

TYX CORPORATION

Productivity Enhancement Systems



Reference	TYX_0051_8
Revision	1.1b
Document	PawsIOHowTo.doc
Date	October 9, 2003

Text I/O exe resource
without MFC tutorial

This document will help with making the Text input-output interface without 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 version is not guaranteed.

Introduction:

First we will address the issue of what to do with the exe that we wish to link to the Atlas with a Text IO. Other documents will address the issue of using the support of Binary IO and the usage of GUI within the COM environment. Then we will see the Atlas and what it takes with a 716.89 environment to make use of it.

Dll versus Exe:

1. 1. Advantage of Dlls versus Exe:

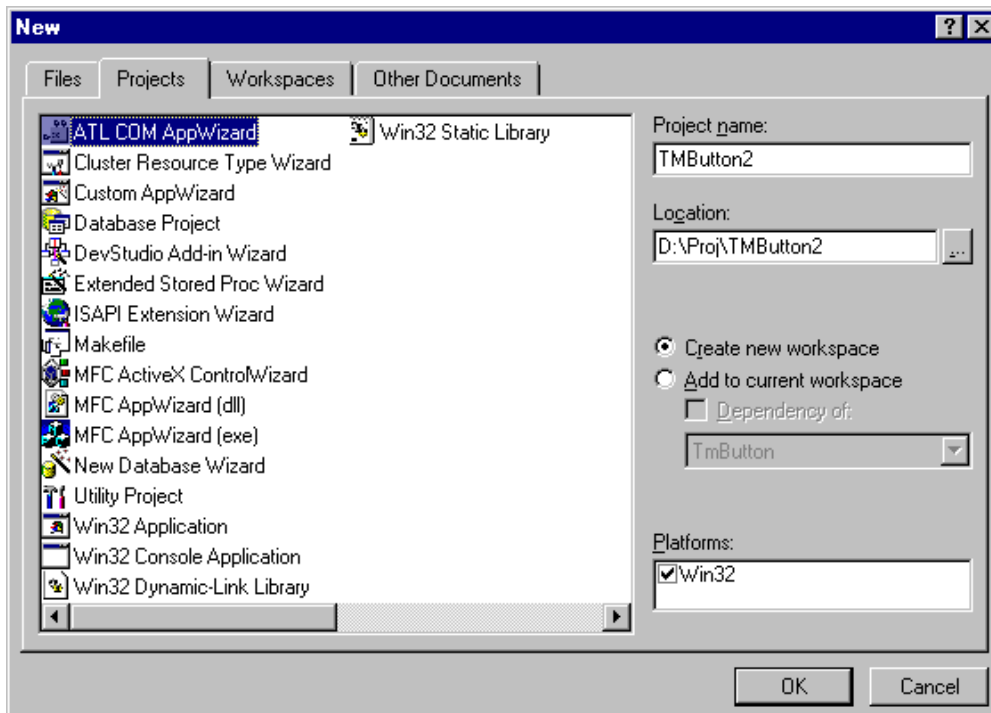
- • The Dll uses fewer resources than the Exe.
- • The Dll can easily remain on top of the Wrts.
- • The accelerators are directly passed on to the Wrts without additional code. When the Exe has the focus, it will need some code in order to pass on the accelerator destined for the Wrts.
- • The message 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:

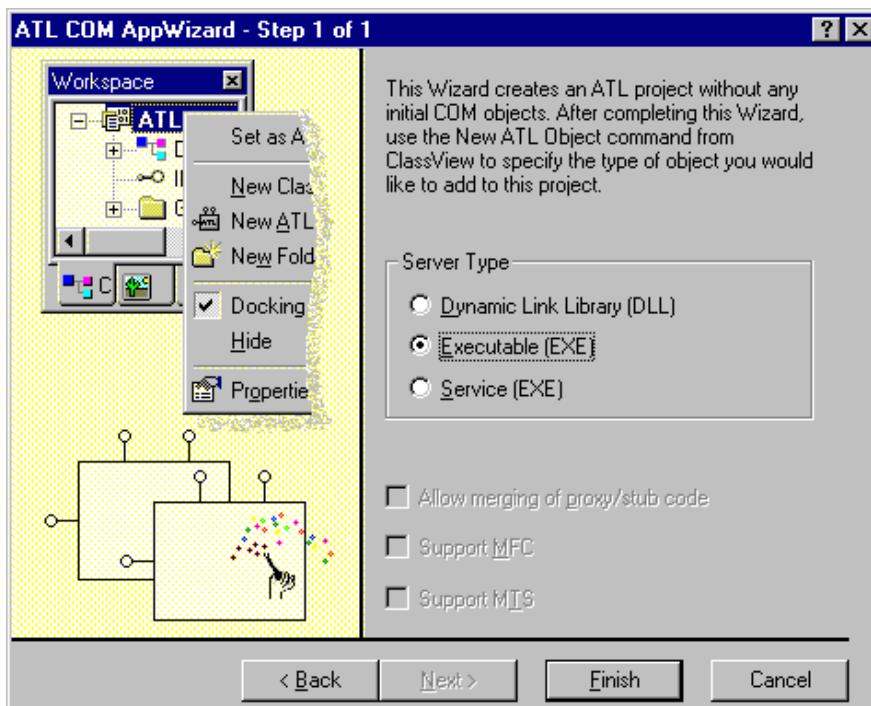
- • The Exe is more isolated than the Dll and if it crashes, it will not affect the Wrts. This may be an issue if the Exe links to code that is not stable.
- • The Exe can be moved in front or behind the Wrts at will.
- • The Exe can be run remotely.

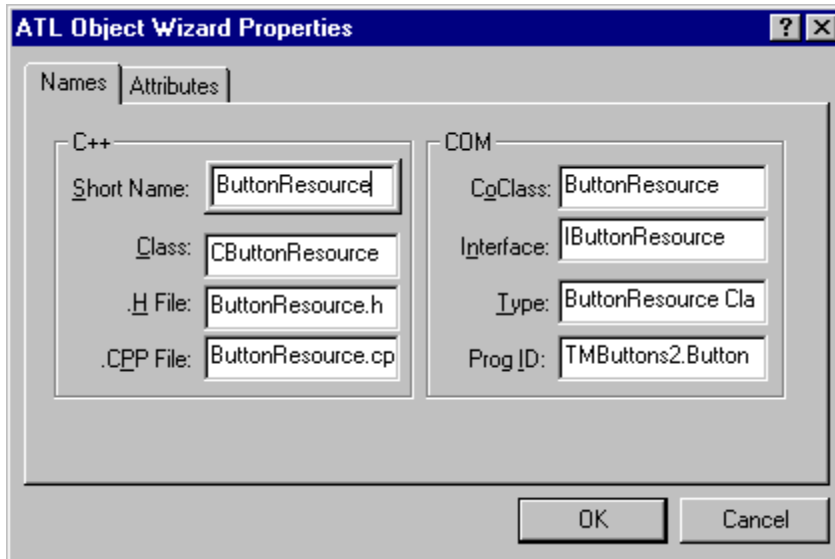
1 1 *When generating a COM .exe with MSVC 6++*

- • From **File/New...**, select an **ATL COM AppWizard** and select a project name as seen below:



- • Click on **OK**
- • Then select **Executable (EXE)**

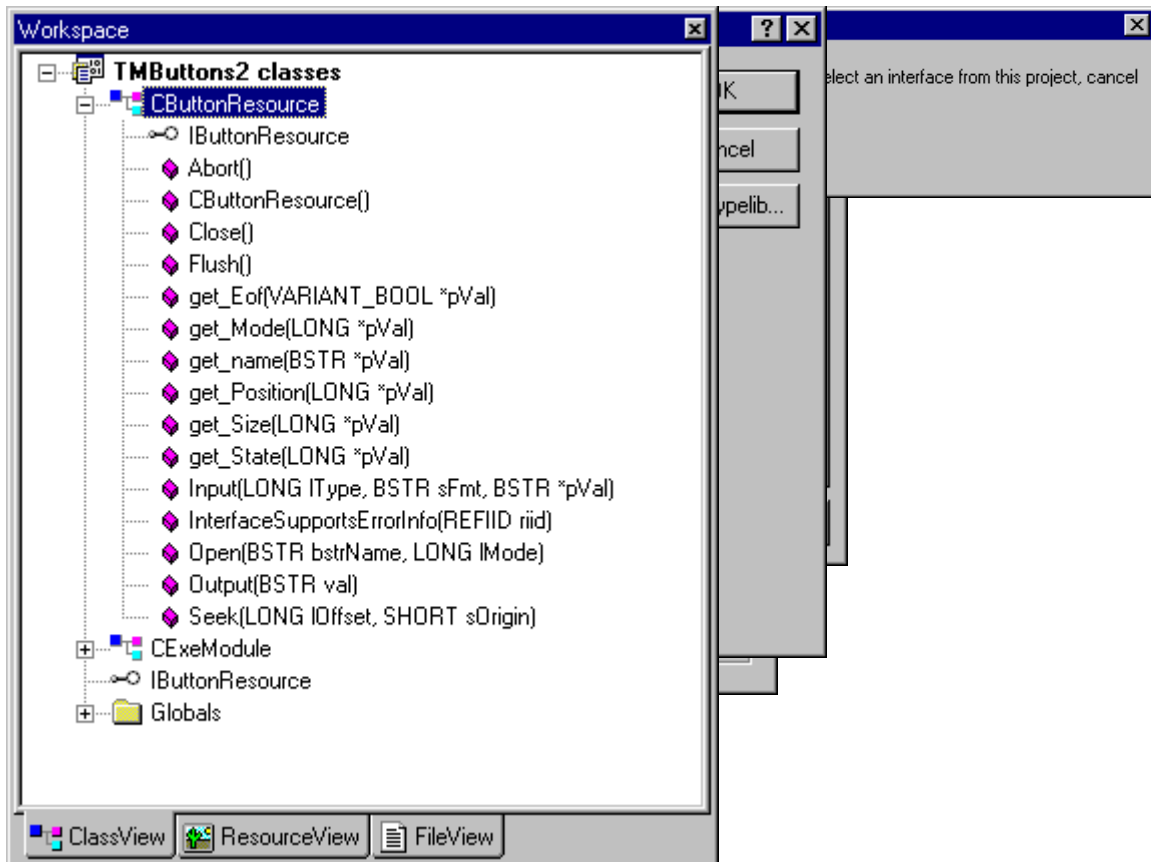




- • Click on **Finish** on this window, and **OK** on the window that will appear after that.
- • From the Studio, select **insert/New ATL Object...** Select **Simple Object** as seen below and click on **Next**.



- • 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 **ButtonResource**. 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 **TMBUTTONS2.ButtonResource** (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 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 **CButtonResource** as seen below.
- • After right clicking on **CbuttonResource**, Select **Implement interface...**
- • You will get the following message and then click on **OK**.
- • 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.
- • In the following window, you need to check **IIOResource** and **ItextResource** (for text IO resources) and then click on **OK**.
- • Now you will need to modify the source of what has been made available to you. In our case, double click on **CButtonResource** under ClassView, or **C<short name>** as shown below:

- • This will give you the following text:

```
// ButtonResource.h : Declaration of the CButtonResource

#ifndef __BUTTONRESOURCE_H_
#define __BUTTONRESOURCE_H_

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

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// CButtonResource
class ATL_NO_VTABLE CButtonResource :
public CComObjectRootEx<CComSingleThreadModel>,
public CComCoClass<CButtonResource, &CLSID_ButtonResource>,
public ISupportErrorInfo,
public IDispatchImpl<IButtonResource, &IID_IButtonResource, &LIBID_TMBUTTONS2Lib>,
public IIOResource,
public ITextResource
{
public:
    CButtonResource()
    {
    }

DECLARE_REGISTRY_RESOURCEID(IDR_BUTTONRESOURCE)

DECLARE_PROTECT_FINAL_CONSTRUCT()

BEGIN_COM_MAP(CButtonResource)
    COM_INTERFACE_ENTRY(IButtonResource)
    COM_INTERFACE_ENTRY(IDispatch)
    COM_INTERFACE_ENTRY(ISupportErrorInfo)
    COM_INTERFACE_ENTRY(IIOResource)
    COM_INTERFACE_ENTRY(ITextResource)
END_COM_MAP()

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

// IButtonResource
public:

// 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;
    }
}
// ITextResource
STDMETHOD(Input)(LONG lType, BSTR sFmt, BSTR * pVal)
{
    if (pVal == NULL)
        return E_POINTER;

    return E_NOTIMPL;
}
STDMETHOD(Output)(BSTR val)
{
    return E_NOTIMPL;
}
};

#endif // __BUTTONRESOURCE_H_

```

- • 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 **16**:

```
public HRESULT Resource,
```

in order to avoid compilation problems.

- 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;
}
```

- The code below is where the text 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. Typically, you will implement your own code as a function of your own requirements:

```
// ITextResource
STDMETHOD(Input)(LONG lType, BSTR sFmt, BSTR * pVal)
{
    if (pVal == NULL)
        return E_POINTER;

    *pVal = ::SysAllocString(L"TRUE"); // Added
    return S_OK;                       // Changed
}
STDMETHOD(Output)(BSTR val)
{
    USES_CONVERSION; // This line and line below are added
    ::MessageBox(NULL, OLE2T(val), _T("ButtonResource"), MB_OK | MB_INFORMATION);
    return S_OK;     // Changed
}
```

You can note that in this case, the return of the input is going to be a string containing “TRUE”. Your code there would determine the exact content of that string.

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

- You are now ready to build the COM 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 **TMButton2 /RegServer** from a command line from the **TMButton2.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 **IOResource** 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(IOResource)
END_COM_MAP()
```

If you had another resource such as text, you will have to tell **IOResource** 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.

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.

2.1.1 Sample for Atlas 416:

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. So, there is no 416 sample code for this tutorial.

2.2.2 Sample for Atlas 716-89/95:

```
001000 BEGIN, ATLAS PROGRAM 'TM_BUTTONS' $
C $
    10 DECLARE, VARIABLE, 'TMSTATUS' IS STRING (80) OF CHAR $
    20 DECLARE, VARIABLE, 'OUT' IS FILE OF TEXT $
C $
E010000 OUTPUT, C'\LF\ \HT\START OF MMEDIA SAMPLE PROGRAM\LF\' $
C $
C***** $
C $
    10 ENABLE, OUTPUT TO NEW C'TM_BUTTON', VIA 'OUT' $
    20 OUTPUT, TO 'OUT', C'Text Stream123'$ $
    30 DISABLE, 'OUT' $
C $
    50 ENABLE, INPUT FROM C'TM_BUTTON', VIA 'OUT' $
    60 INPUT, FROM 'OUT', INTO 'TMSTATUS' $
    70 DISABLE, 'OUT' $
    80 OUTPUT, C'TMSTATUS =', 'TMSTATUS' $
C $
999998 OUTPUT, C'THE END' $
999999 TERMINATE, ATLAS PROGRAM 'TM_BUTTONS' $
```

This atlas will simply do two things:

It will output a message box with the content “**TMSTATUS**” and it will input a string that has been hard-coded in the COM exe as “**TRUE**”. That string that has been retrieved will be displayed in the WRTS.

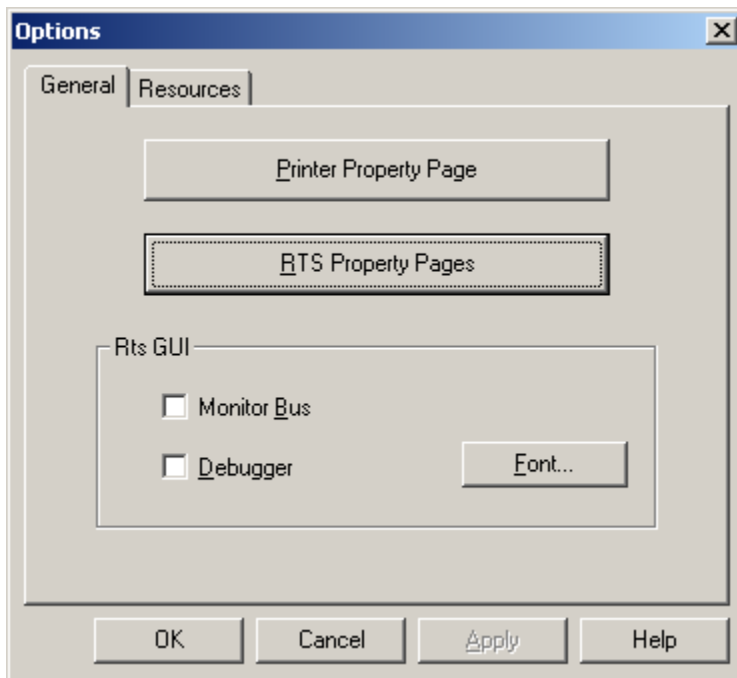
You can note that for **IEEE716.89/95** Atlas, you may use the same **COM** resource for input and output which is not necessarily the case for **IEEE416** (not all revisions in **IEEE416** allow the usage of **INPUT** and **OUTPUT**) and **IEEE716**.

You will need to compile this Atlas and have it ready to run.

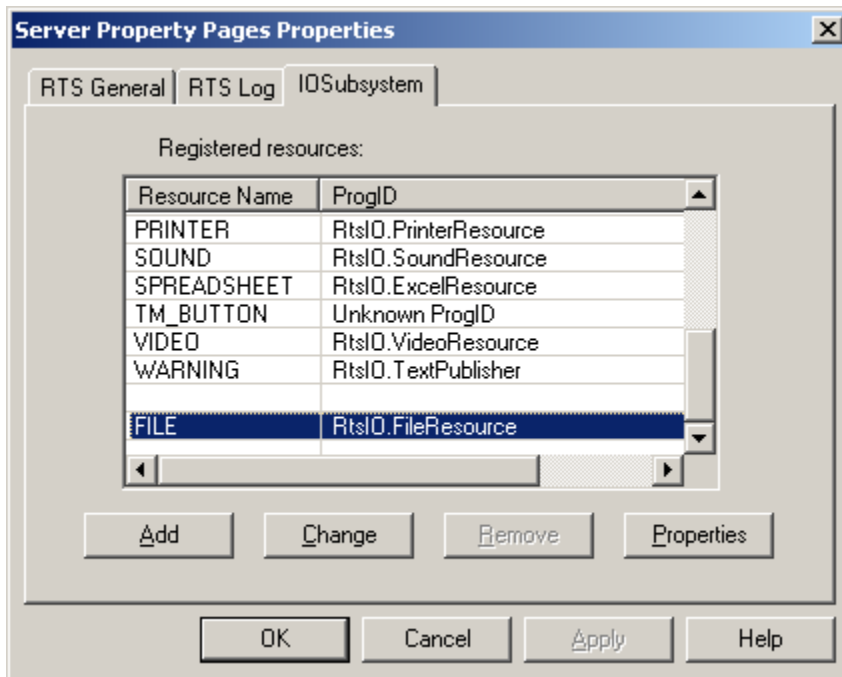
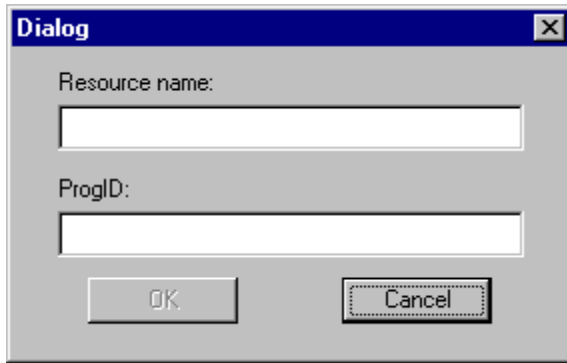
Note: In your Atlas code, on line **010020**, if you leave a space between “ ‘ ” and “\$”, you will enter the text output resource twice, and the second time, with no parameter.

2.3 2.3 Wrts Settings:

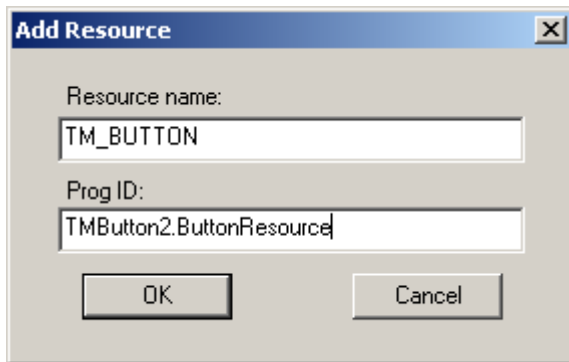
- • Now that Atlas is ready to run, you should launch the Wrts.
- • Go into **Control/Options...**



- • Now Click on **RTS Property Pages** and select **IOSubsystem**



- • 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 **TMButtons2.ButtonResource**:



- • Click on **OK** on this window and the one that will appear after that.
- • 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.