

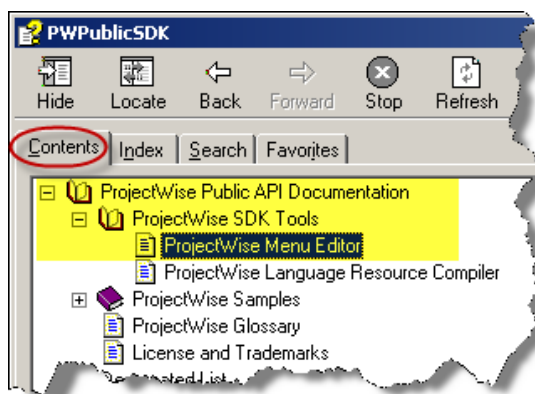
LESSON NAME: CREATING MENU COMMANDS IN PROJECTWISE EXPLORER VIA A MRR FILE

LESSON OBJECTIVE:

You will learn how to create Menu Commands in ProjectWise Explorer using a menu file (*.mrr). The easiest way to create a Menu Command is to use the ProjectWise *Menu Editor* tool. In this exercise we will create a Menu Command that opens a simple dialog box. You can use the same procedure to create Project, Document and other menu-based commands.

> **EXERCISE #1: CREATE THE .MRR FILE**

When ProjectWise Explorer starts, it looks for .mrr files in the ProjectWise “bin” directory. MRR files can define new menus and menu commands or it can be used to change the behavior of existing menu commands. For this exercise, it would be helpful if you open the ProjectWise SDK Help and search for “mrredit” or you can navigate right to it on the “Contents” tab.



1. Launch the Menu Editor.

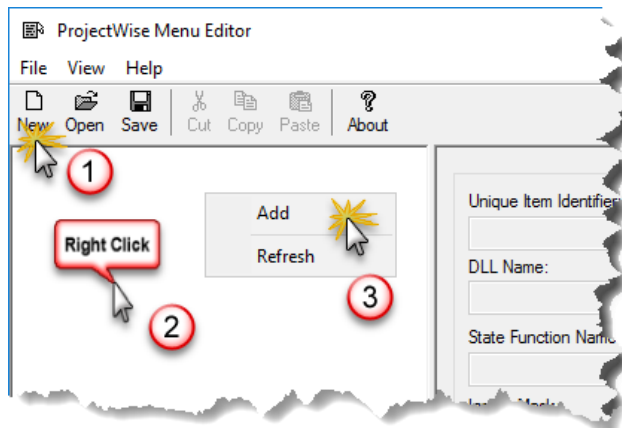
Start > All Apps > Bentley > Menu Editor

2. Create a new.MRR file for this lab.

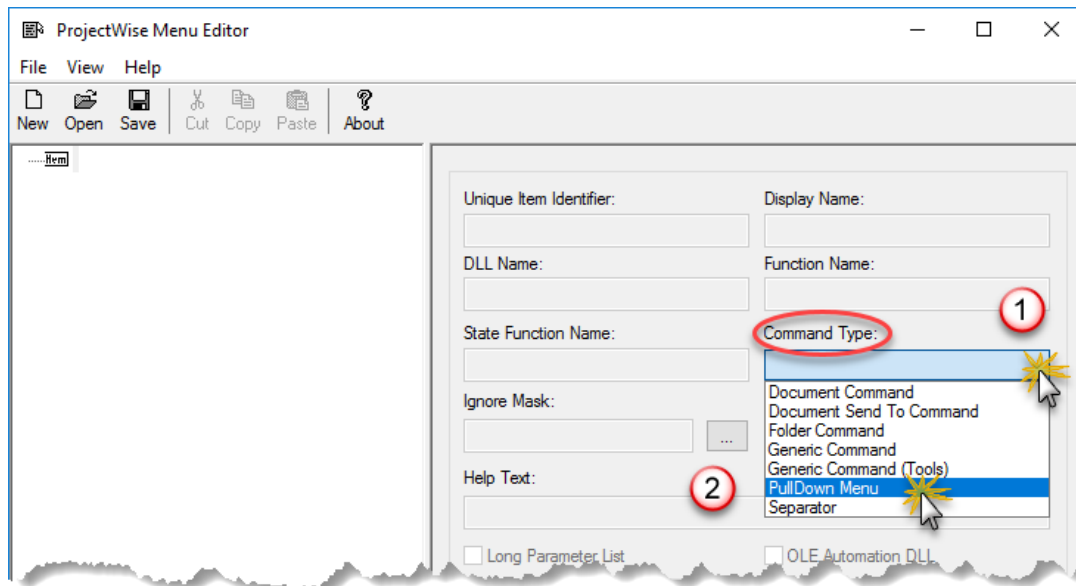
To do so, you must first click on the “New” icon on the toolbar and then right click on the white area to the left of the main panel. You will get a popup menu with two items, “Add” and “Refresh”. Select “Add”.

Creating Menu Commands in ProjectWise Explorer via a MRR File

ProjectWise SDK



3. From the *Command Type* combo box, select “PullDown Menu”.



4. Then key in MY_PDMENU_01 into the *Unique Name Identifier* text box.

Using a naming convention is a good idea, but don't use the same conventions as ProjectWise does as you might inadvertently pick a real menu ID.

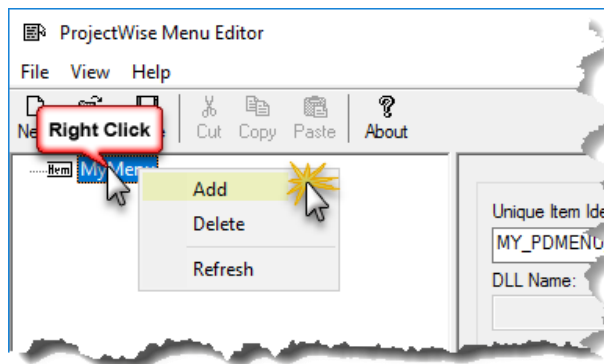
5. Next key in MyMenu into the *Display Name* text box.

You should use **one word** because if any white space is used it will make it look like two pull down menus instead of one. Notice that this appears as the menu name in the left-hand pane of the Menu Editor dialog.

We now need to create a generic menu command under “MyMenu”.

If you just right click in the white area again, you will be creating an item on the same “level” as your pull-down menu. What we want is a menu item UNDER the MyMenu pull down menu.

6. Select “MyMenu” on the left pane by clicking on it once, then right click and then select “Add”.

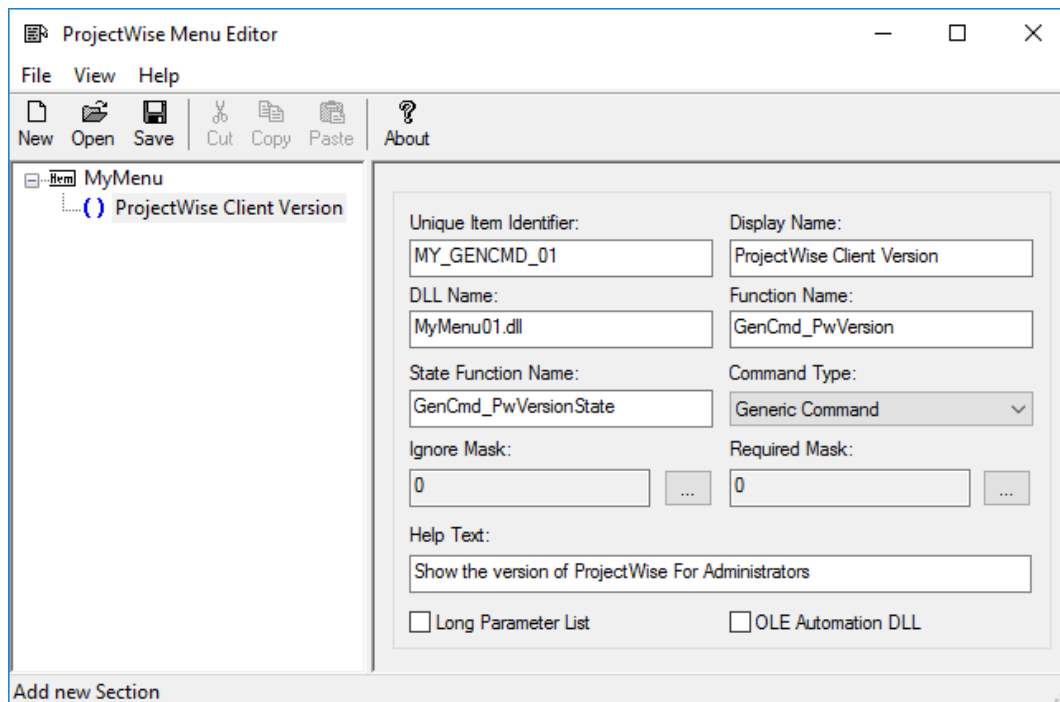


7. Select “Generic Command” from the *Command Type* combo box.

Be careful not to pick the “Generic Command (**Tools**)” command type as that command type will place your menu command on the “Tools” menu in ProjectWise Explorer and not on your custom Pull Down menu where we want it to appear.

8. Fill in the text boxes with the following data:

- Unique Item Identifier: MY_GENCMD_01
- Display Name: ProjectWise Client Version
- DLL Name: MyMenu01.dll
- Function Name: GenCmd_PwVersion
- State Function Name: GenCmd_PwVersionState
- Help Text: Show the version of ProjectWise For Administrators



9. Save the Menu file as “MyMenu01.mrr” in the ProjectWise “bin” directory.

The ProjectWise “bin” directory default location is

“C:\Program Files (x86)\Bentley\ProjectWise\bin”

When you save a menu file with the menu editor, it always closes the file. We will use “MyMenu01.mrr” with the DLL we will be creating next, so open the menu once more with the menu editor so that you can copy and paste the text entries from the menu editor into Visual Studio to avoid simple “typo” mistakes.

> **EXERCISE #2: EDIT THE PROVIDED MyMenu01 VISUAL STUDIO PROJECT**

To use the MRR file, you need to create and code a DLL using Visual Studio. There is already a project created for you. You just need to add the code to the project. Some of the required steps have already been done for you to save time, but we will walk you through the steps so that you will have a reference when you need to create your own MRR customization.

1. Launch Visual Studio 2015 and open the existing project “MyMenu01”.

File > Open > Project/Solution...

Navigate to “C:\PWCESDK\MyMenu01” and open the solution file `MyMenu01.sln`.

We now would need to add two menu functions to this file and we would have to EXPORT these functions so that ProjectWise Explorer will be able “see” them. If ProjectWise cannot “see” the functions declared in the .mrr file, it will not load all of our menu items. ProjectWise Explorer does NOT report any errors when processing.mrr files so you need to make sure that they are correct and properly deployed.

2. Open the source file “MyMenu01.cpp” and go to the end of the file. Make sure that you follow the prototypes documented in the SDK help. (Search for `mrredit` and click on the link to ***mrredit***.)

Here’s what you would normally do at this point:

- Scroll down in the ProjectWise help file under *ProjectWise Menu Editor* until you find **Function prototypes** and look for “CustomGenericCmd”.

Function prototypes

Below there are examples of functions prototypes called by specified menu commands. For all functions the following return values are valid:

Return values:

- `0` The command completed successfully
- `IDCANCEL` The operation was cancelled by the user
- `<0` For added commands, a negative value indicates an error occurred while performing the operation. For a command that replaces a standard ProjectWise menu command, a negative value indicates that the default ProjectWise command was executed.

The prototype for a Generic Command, short parameter list (standard dll and OLE Automation dll):

```
int WINAPI CustomGenericCmd ( void );
```

Typically, you would use the standard dll prototype. In the future, you would copy and paste this prototype into your source file, but this has already been done for you. (Search for the function `GenCmd_PwVersion.`)

- The ProjectWise APIs are “C” based, so you need to declare your function as such and change the name of the prototype function that you copied to match what you have in your .mrr file for *Function Name*, in our case, `GenCmd_PwVersion`. Again, this has already been done for you.

Next you would jump to the top of the `MyMenu01.cpp` file and copy the MFC macro and then paste it into your function. You need to do this for any extern “C” function just in case you use some MFC functionality. This also has already been done for you.

```
extern "C" int WINAPI GenCmd_PwVersion()
{
    AFX_MANAGE_STATE(AfxGetStaticModuleState());
    // Exercise 2, step 11
    // normal function body here
    return 0;
}
```

- Then you would also do the same for the other function, the *State Function Name*, `GenCmd_PwVersionState`. In the ProjectWise SDK help file you will find the prototype for the state function near the bottom of the page that has the other prototypes.

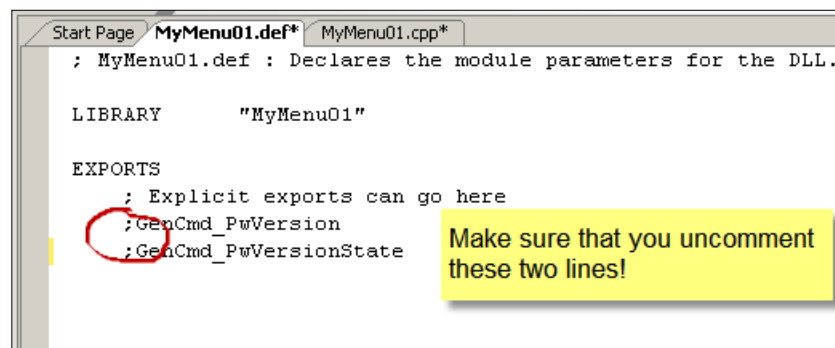
State function prototype

```
int WINAPI CustomStateFunction  
(  
    UINT uiCmdId,  
    ULONG ulState  
);
```

This function is also already in the MyMenu01.cpp file and for now, it just returns the value "AAMS_GRAYED".

```
extern "C" int WINAPI GenCmd_PwVersionState  
(  
    UINT    uiCmdId,  
    ULONG   ulState  
)  
{  
    AFX_MANAGE_STATE(AfxGetStaticModuleState());  
    // Exercise 2, step 10  
    // normal function body here  
    return AAMS_GRAYED;  
}
```

- In order for ProjectWise Explorer to "see" these two functions, they need to be exported. You can do this by adding the functions to the MyMenu01.def file.



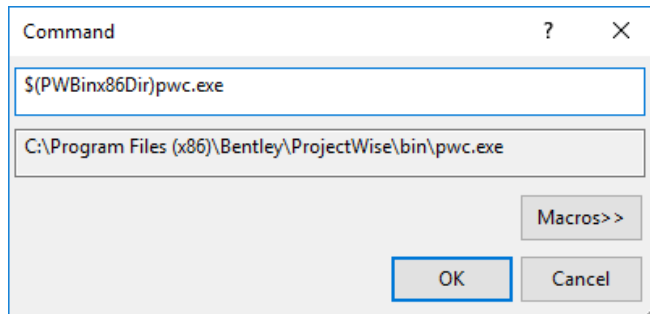
These are already in the .def file, **but you need to uncomment them**. Open the MyMenu01.def file and remove the semicolons from each line.

10. Build and run the code.

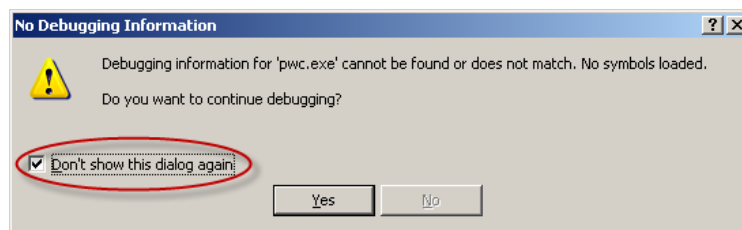
You will need to specify an executable, so use *pwc.exe* located in the ProjectWise "bin" directory.

Creating Menu Commands in ProjectWise Explorer via a MRR File

ProjectWise SDK

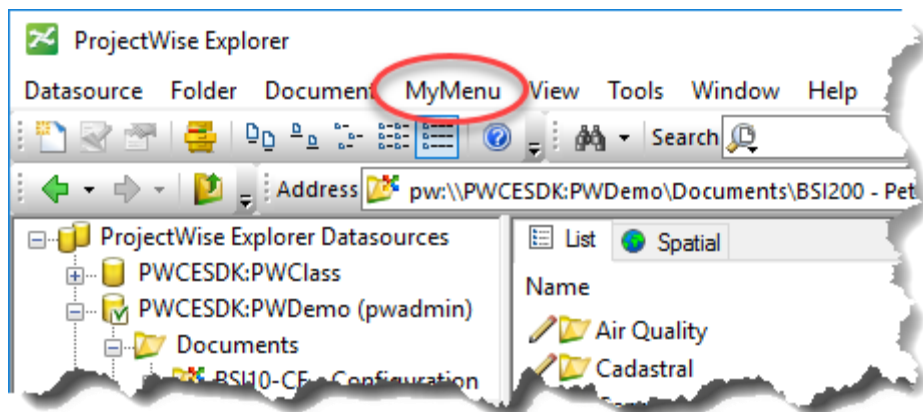


You may get a message about debugging information for the calling program not being available. If so, just check the “Don’t show this dialog again” box and then click on the Yes button. Visual Studio is simply warning you that the program “pwc.exe”, i.e. ProjectWise Explorer, doesn’t have any debug information available, but that’s OK and expected.



Once ProjectWise Explorer starts, you will probably have to change the focus to it manually (it may be behind the Visual Studio window).

11. Log into ProjectWise Explorer and you should see your pull-down menu *MyMenu* on the main menu along the top of ProjectWise Explorer.



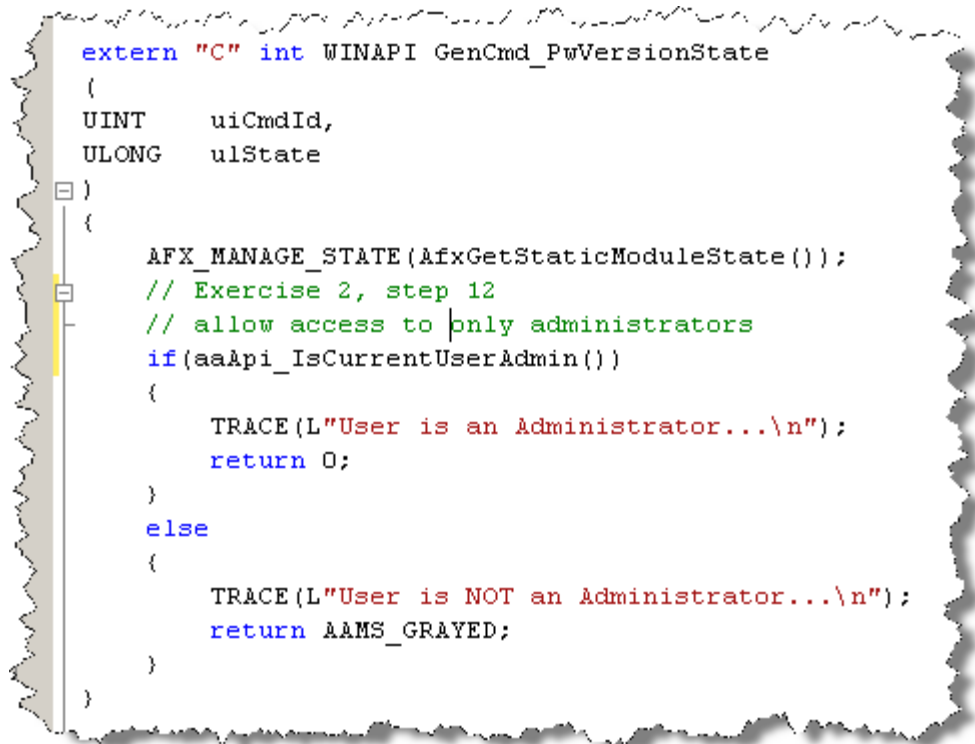
If you do not see your pull-down menu then you need to check all of the steps.

- Make sure that the DLL is in the “bin” directory.
- Make sure that the DLL is named the same as what you used in your menu file.
- Make sure that the two functions are named the exactly same in your code, the .def file and your menu file.
- Make sure that you have exported both functions (placed their names in the .def file).
- Make sure that the menu types are correct. The “MyMenu” is a *PullDown Menu* item, while “ProjectWise Client Version” is a *Generic Command* item, and not a *Generic Command (Tools)* item
- Make sure that you have actually saved the final version of the menu file to disk. If you have not re-opened the menu file, then you haven’t saved it yet.

Right now, the only function that gets called is the “state” function, GenCmd_PwVersionState. Placing a break point in the “state” function can be troublesome as it interrupts the flow of the program. If you really need to debug the state function, using TRACE macros or remote debugging will work better. Exit ProjectWise Explorer.

The next step is for us to add the code for the state function. We only want the menu item to work for administrators. There is a function that you can use to check to see if the current user is an administrator or not -

`aaApi_IsCurrentUserAdmin()`. Here’s what the code should look like:



```
extern "C" int WINAPI GenCmd_PwVersionState
(
    UINT    uiCmdId,
    ULONG    ulState
)
{
    AFX_MANAGE_STATE(AfxGetStaticModuleState());
    // Exercise 2, step 12
    // allow access to only administrators
    if(aaApi_IsCurrentUserAdmin())
    {
        TRACE(L"User is an Administrator...\n");
        return 0;
    }
    else
    {
        TRACE(L"User is NOT an Administrator...\n");
        return AAMS_GRAYED;
    }
}
```

12. Add the code to the `GenCmd_PwVersionState` function to limit access to only ProjectWise Administrators.
13. Build and run the code and see how it behaves for both an administrative user and a regular user.

Remember, breakpoints in the MRR “state” function interrupt the flow of the program and you won’t be able to step through the code as you would expect. If you need to debug a MRR “state” function, use TRACE or some other means that will not disrupt ProjectWise’s GUI events.

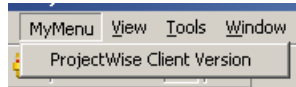
14. Add the following code to the `GenCmd_PwVersion` function so we can get the version information displayed in a dialog box:

```
extern "C" int WINAPI GenCmd_PwVersion()
{
    AFX_MANAGE_STATE(AfxGetStaticModuleState());
    // Exercise 2, step 14
    CString myStr;

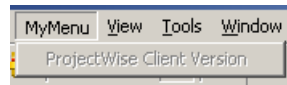
    // determine the version of the SDK/client
    LONG    lMajorVersionMS = 0;
    LONG    lMajorVersionLS = 0;
    LONG    lMinorVersionMS = 0;
    LONG    lMinorVersionLS = 0;
    if(!aaApi_GetAPIVersion(&lMajorVersionMS,
        &lMajorVersionLS, &lMinorVersionMS, &lMinorVersionLS))
    {
        return AAMS_UNDEFINED; // indicates error
    }
    else
    {
        myStr.Format(L"API Version: %ld.%ld.%ld.%ld\n",
            lMajorVersionMS,
            lMajorVersionLS,
            lMinorVersionMS,
            lMinorVersionLS);
        aaApi_MessageBox(myStr, MB_OK);
        return 0;
    }
}
```

15. Build and run the code and see how it behaves for both an administrative user and a regular user.

Breakpoints are OK in this function of course. The menu command should now be enabled if the current user is a member of the Administrator group and disabled if they are not:



Administrative User



Regular User

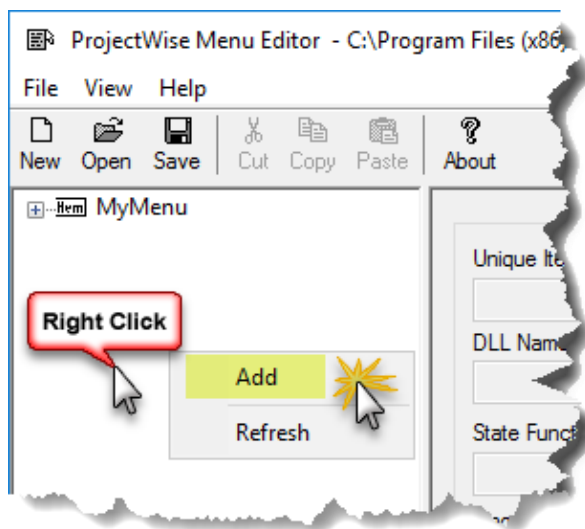
> **EXERCISE #3: DISABLING MENU COMMANDS IN PROJECTWISE EXPLORER**

You will learn how to disable Menu Commands in ProjectWise Explorer using a menu file (*.mrr) in general and the Delete Document Command specifically.

The easiest way to disable a Document Menu Command is to use ProjectWise's Menu Editor tool to intercept the document command when it is about to appear on the Document Menu and disable it. We will use the same MRR file as the previous exercises.

1. Launch the ProjectWise *Menu Editor* and open the previous .mrr file "MyMenu01.mrr".
2. In the white area on the left, right mouse click and select "Add".

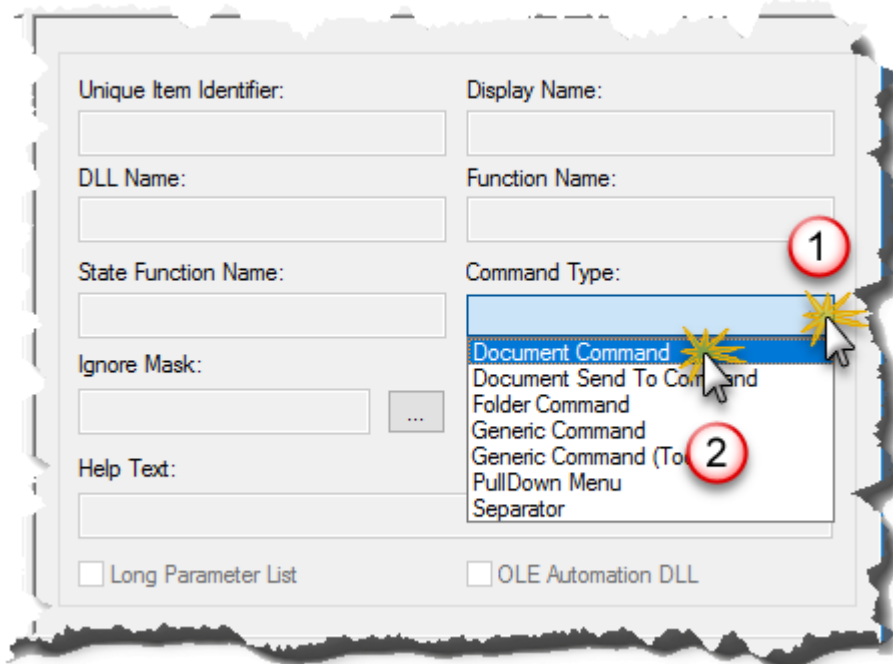
This will allow you to add a command OUTSIDE of the PullDown menu. Note that you must NOT have the "MyMenu" item selected. If you do have it selected, you will be adding a menu item UNDER "MyMenu".



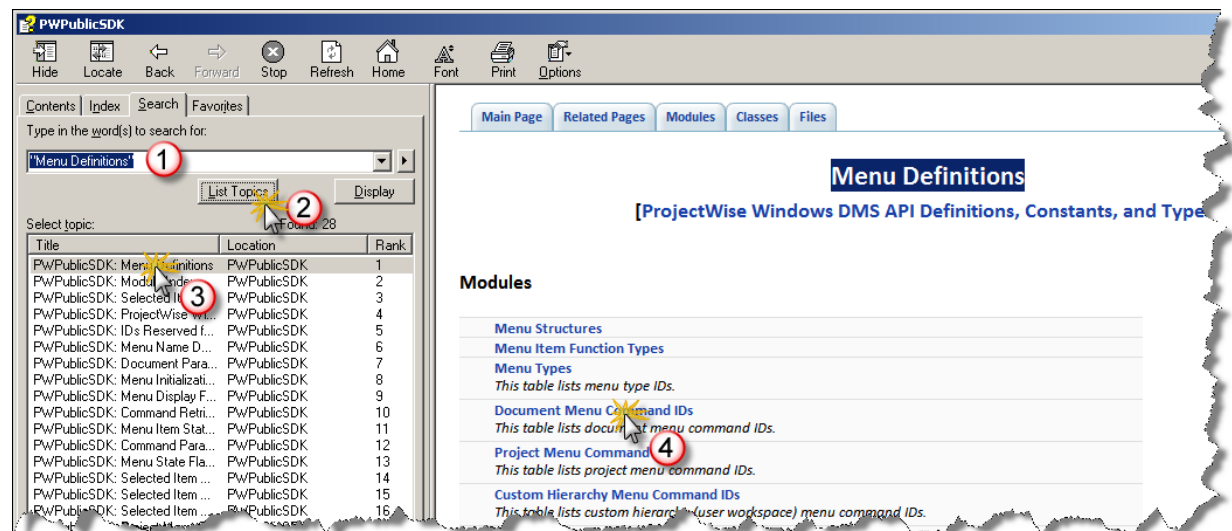
3. Select "Document Command" as the *Command Type*.

Creating Menu Commands in ProjectWise Explorer via a MRR File

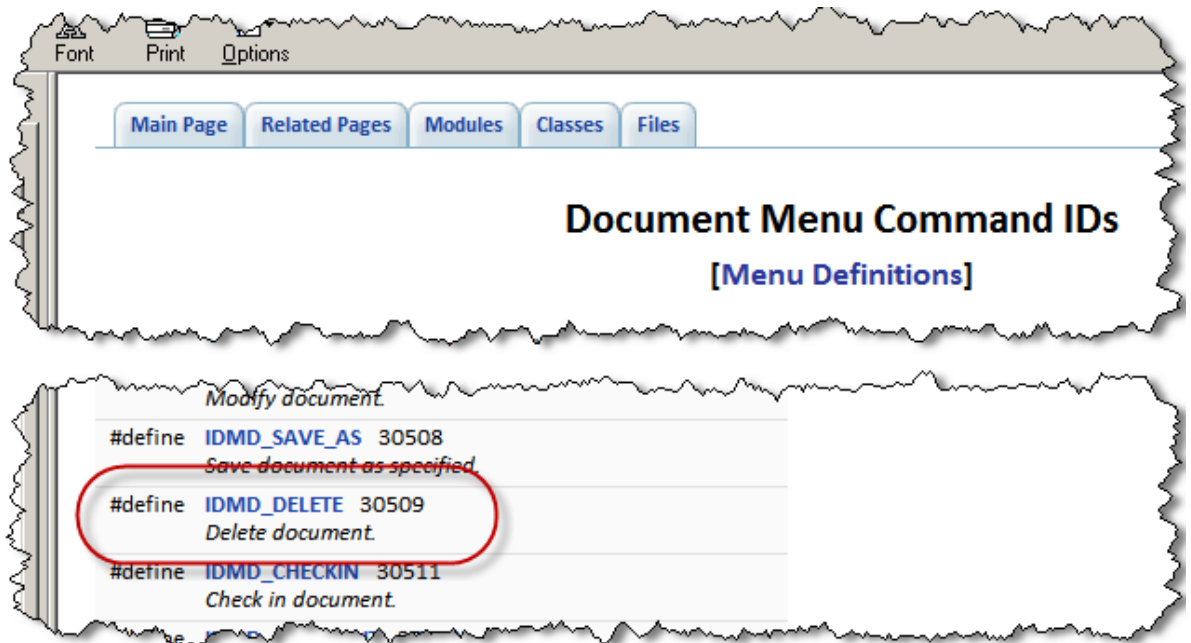
ProjectWise SDK



You need to know the menu identifier for the command you want to intercept. You can find these in the ProjectWise help file by searching for “Menu Definitions” (include the double quotes).



Since we want to disable the document delete command, we then look under “Document Menu Command IDs”

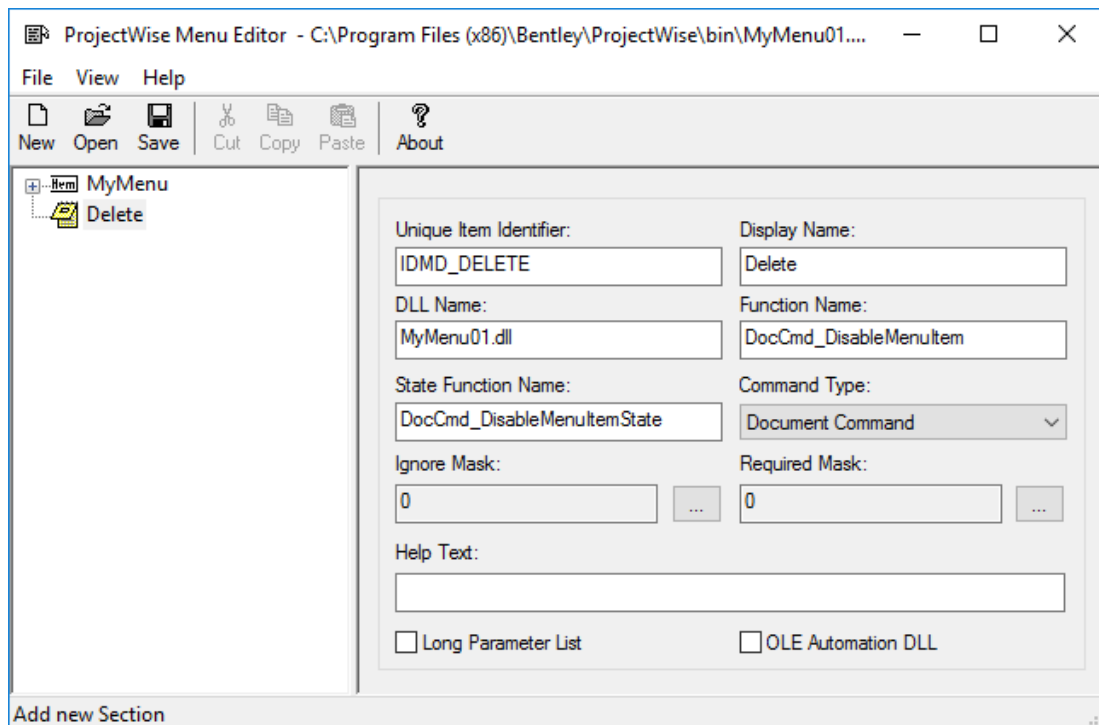


4. Fill in the text boxes on the ProjectWise Menu Dialog as follows:

- Unique Item Identifier: IDMD_DELETE
- Display Name: Delete
- DLL Name: MyMenu01.dll
- Function Name: DocCmd_DisableMenuItem
- State Function Name: DocCmd_DisableMenuItemState
- Help Text: (leave it blank)

Creating Menu Commands in ProjectWise Explorer via a MRR File

ProjectWise SDK



5. Save your menu file now.
6. Look in the ProjectWise SDK help for the prototype for a MRR Document Command and use it to create your function as in the previous exercises.

Name your function "DocCmd_DisableMenuItem". The function should just return zero.

```
extern "C" int WINAPI DocCmd_DisableMenuItem
{
    unsigned int    count,
    long*           pProjects,
    long*           pDocuments
}

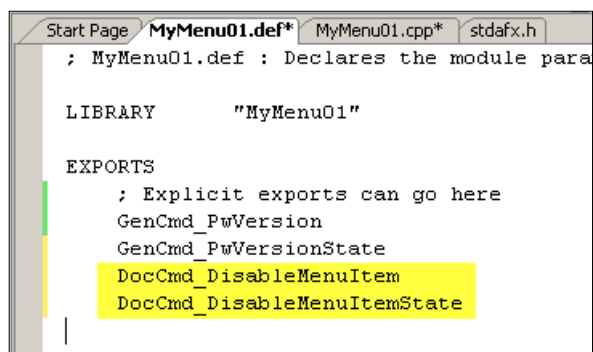
{
    AFX_MANAGE_STATE(AfxGetStaticModuleState());
    return 0;
}
```


7. Create the state function "DocCmd_DisableMenuItemState".

The function should just return AAMS_GRAYED.

```
extern "C" int WINAPI DocCmd_DisableMenuItemState
(
    UINT    uiCmdId,
    ULONG    ulState
)
{
    AFX_MANAGE_STATE(AfxGetStaticModuleState());
    return AAMS_GRAYED;
}
```

8. Update the .def file with these two new menu functions:



```
Start Page  MyMenu01.def*  MyMenu01.cpp*  stdafx.h
; MyMenu01.def : Declares the module parameters for the
;               MyMenu01.dll

LIBRARY        "MyMenu01"

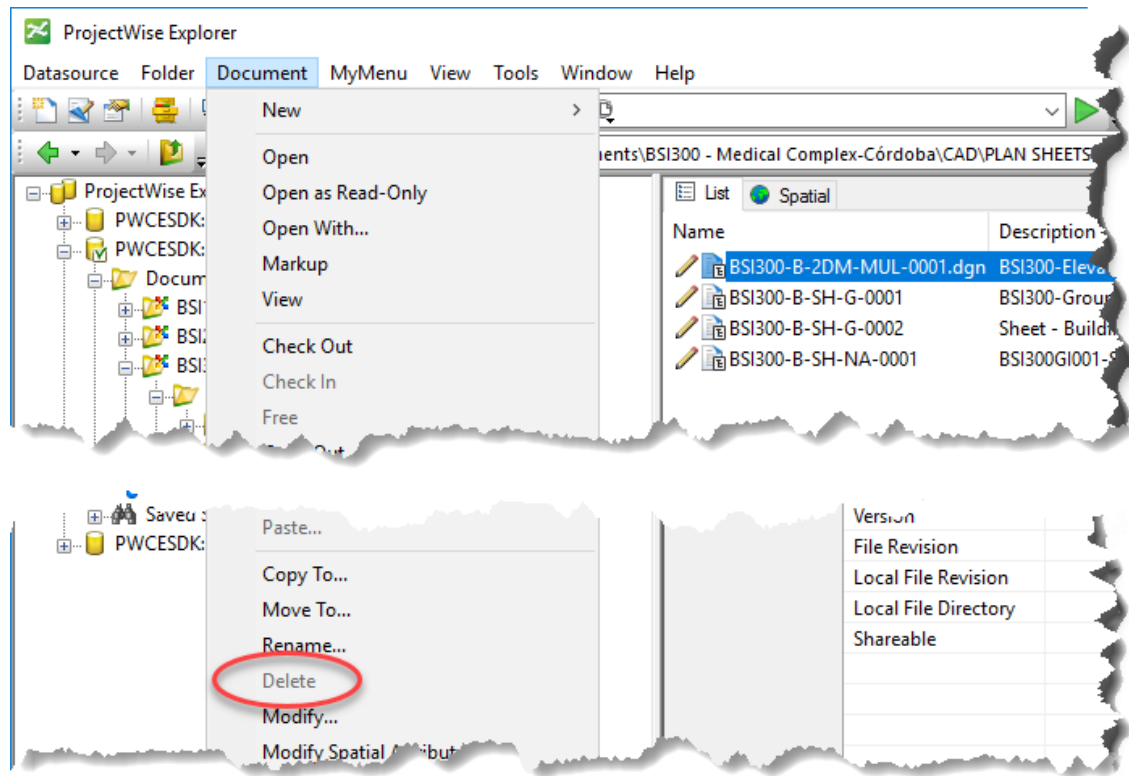
EXPORTS
; Explicit exports can go here
GenCmd_PwVersion
GenCmd_PwVersionState
DocCmd_DisableMenuItem
DocCmd_DisableMenuItemState
```

9. Build and run the code.

You should not be able to delete a document as the document delete command should be grayed out, even if you are an Administrator.

Creating Menu Commands in ProjectWise Explorer via a MRR File

ProjectWise SDK



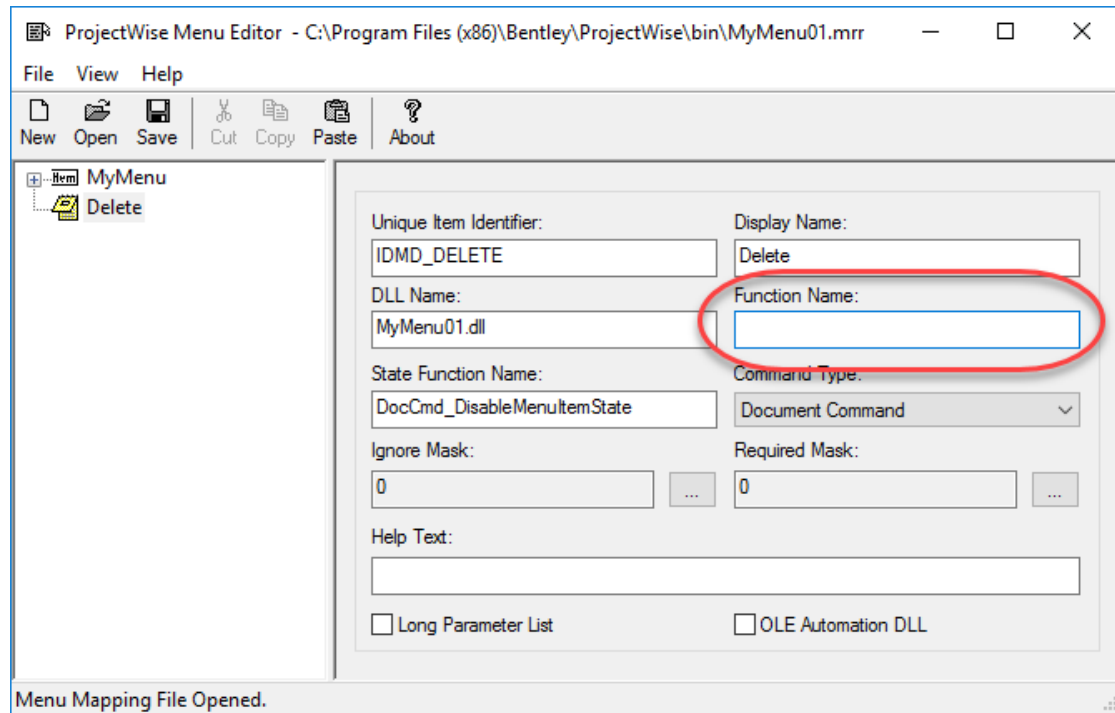
Note: If you do not specify a function in your MRR file, you can disable the menu command by either using the Ignore Mask in the menu editor, or by using a state function. However, the menu item you are trying to override can be called by other features and will still work.

To see this work, create a test document (it doesn't need to have a file attached) and then try to delete it with the Del key. If your code matches the solution provided, you should NOT be able to delete the document this way.

If you edit the menu file and remove the function, but leave the state function like this:

Creating Menu Commands in ProjectWise Explorer via a MRR File

ProjectWise SDK



and then run the program again and try to delete a document with the Del key, you should be able to delete the document.

So, if you want to completely disable a menu command using the menu editor, be sure to provide a function that does nothing but returns zero, so if there is another way to activate the menu command, it will not be able to run the menu command.

10. After you are done with these exercises, please select "Clean Solution" in Visual Studio so that this customization is no longer available. This is to minimize any confusion in future exercises.

Build > Clean Solution

11. Exit Visual Studio.