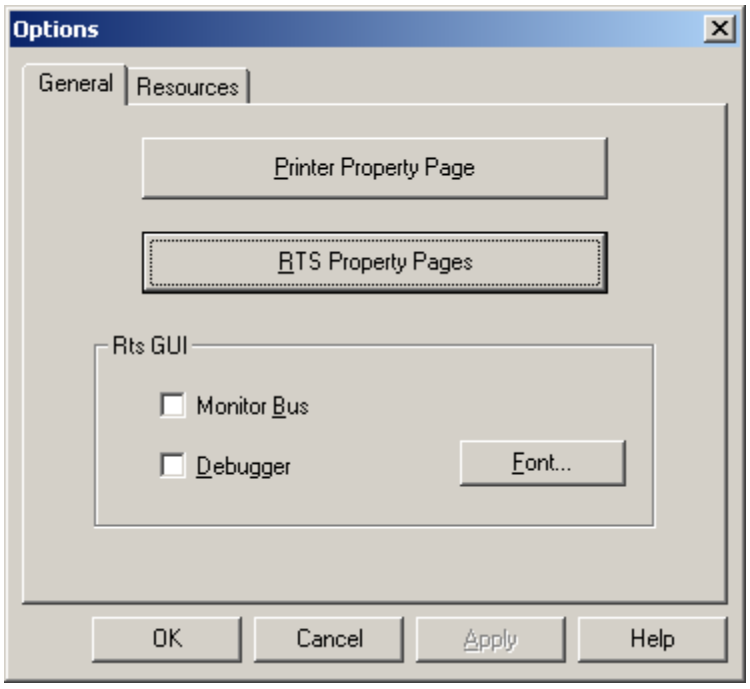
2.2 2.2 Sample for Atlas 716-89/95:

```
 001000 BEGIN, ATLAS PROGRAM 'BUTTONS'                                  $
     10 DECLARE, VARIABLE, 'TXT_DATA' IS STRING(80) OF CHAR             $
     12 DECLARE, VARIABLE, 'OUT' IS FILE OF UNTYPED                     $
C                                                                       $
     20 DECLARE, VARIABLE, 'TM1' IS INTEGER                             $
     25 DECLARE, VARIABLE, 'I' IS INTEGER                               $
     30 DECLARE, VARIABLE, 'TMSTATUS' IS INTEGER                        $
C                                                                       $
E010000 OUTPUT, C'*********** THE START ***************'               $
     10 OUTPUT, C'\LF\*********** OUTPUT ******************'           $
C                                                                       $
     11 CALCULATE, 'TM1' = 1                                            $
     12 ENABLE, I-O NEW C'TM_BUTTON', VIA 'OUT'                         $
     22 OUTPUT, TO 'OUT', 'TM1'                                         $
     25 OUTPUT, C'BUTTON ON'                                            $
     32 WAIT FOR, 1 SEC                                                 $
     35 CALCULATE, 'TM1' = 0                                            $
     45 OUTPUT, TO 'OUT', 'TM1'                                         $
     48 OUTPUT, C'BUTTON OFF'                                           $
     55 WAIT FOR, 1 SEC                                                 $
     57 CALCULATE, 'TM1' = 1                                            $
     65 OUTPUT, TO 'OUT', 'TM1'                                         $
     70 OUTPUT, C'BUTTON ON'                                            $
     75 WAIT FOR, 1 SEC                                                 $
C                                                                       $
     77 CALCULATE, 'TM1' = 1                                            $
     85 OUTPUT, C'\LF\*********** INPUT *******************'
                                                                        $
     95 FOR, 'I' = 1 THRU 5 BY 1, THEN                                  $
     97         INPUT, FROM 'OUT', INTO 'TMSTATUS'                      $
 010100         OUTPUT, C'TMSTATUS ---', 'I', C'/ 5 --- =', 'TMSTATUS' $
     20         OUTPUT, C'Press TRUE or FALSE to continue...\LF\'       $
     30         INPUT, GO-NOGO                                          $
     40 END, FOR                                                        $
     50 DISABLE, 'OUT'                                                  $
     90 OUTPUT, C'*********** THE END ******************'              $
C                                                                       $
 010200 TERMINATE, ATLAS PROGRAM 'BUTTONS'                              $
```

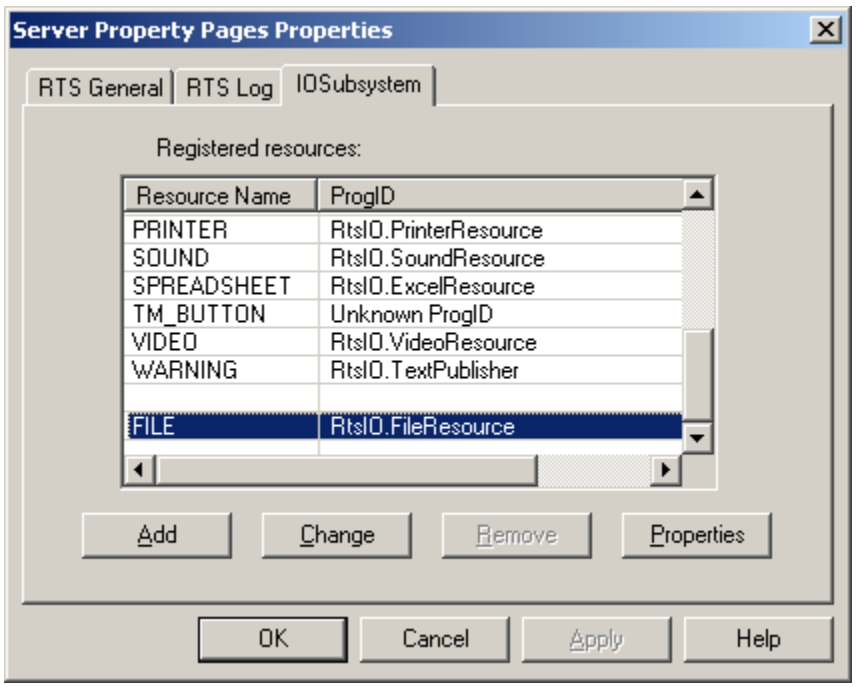
This atlas will simply demonstrate two things:
It will output updated status of the button from checked to unchecked and vice-versa, and it will retrieve the status of the button in the Atlas via a Binary IO resource. The resource will be visible after is has been enabled and will close when it is disabled.
You will need to compile this Atlas and have it ready to run.

2.3 2.3 Wrts Settings:

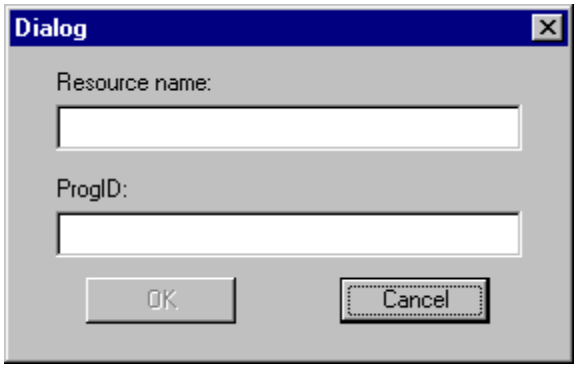
- Now that the Atlas is ready to run, you should launch the Wrts.
- Go into **Control/Options…**



- Now Click on **RTS Property Pages** and select the **IOSubsystem** tab



- You now need to create the link between your Atlas IO resource and the COM resource that you wish to link to. Go into the list of Resource Names. If **TM_BUTTON** is already there with an unknown **ProgID**, remove it by selecting it and



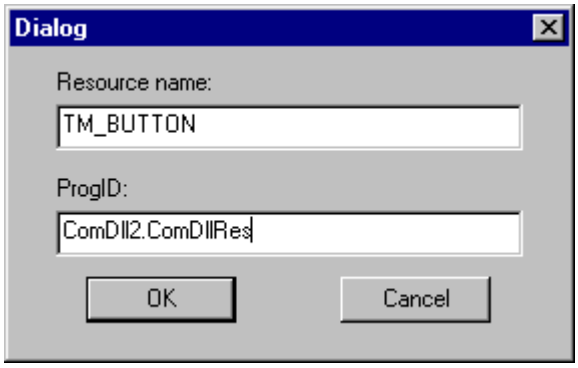
by pressing **Remove**.

- Press **Add** and you will see the following window:

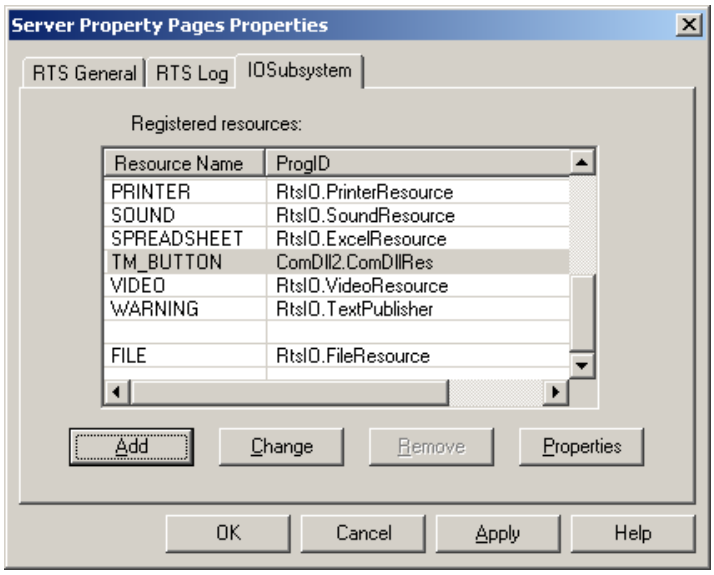
The **Resource name** is the one that you have defined in your Atlas. In our case **TM_BUTTON**.

Your **ProgID** will be the one that you will have remembered from **Atlas Object Wizard Properties** in the previous task of generating the COM exe source.

In this case, you will want to use **ComDll2.ComDllRes**:



- Click on **OK** on this window. Both of the entries will appear in the **Server Property Pages Properties**.



- If this step fails, you need to make sure that the I\O resource exe (**ComDll2** in this case) is registered before trying again. Your Wrts is now ready to link your Atlas IO resource to your COM dll.

Note: The effect will take place the next time you will reopen the Wrts, so close the Wrts and reopen it.

You may now run the Atlas program and the COM dll will be invoked when necessary. The location of the COM dll does not matter because it has been registered in your system.

3 3 How to debug the Com dll?

- First, you put the breakpoints where you want them in your dll COM source code. You would now run the COM dll in debug mode from MSVC++ by going into **Build/Start debug/Go** or by pressing **F5**.
- Now you would go into the TYX studio and run the Wrts.

The run time system would execute until it used the IO Com dll that you have running in debug mode. The execution of the Wrts would stop at your breakpoints in the COM dll source.