TYX CORPORATION

Productivity Enhancement Systems

Reference	TYX_0051_6
Revision	1.1b
Document	PawsIOHowTo3.doc
Date	October 9, 2003

Binary I\O exe

MFC tutorial

This document will help with the making of a Binary input-output 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 the Studio version is not guarantied.

Introduction: First we will address the issue of what to do with the exe 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 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.

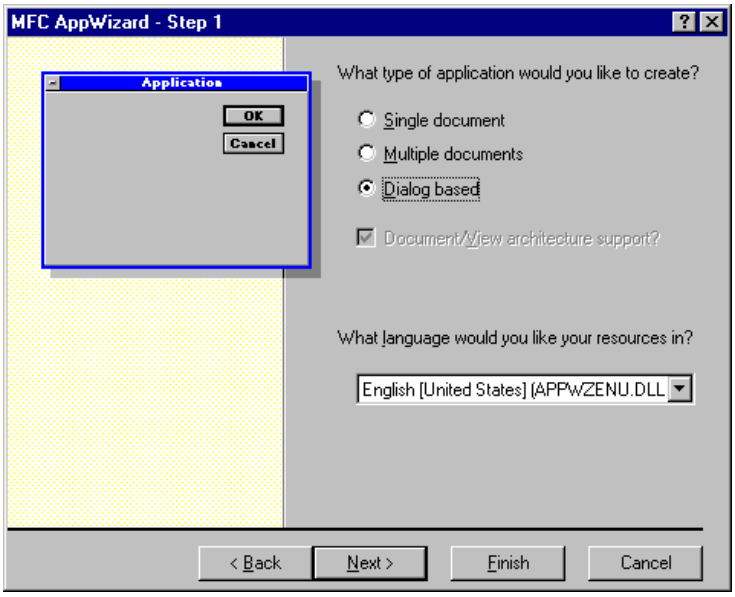
Dll versus Exe:

1. 1. **Advantage of Dlls versus Exe:**
   - **The Dll uses fewer 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… 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. 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.**

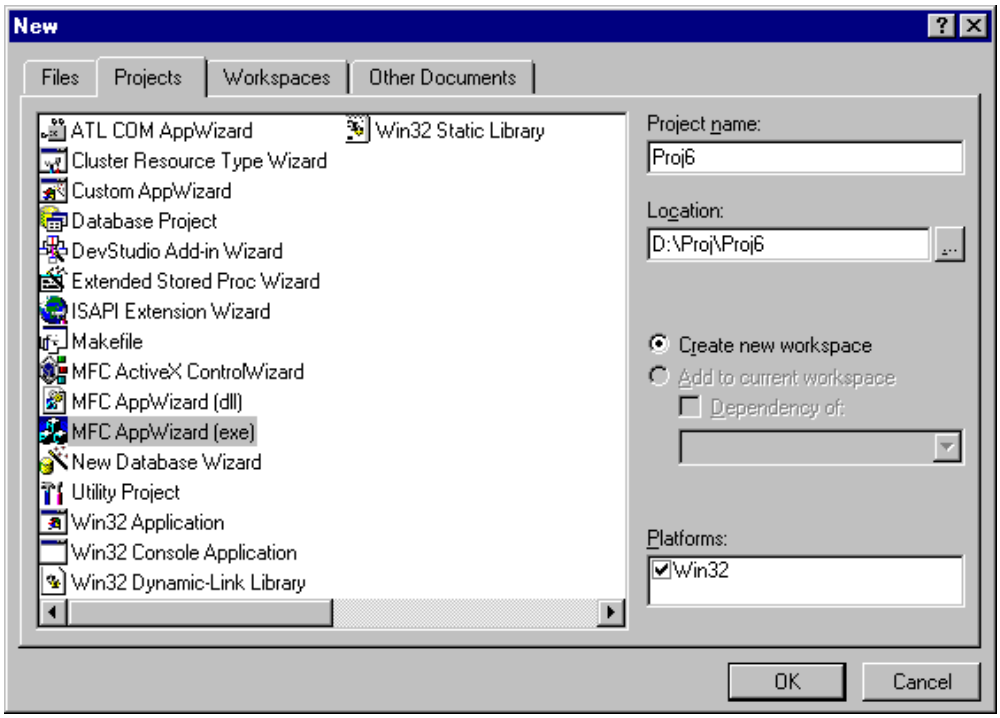
1 1 When generating an exe with MSVC 6++

Combining ATL and MFC is not totally straight forward 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 MFC project, dialog based with one toggle button:

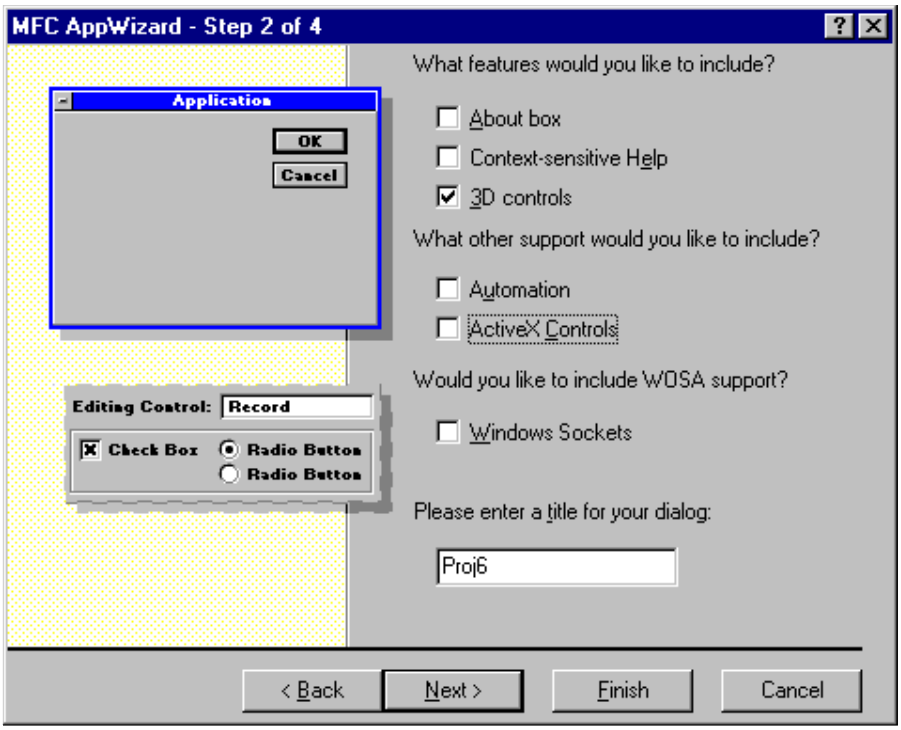


- From **File/New**…, select an **MFC AppWizard (exe)** and select a project name as seen below:

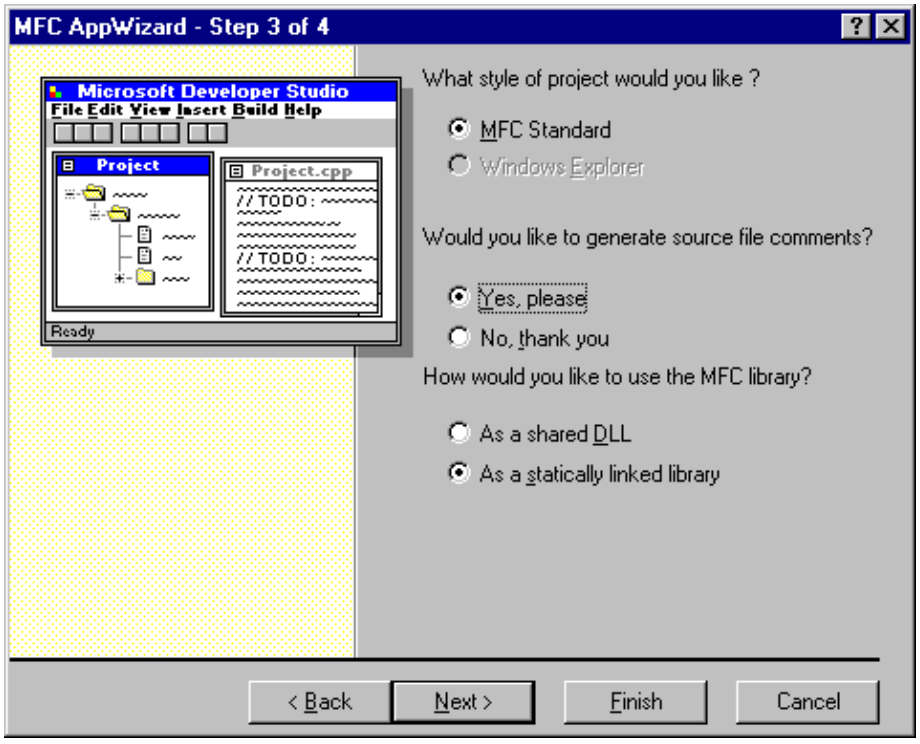


- Click on **OK**
- Then select a **Dialog Based** project as seen below and click on **Next**:

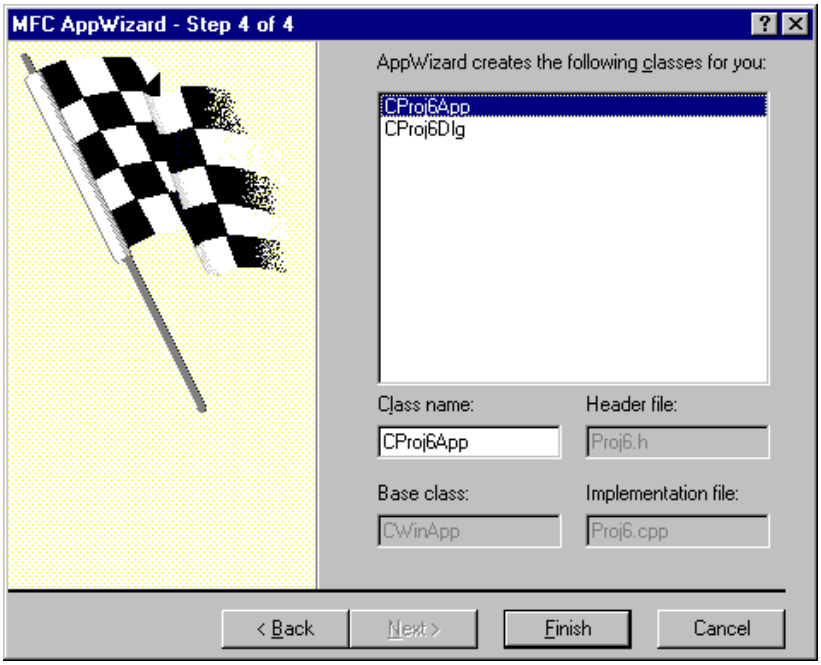
- We will here uncheck the **About box** check box because we will not be using it. It is here that you will decided



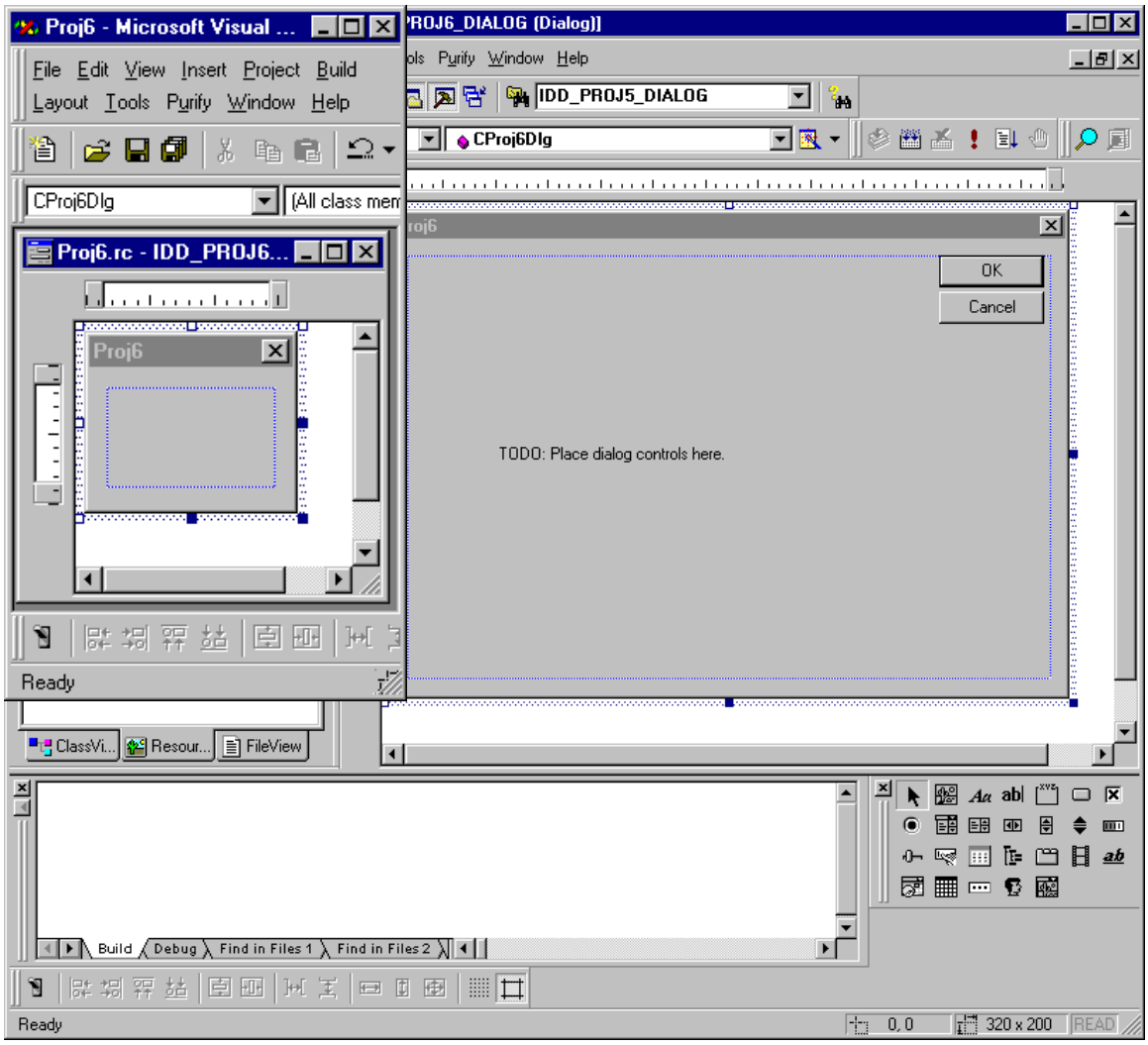
whether you want to include **ActiveX Controls**. We will not be using it in our example. Uncheck **ActiveX Controls** and then on **Next**:



- In the window below, we will want to link the **MFC library** statically for greater independence of our exe and click on **Next**:



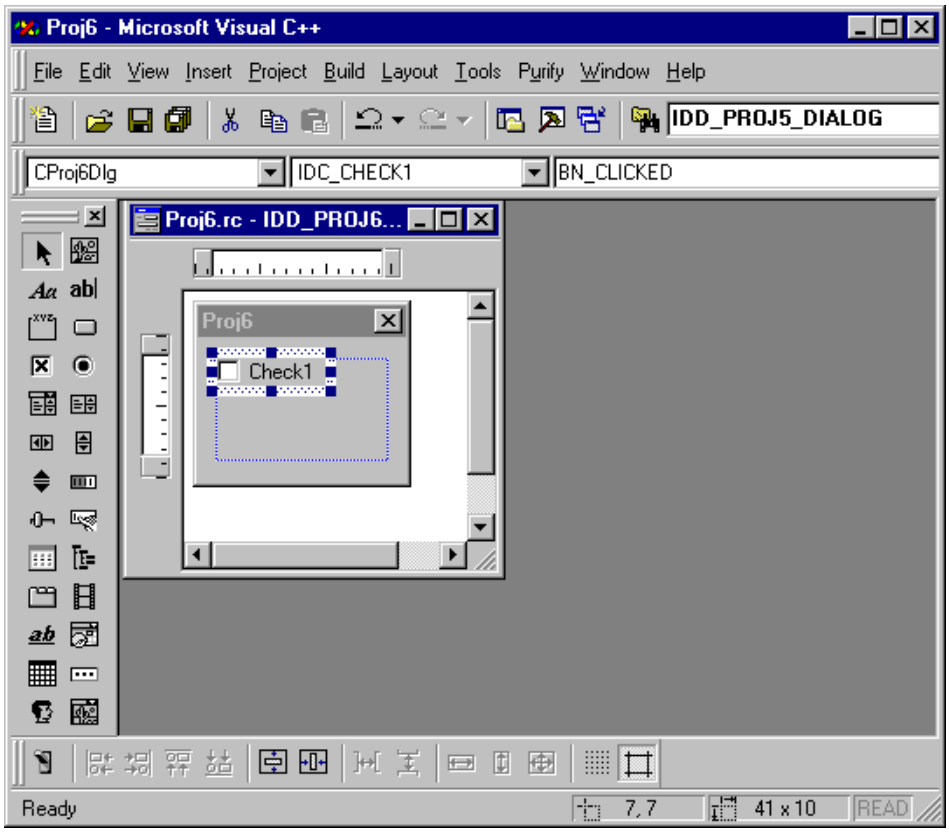
- Click on **Finish** on the window below after accepting the default class name and **OK** on the window that will appear after that.:



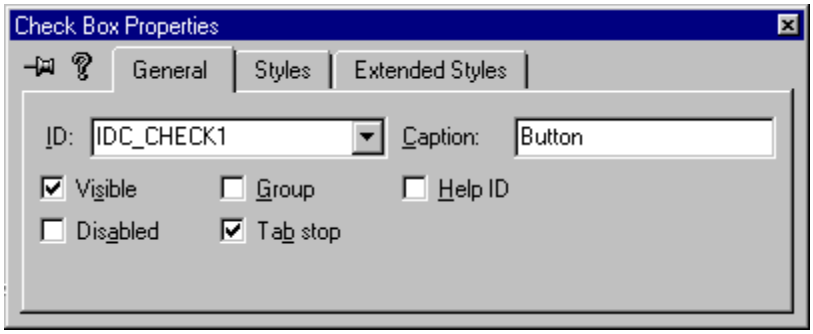
- • If we look at the image below, we can see what the project should look like.

- • We will go into the dialog box and delete the **OK** button and the **Cancel** button. All you need to do is to left click on the buttons and press the delete button on your keyboard. We then want to click on the **TODO** caption and delete it as well. We will want to resize the dialog window to something sufficient for one button as seen below:

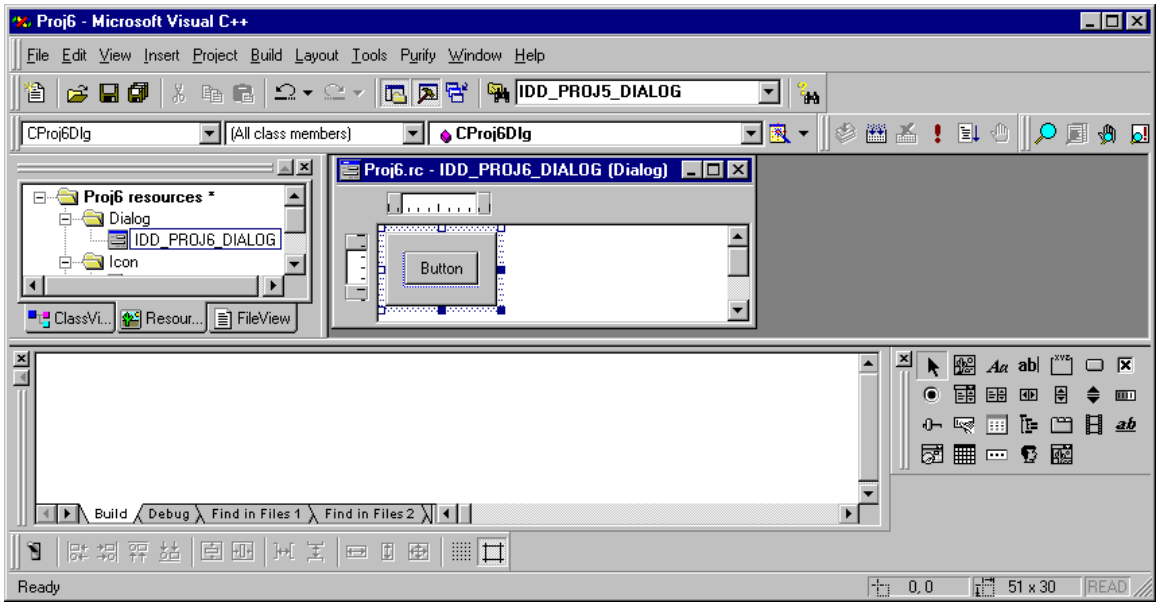
- • Now we will add a check-box. Click on the check box and drag it onto the dialog box. You will see the following below:

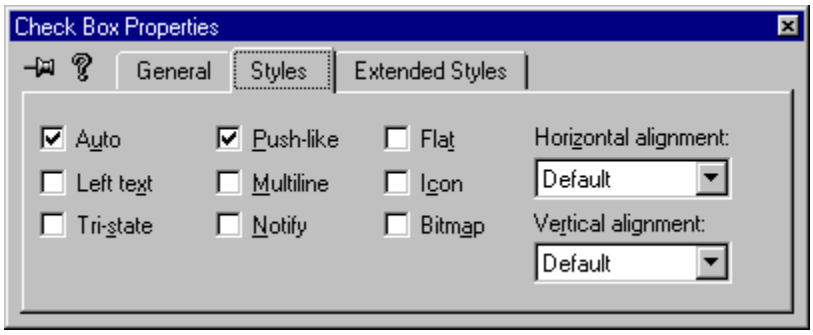


- Right click on the checkbox and select the **Properties**. You will see the windows below and change the caption to **Button**.

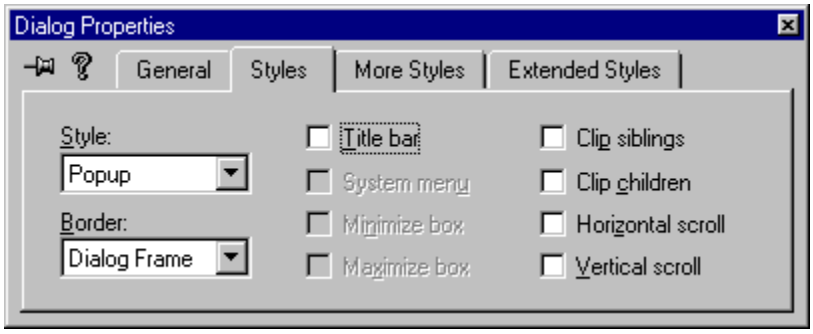


- In the **Style** tab, select **Push-like** check box.

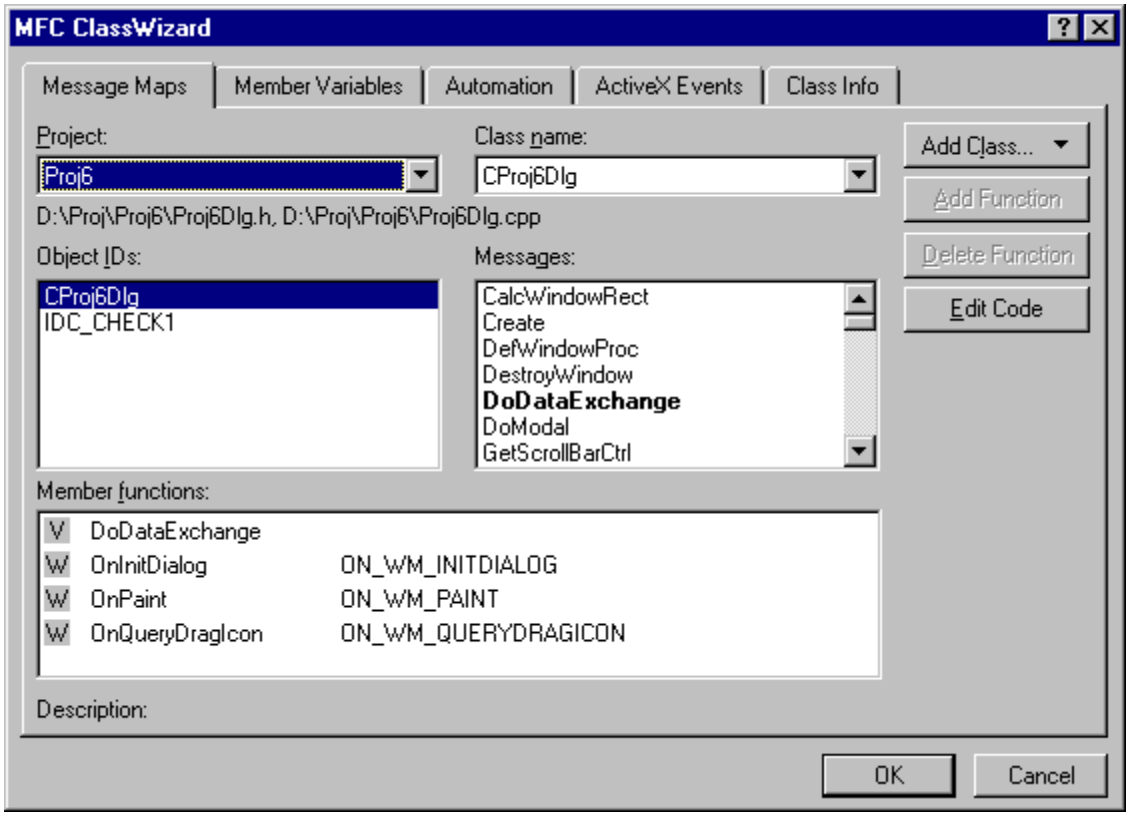




- In the Extended Styles, select the options that you see fit.
- Then right click on the dialog box and select **Properties**. In **Style**, uncheck the **title bar** and close that window:



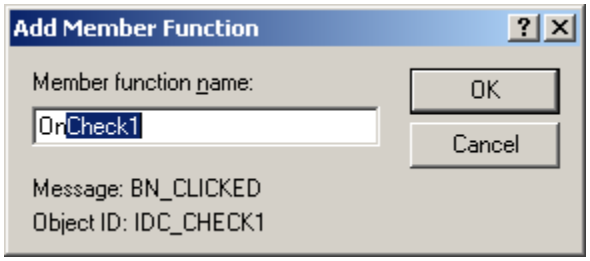
- You will see now have the following project:



- Build your project by going into **Build/Build Proj6.exe** or press **F7**. Your project should build without errors.
- Now, you can write click on the dialog box window and select **ClassWizard**. You will 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 double-click on **BN_CLICKED** in the **Messages**. This will have the same effect as adding a function to the message event **BN_CLICKED**.

- You will see the window below:

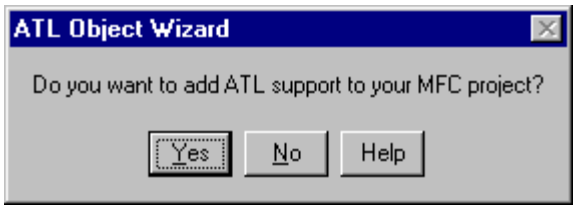


- Click on **OK** after allowing the default name to be chosen.
  Now you have allowed the wizard to add the code to your project. Click on **OK** on **the MFC ClassWizard** window.

1.2 1.2 Now, we will add the ATL environment:

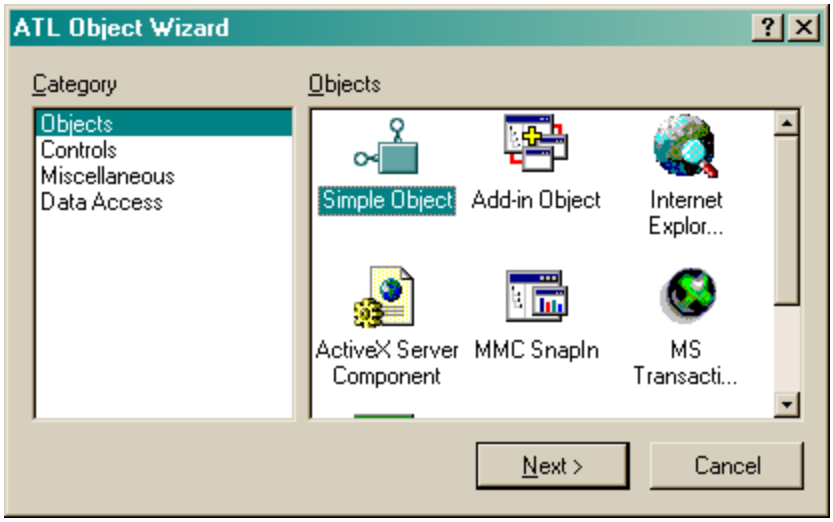
Note: Due to a known problem with **MFC**, it is possible that this step may be giving you some problems. In order to avoid those problems, we will need to clean up the files a little bit and proceed from there. If the process of adding the **ATL object** crashes, by cleaning it up once more, the second attempt is usually successful.

- We will want to:
  - Close the **MSVC Studio** after saving the project workspace.
  - 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**.

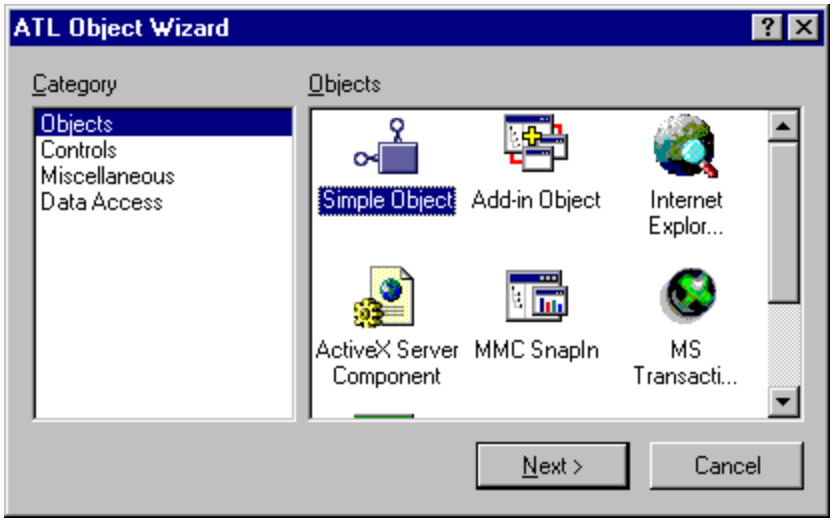


  - Click on **Yes**
  - You may see the following window:

  - Click on **OK**. If you're lucky, you'll see the **ATL Object Wizard** window displayed below. If you do, proceed from there, otherwise, follow the next instruction.
  - 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:

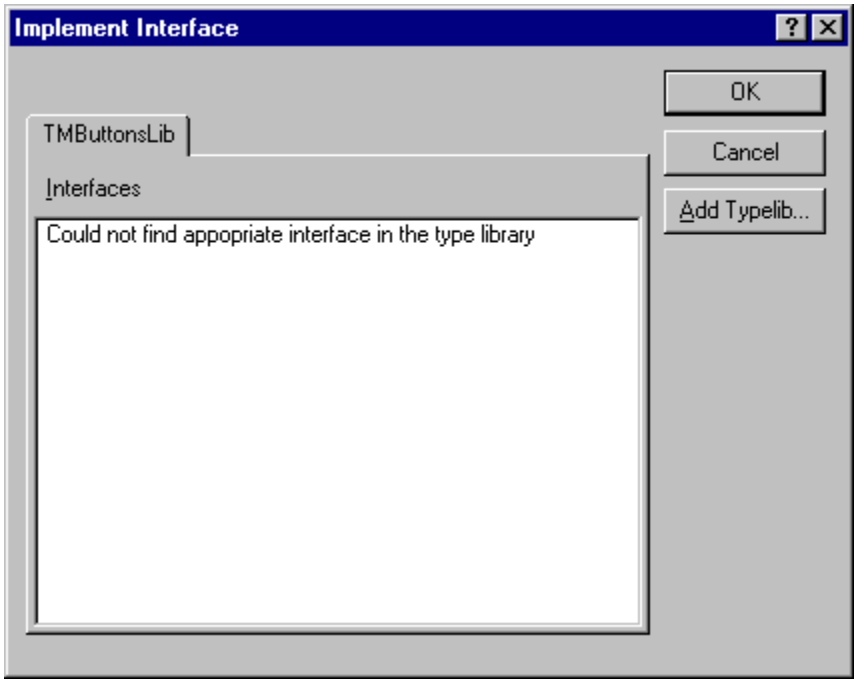


- From the Studio, select **insert/New ATL Object…** Select **Yes** on the following window:
- Select **Simple Object** as seen below and click on **Next**.

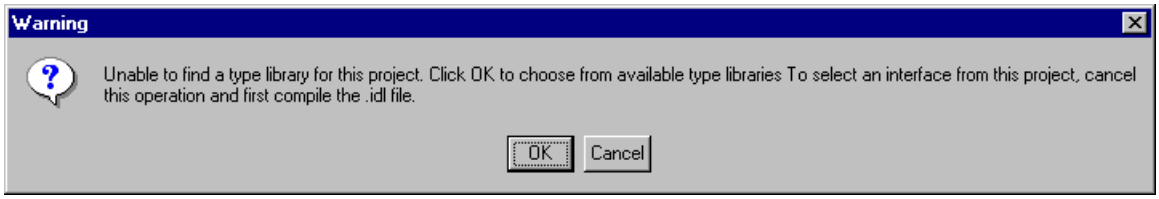


- You will then see the following window. In the **Short name:** box, enter the name of the ATL object that you wish to create. The other boxes will fill in by themselves. In this case, we chose the name **ButtonRes**. Don't chose the name **IOResource** because it's already used by the system and it will prevent you from building the COM exe.
  Also, take note of the name in **Prog ID**: In this case **Proj6.ButtonRes** (exe name followed by the short name). This will be used as an entry in the Wrts options.

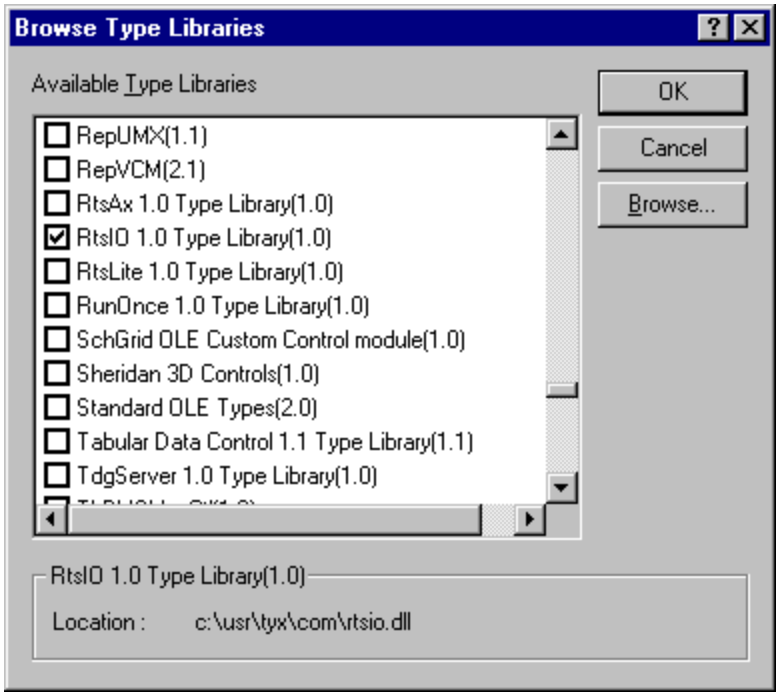
- In the list of attributes, as shown below, all defaults can be acceptable. You may want to add the option of **Support IsupportErrorInfo**.
  These options make more sense for those that are familiar with COM.



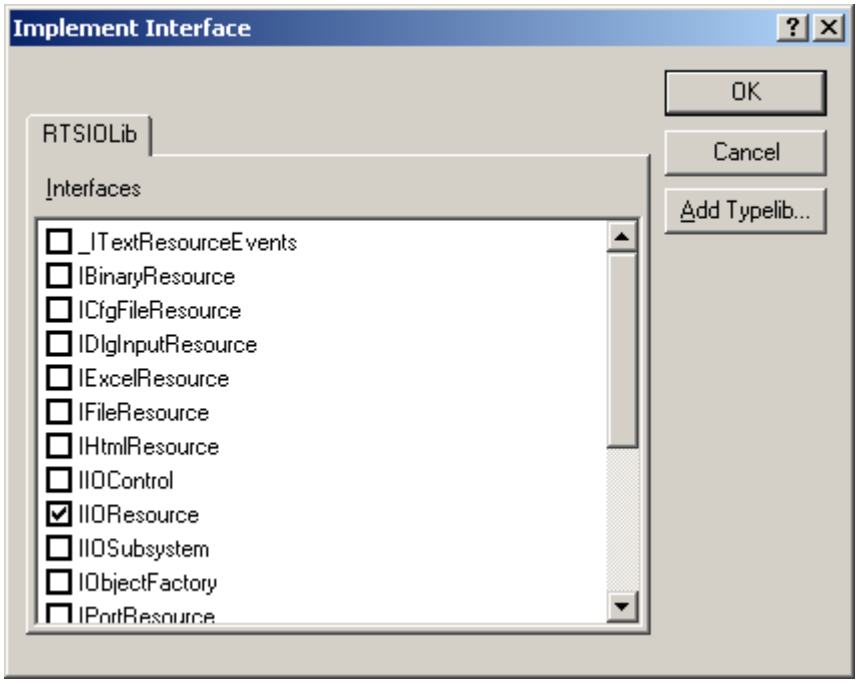
- Click on **OK**.
- Now we are back to the Studio.
- Now you need to link the project to an interface, which in this case relates to the Wrts capabilities. In the workspace, you should right-click on the Class that starts with **C<short name**>, where short name is from the **ATL Object Wizard Properties**. In our case **CButtonRes** as seen below.
- After right-clicking on **CbuttonRes**, 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 **Add Typelib…**
- 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 Studio that is older than **1.10.x**.

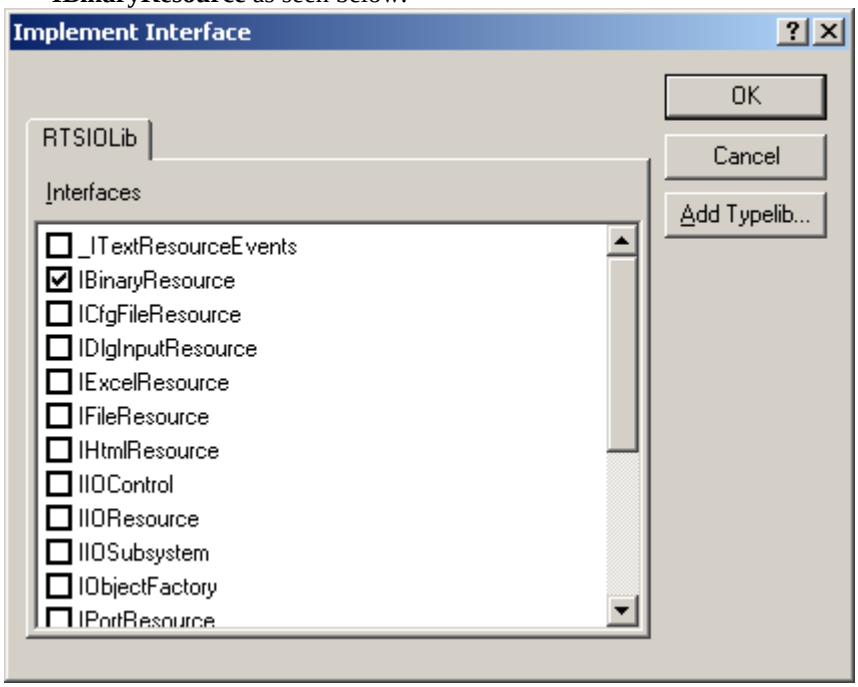


- Make sure that the **rtsio.dll** is located in the location described at the bottom of the window above.
- In the following window, you need to check **IIOResouce** and then click on **OK**.

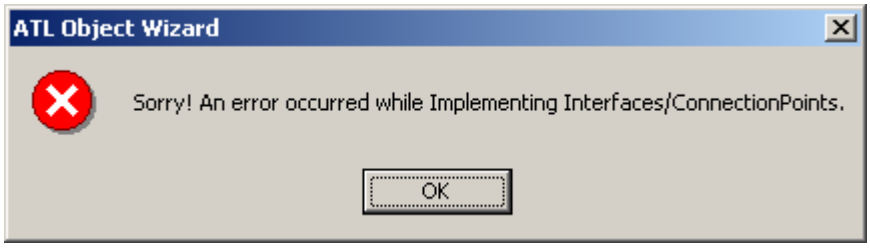


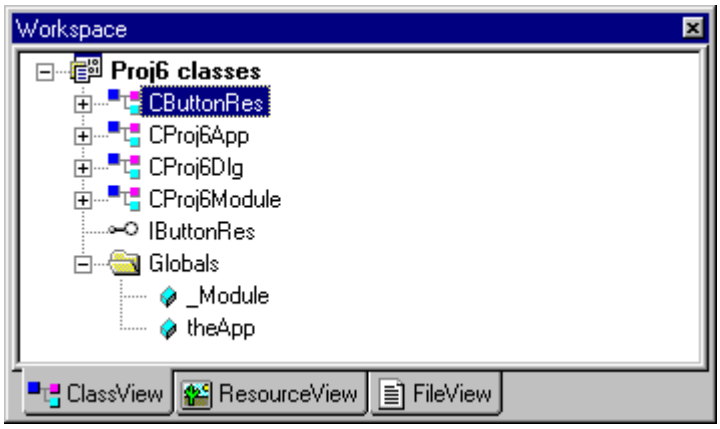
- Repeat the interface implementation for **IBinaryResource** (for text IO resources). For this, go back to right-clicking

on the **CButtonRes** from **ClassView** and rather than selecting **IIOResource** in the **Implement Interface** window, select **IBinaryResource** as seen below.



- In the event that you see this window below, clean the project as we've seen above, reopen the **dsw** project and make follow the instruction below. You will want to make sure that the code is the same as the one listed below.





- Now you will need to modify the source of what has been made available to you. In our case, double click on **CbuttonRes**, or **C<short name>** as shown below:

```
// ButtonRes.h : Declaration of the CButtonRes

#ifndef __BUTTONRES_H_
#define __BUTTONRES_H_

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

/////////////////////////////////////////////////////////////////////////////
// CButtonRes
class ATL_NO_VTABLE CButtonRes :
        public CComObjectRootEx<CComSingleThreadModel>,
        public CComCoClass<CButtonRes, &CLSID_ButtonRes>,
        public ISupportErrorInfo,
        public IDispatchImpl<IButtonRes, &IID_IButtonRes, &LIBID_Proj6Lib>,
        public IBinaryResource,
        public IIOResource
{
public:
        CButtonRes()
        {
        }

DECLARE_REGISTRY_RESOURCEID(IDR_BUTTONRES)

DECLARE_PROTECT_FINAL_CONSTRUCT()

BEGIN_COM_MAP(CButtonRes)
        COM_INTERFACE_ENTRY(IButtonRes)
        COM_INTERFACE_ENTRY(IDispatch)
        COM_INTERFACE_ENTRY(ISupportErrorInfo)
        COM_INTERFACE_ENTRY(IBinaryResource)
        COM_INTERFACE_ENTRY(IIOResource)
END_COM_MAP()

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

// IButtonRes
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 //__BUTTONRES_H_
```

For complete information on each one of those functions, please refer to the TYX website or the updated online help. You can also look at the **rtsio.dll** with the **OleView** tool that is provided with your compiler.

- 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 program 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 17:

```
public IIOResource
```

in order to avoid compilation problems. Make sure that you also delete the comma at the end of line 16.

- 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
}
```

- 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:

```
#include "Proj6Dlg.h" // Included at the top of file in order to allow the code
                      // beneath to build

// IBinaryResource
```

```
STDMETHOD(Read)(LONG lType, VARIANT * pVal)
{
	CProj6Dlg* pDlg = (CProj6Dlg*)AfxGetMainWnd();		// Added

	if (pVal == NULL)
		return E_POINTER;

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

	return S_OK;						// Changed
}
STDMETHOD(Write)(LONG lType, VARIANT val)
{
	CProj6Dlg* pDlg = (CProj6Dlg*)AfxGetMainWnd();		// Added
	// Set button to required state
	if (val.lVal == 1)
		pDlg->CheckDlgButton(IDC_CHECK1, BST_CHECKED);	// Added
	else if (val.lVal == 0)					// Added
		pDlg->CheckDlgButton(IDC_CHECK1, BST_UNCHECKED);	// Added
	else							// Added
		return E_FAIL;					// Added

	return S_OK;						// Changed
}
```

the first command in **Read** using **AfxGetMainMod** give you a pointer to the current window that consists in the one containing the button. This pointer will then give you access to it's content.
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 pointer to the window the exact same way. The following lines ckeck 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 **ButtonRes.h** file and:
Change the code for the **Open** and **Close** function to the following code:

```
STDMETHOD(Open)(BSTR bstrName, LONG lMode)
{
	CProj6Dlg* pDlg = (CProj6Dlg*)AfxGetMainWnd();	// Added
	if (pDlg) pDlg->ShowWindow(SW_SHOW);		// Added
	return S_OK;					// Changed
}
STDMETHOD(Close)()
{
	CWnd* pDlg = AfxGetMainWnd();			// Added
	if (pDlg) pDlg->ShowWindow(SW_HIDE);		// Added
	return S_OK;					// Changed
}
```

- You are now ready to build the exe with the option **Build/Build**, by using the icon bar, or by pressing **F7**.
- You may register your exe manually or do it from within your project in the **Custom Build** setting.
To register is manually, you need to execute **Proj6 /RegServer** from a command line from the **Proj6.exe** location.
- **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**, **ButtonRes.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(CButtonRes)
    COM_INTERFACE_ENTRY(IButtonRes)
    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(CButtonRes)
    COM_INTERFACE_ENTRY(IButtonRes)
    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 **ButtonRes.h** that will do that.

1.3 1.3 What is wrong and how to fix it?

1.3.1 1.3.1 Registration problems:

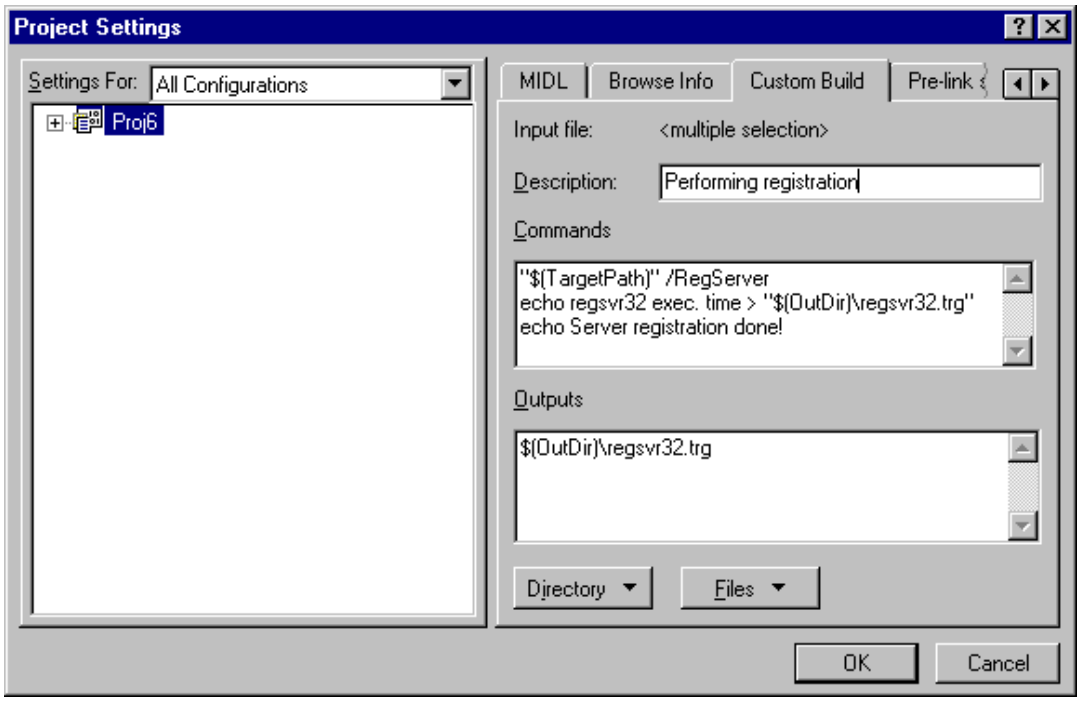
- The first thing that is wrong, is that the COM exe is not registered. The reason are related to the bugs in the MS wizard.
  So, we are going to do manually, what the Wizard does when dealing with an ATL/COM project and not when you start with a MFC project.

- In order to be able to edit the **Proj6.rc**, you will need to search with **Edit/Find in File …** from the menu bar, for the keyword "**registry**".
- Then it in the **Output** window as shown below, double click on a line that includes **Proj6.rc**.

- Click **OK** on the following Window

Add the following lines

```
/////////////////////////////////////////////////////////////////////////////
//
// TYPELIB
```




```
//
1                       TYPELIB MOVEABLE PURE   "Proj6.tlb"
```

at the line 161.

- - Now, for the registration to be done automatically when you build your project, you will need to change some settings in the project.
  From the menu, select **Project/Settings…**
  In the **Custom Build** tab, you need to select **All Configurations** for **Settings For:** and you need then to enter 3 things:

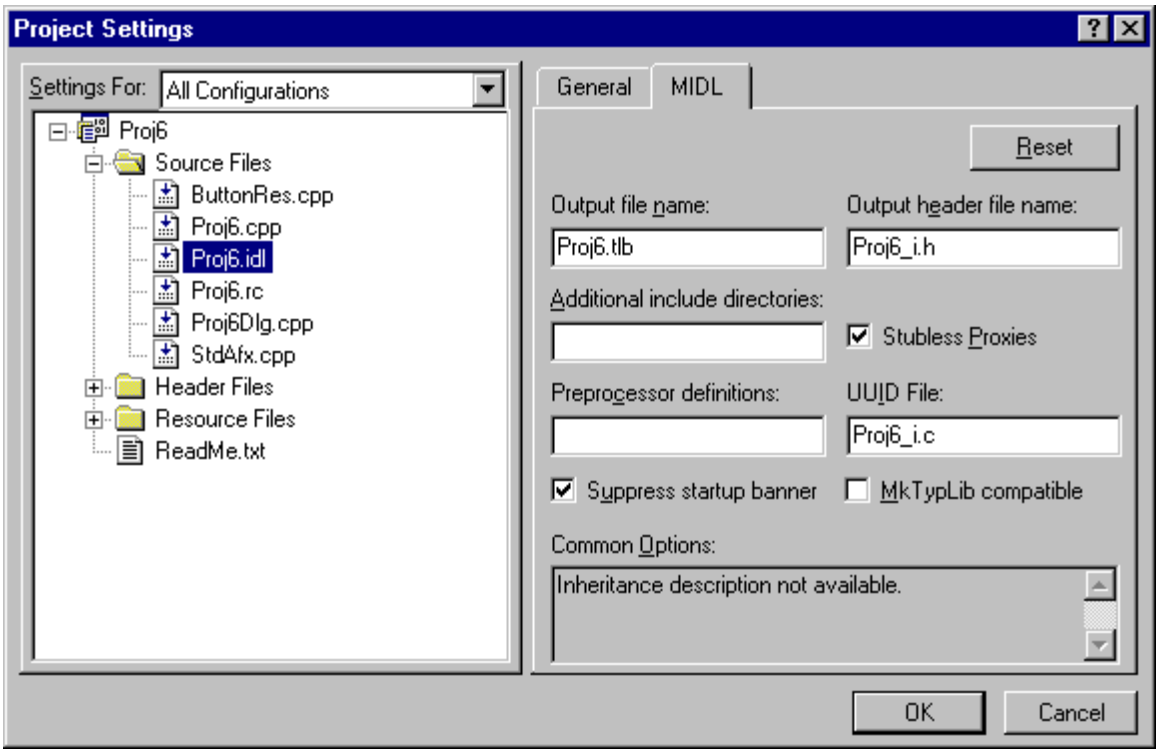
  In **Description**: `Performing registration`

  In **Commands**:
  ```
  "$(TargetPath)" /RegServer
  echo regsvr32 exec. time > "$(OutDir)\regsvr32.trg"
  echo Server registration done!
  ```

  In **Outputs**: `$(OutDir)\regsvr32.trg`

  The picture below will show exactly that.

- - Now, expand **Proj6** in **Settings For:** and then expand **Source Files.** Make sure that you are using **All configurations**.



- Select the **Proj6.idl** file and select the **MIDL** tab as shown below:

- In the **Output file name:** edit box, write Proj6.tlb. Click then on **OK**. You are basically removing the **Debug** and **Release** path to the **Proj6.tlb** file in the debug and release configuration.

1.3.2 1.3.2 Problems with viewing the window when running the Wrts and invoking the Binary resource Proj6.ButtonRes:

As you may realize later, you wouldn't be able to see your Proj6 exe GUI when you run your Atlas program and invoke your IO resource. However, the Binary IO will still work.
This can be fixed in the files that were given to you by the wizards.

In the file **Proj6.cpp**, you need to locate the function **`BOOL CProj6App::InitInstance()`** and in the if condition **`if (cmdInfo.m_bRunEmbedded || cmdInfo.m_bRunAutomated)`**, comment out the return as shown below, in order to avoid breaking out of the function:

```
if (cmdInfo.m_bRunEmbedded || cmdInfo.m_bRunAutomated)
{
        //return TRUE;  // Comment this return
}
```

1.3.3 1.3.3 What if you want to use ActiveX controls?

You will need to add a line in **Proj6.cpp** in the following function **BOOL CProj6App::InitInstance()** as seen below:

```
BOOL Cproj6App::InitInstance()
{
        if (!InitATL())
                return FALSE;

        AfxEnableControlContainer();  // Added
…
```

This command enables just that possibility.

1.3.4 1.3.4 How to move the TMButtons.exe?

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:

Proj6 /RegServer or in any other case **<your exe>/RegServer**

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 your COM exe.

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:

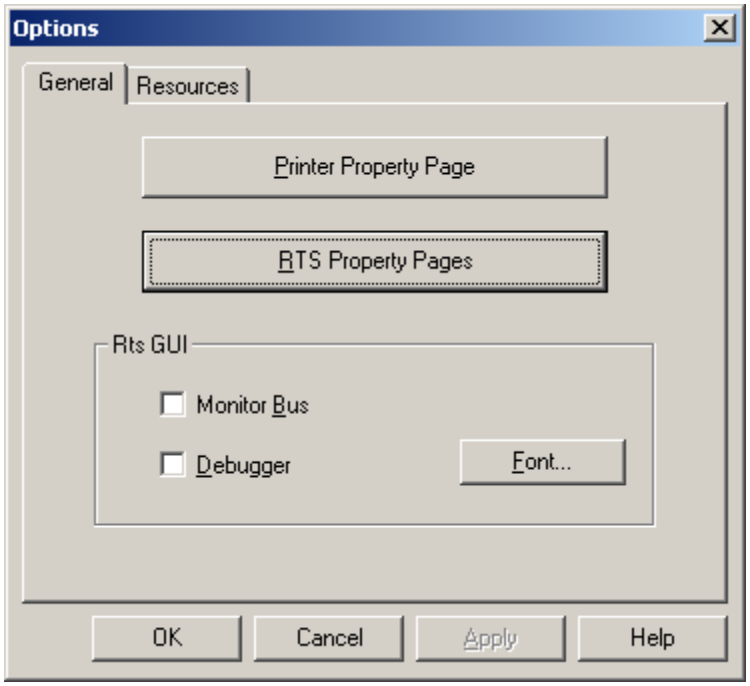
```
 001000 BEGIN, ATLAS PROGRAM 'BUTTONS'                                  $
     10 DECLARE, VARIABLE, 'TXT_DATA' IS STRING(80) OF CHAR             $
     12 DECLARE, VARIABLE, 'OUT' IS FILE OF UNTYPED                     $
C                                                                       $
     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'                            $
```

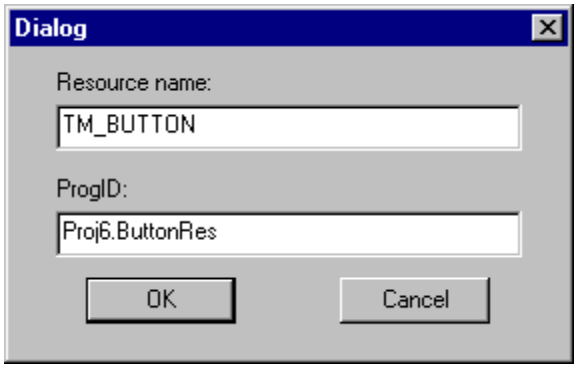
This atlas will simply demonstrate two things:
It will output updated status of the button from checked to unchecked and vice-versa, and it will retrieve the status of the button in the Atlas via a Binary IO resource. The resource will be visible after is has been enabled and will close when it is disabled.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

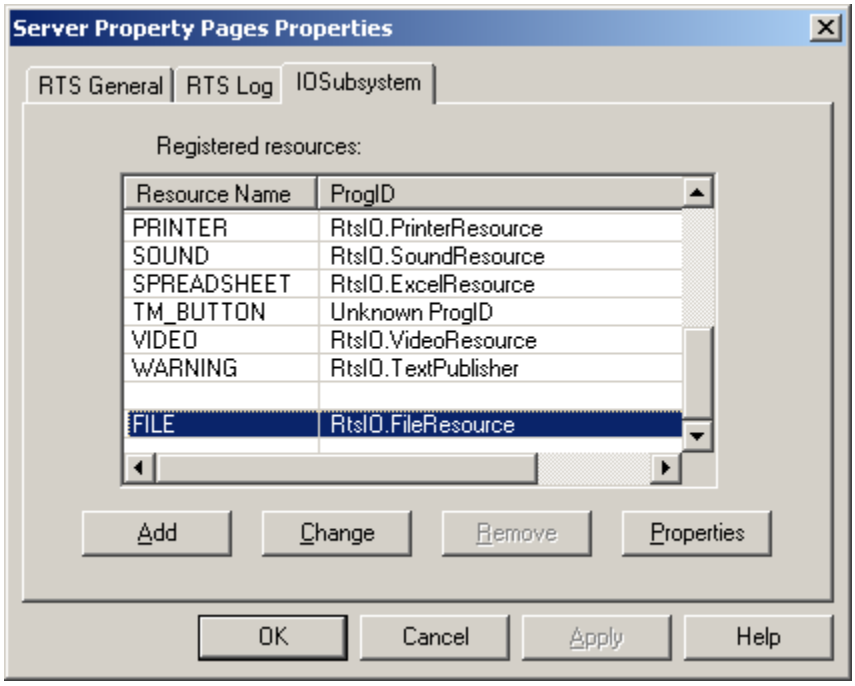
2.3 2.3 Wrts Settings:

- 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





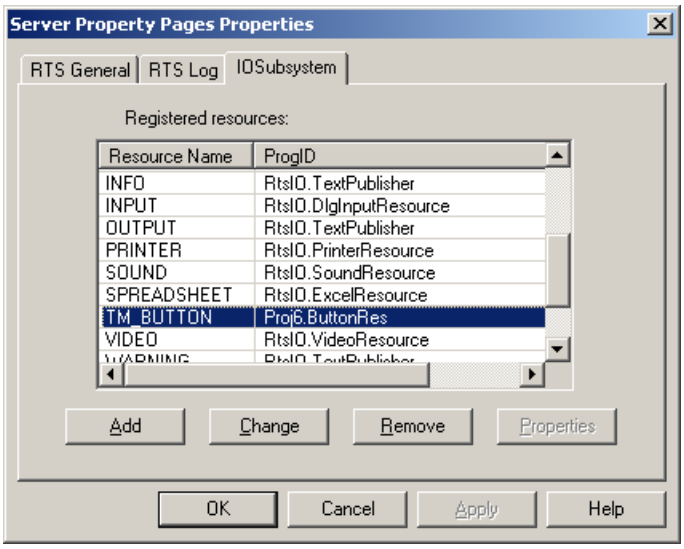
- • You now need to create the link between your Atlas IO resource and the COM resource that you wish to link to. Go into the list of Resource Names. If **TM_BUTTON** is already there with an unknown **ProgID**, remove it by selecting it and by pressing **Remove**.
- • Press **Add** and you will see 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 **Proj6.ButtonRes**:

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



- If this step fails, you need to make sure that the I\O resource exe (**Proj6** in this case) is registered before trying again. Your Wrts is now ready to link your Atlas IO resource to your COM exe.

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 exe will be invoked when necessary. The location of the COM exe does not matter because it has been registered in your system.

3 3 How to debug the Com exe?

- First, you put the breakpoints where you want them in your exe COM source code. You would now run the COM exe 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 exe that you have running in debug mode. The execution of the Wrts would stop at your breakpoints in the COM exe source.