



Adding 3rd Party Applications to ProjectWorX32

APPLICATIONS NOTE

August 2004



Description: Adding Additional Items to the ProjectWorX32 Projects Tree.

OS Requirement: Windows NT, 2000, ME, XP.

General Requirement: Requires knowledge of ICONICS ProjectWorX32, as well as some knowledge of Microsoft Visual C++ and programming in a Windows Environment.

Introduction

ProjectWorX32 has been designed with the ability to adopt additional products to either its “Applications” or “OPC Server” trees. In order to add these customized items, ProjectWorX32 needs various data for string constants, icons, and application code that will make an item unique, with a user-defined set of behavior and appearance.

The required data is gathered from customized dynamic link library (.dll) files. One .dll file represents one child of the “Applications” or “OPC Server” node. The .dll file’s resource descriptor contains all of the string constants and icons needed to make the item appear in ProjectWorX32 as desired. Internal code in the .dll file defines actions that allow ProjectWorX32 to interface with the product being added to the tree, and perform such actions as launching the application or creating a new document for the application.

Initially, ProjectWorX32 is blind to the capabilities and appearance of the application being added, and the way it is to appear in the tree. Therefore, all of the .dll files that are representing custom applications share a common automation interface, or a common set of exposed functions, through which ProjectWorX32 can discover specifics such as string constants, icons, and code. When the .dll file registers, it informs ProjectWorX32 of its existence (and its interface) and when it un-registers, it clears itself from a list of custom applications.

The ProjectWorX32 Add-on AppWizard

ICONICS developed a Microsoft Visual C++ Custom AppWizard, which will create the framework of the .dll file. The Wizard can be located on the GENESIS32 Product CD in **Tools\ProjectWorX32 Toolkit\PWXExtension.awx**. The PWXExtension.awx file must be copied into the “Program Files\Microsoft Visual Studio\Common\MSDev98\Template” directory. (Visual Studio/Visual C++ 6.0 must be installed). Launching a new instance of Visual C++ and going to File -> new should show the new toolkit.

After this wizard is installed, a new item named “ICONICS ProjectWorX32 Add-on AppWizard” will be seen in the “New project” dialog of Visual C++ 6.0. In the following example we will go through the process of creating a branch in ProjectWorX32 representing an imaginary product called “CustomWorX32.”

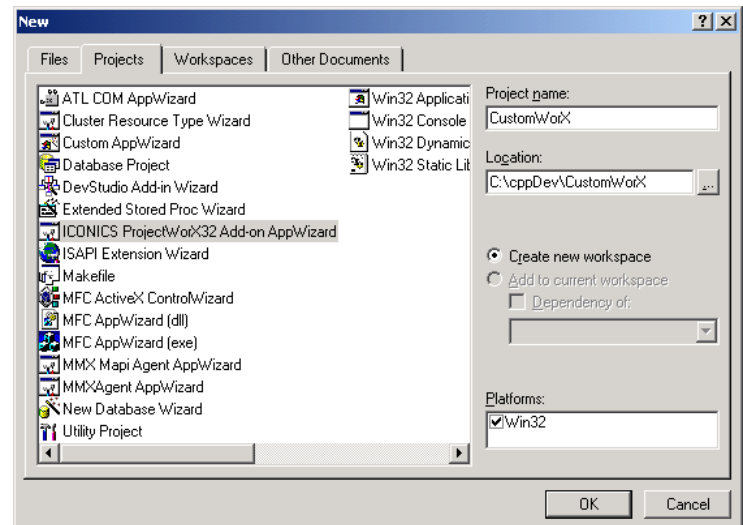


Figure 1: Starting the ProjectWorX32 Add-on AppWizard

A ProjectWorX32 Add-on AppWizard begins the way any wizard-based Visual C++ project would begin. Simply supply a name and location for your project, and make sure that the proper item is selected in the list on the left. The wizard begins with the figure below.

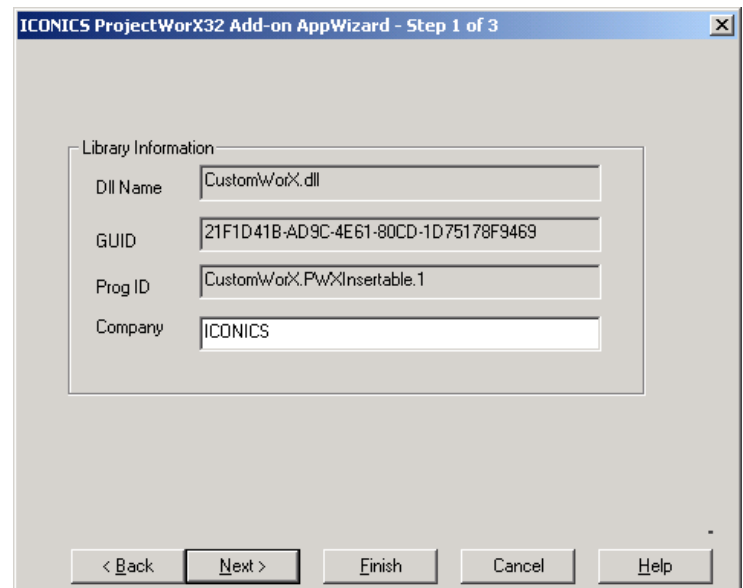


Figure 2: First Page of the Wizard

The purpose of the first page of the wizard simply supplies information about the workspace that is about to be created. For those familiar with COM programming, the GUID (guaranteed unique identifier) and ProgID are the GUID and ProgID of the interface that ProjectWorX32 will go through. All items created with this wizard must have a Prog ID of *ProjectName.PWXInsertable.1*. Given the ProjectWorX32 internally handles the locating of all of its Custom Apps, the



Adding 3rd Party Applications to ProjectWorX32

APPLICATIONS NOTE

August 2004



average end user need not be concerned with these parameters. The user may enter a Company Name. The next step of the wizard allows the user to begin constructing a sub-tree for the custom application.

Figure 3: Parent and Child Elements

The “Product Name” field is where you place the name of the item, as it will appear in ProjectWorX32 under the “Applications” or “OPC Server” branch. The checkbox underneath this field determines which of those two branches the item will reside. If the product is an OPC Server, the user must enter the OPC Server ProgID. An “OPC Server” item also does not have a need for any child nodes, so in the event that the “Product is OPC Server” checkbox is enabled, the “Child Products” frame will become disabled. If the box is left unchecked, however, as seen in the example above, you are required to enter in several child products. For example, if the product name were “Microsoft Office”, then the child products might be “Word”, “Excel”, “Access” and “Power point”. (See the Microsoft Office Example). The name chosen for “Product Name” should be broad enough as to lend itself to “child” products. Our imaginary product “CustomWorX32” has two child nodes, “Configurator” and “Viewer.” If the product is not an OPC Server, the next leads to a series of dialogs in which each of the children specified is described in more detail. If the product is an OPC Server, the same page appears requesting detailed information on the OPC Server (treated like the OPC Server is its own one and only child node). The following figure shows the Customization dialog.

Figure 4: Customizing a Child Element

Above, we see the configuration for the “Configurator” child of “CustomWorX32”. We have omitted the configuration for “Viewer” for the sake of avoiding redundancy. Child nodes in ProjectWorX32 often have the option to create a new document, add an existing document, or “make active” a document that has been added. These menus can be turned on and off by using the checkboxes on the right-hand side. Captions for the menus go in the appropriate fields. If the “New Document” menu is turned on, then there are two ways the user can create a new document. The user may have a “template” file, which is essentially an empty document for the custom application, residing in the Genesis32\bin\PWXTemplates directory. If the appropriate box is checked, then ProjectWorX32 will look for the specified file in that location and make a copy every time a new document is requested. If the box is left unchecked, then the user will be responsible, in the future, for supplying code in the .dll file, that will programmatically handle the creation of a new document. Likewise, if the “Make Main Document” is enabled, then the user has the future responsibility of supplying code that will make a document active. ProjectWorX32 has the ability to interface with Microsoft Visual Source Safe and put files under version control. This ability can be extended to your custom application by selecting the “Enable Source Control Menus” button. Next to “Product Name” is a field to enter how the item will appear in the ProjectWorX32 tree underneath the previously configured parent element. In this example, a parent element “CustomWorX32” with a child “Configurator” will be seen in the Applications tree. The associated database entry can be ignored. Enter a description for the product and the file extension associated with all of your product’s documents. Once all of the child nodes have been configured, the wizard will be completed and a new Visual C++ workspace will be generated. This workspace may be compiled immediately to



Adding 3rd Party Applications to ProjectWorX32

APPLICATIONS NOTE

August 2004



generate a “bare-bones” likeness of the custom product in the ProjectWorX32 projects tree.

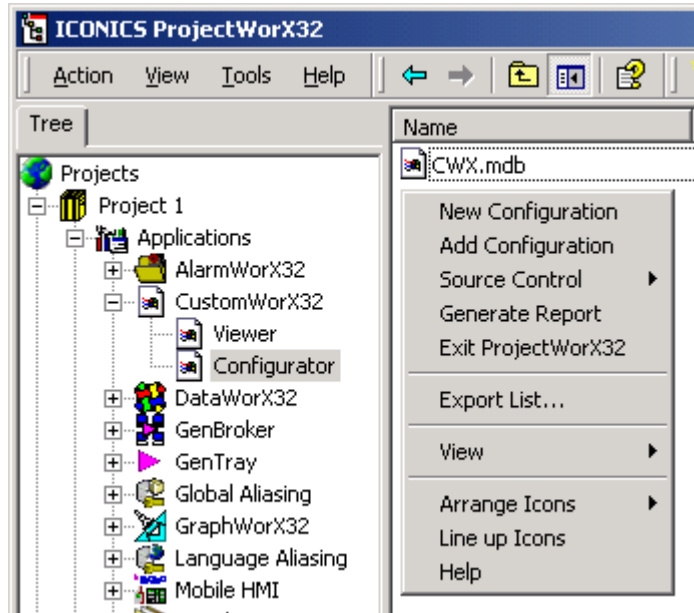


Figure 5: Custom Application Appearing in ProjectWorX32

In the figure above, we see that CustomWorX32 has been added to the Applications tree. The menu to the right shows the “new” and “add” menus as configured in the wizard. You will also notice the generic icons associated with CustomWorX32. Furthermore, once a document is added (like CWX.mdb in the figure above), there is no way to launch the imaginary CustomWorX32 Configurator from ProjectWorX32, and no way to make CWX.mdb the active document. The further customization of strings, icons and code is explained in the next section.

Customizing the New Workspace

The first place to look after creating the new workspace is in the “ResourceView” tab of the Visual C++ development environment. Here, you will see a series of bitmaps. These bitmaps represent all of the icons needed in the ProjectWorX32 tree and results views.

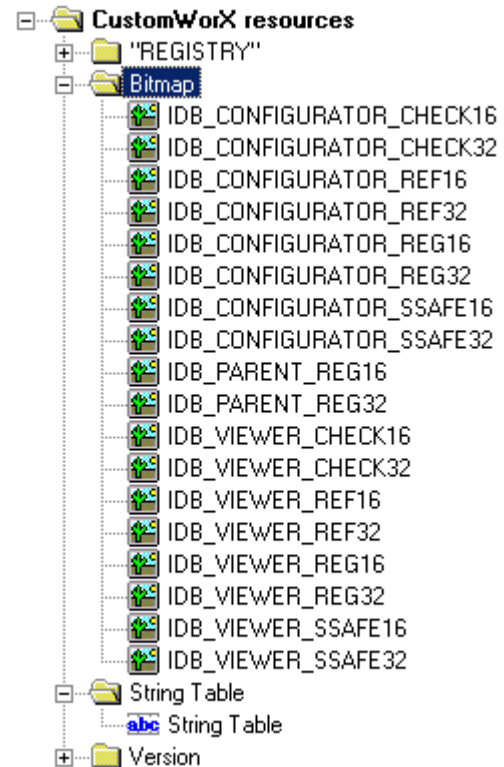


Figure 6: Resource of Workspace

All Bitmaps conform to the following convention

IDB_NodeName_CategorySize

Where “node name” is the name of a child node as configured in the wizard (Parent is a reserved word denoting the parent node). Category is divided into four possible strings; “Reg”, “Ref”, “Ssafe” and “Check”. “Reg” denotes an icon’s appearance under normal circumstances. “Ref” denotes the appearance of an icon if it is a reference to a file rather than a link to a file directly contained in the project. “Ssafe” and “Check” are applicable to files that are under source control. If you add a file to source control, there needs to be some way of denoting that in the items icon. Furthermore, there needs to be some way to mark an item that is under source control and checked out. The “Size” field is either 16 or 32 and denotes “small icon” and “large icon” respectively. To customize your application with your own icons, simply edit the bitmap(s) associated with the item(s) you wish to change from the generic icon supplied. Make sure that all size 16 items remain 16 x 16 pixels and that all size 32 items remain 32 x 32 pixels. Do not modify the name of any bitmap. All of the string constants configured in the wizard now appear in the string table of the .dll file’s resource. They can be modified at any time.



Adding 3rd Party Applications to ProjectWorX32

APPLICATIONS NOTE

August 2004



ID	Value	Caption
IDS_OPCSERVER_PROGID	99	N/A
IDS_PROJNAME	101	Custom\WorX
IDS_PARENT_PROJECT_NAME	102	Custom\WorX32
IDS_CONFIGURATOR_NAME	103	Configurator
IDS_CONFIGURATOR_DESCRIPTION	104	Configurator for Custom\WorX32
IDS_CONFIGURATOR_MENU_NEW	105	New Configuration
IDS_CONFIGURATOR_MENU_ADD	106	Add Configuration
IDS_CONFIGURATOR_MENU_MAKE	107	
IDS_CONFIGURATOR_FILE_EXTENSION	108	.mdb
IDS_CONFIGURATOR_DATA_PATH	109	22 Configurator
IDS_CONFIGURATOR_TEMPLATE_PATH	110	Custom\WorX32.mdb
IDS_VIEWER_NAME	111	Viewer
IDS_VIEWER_DESCRIPTION	112	Viewer for Custom\WorX32
IDS_VIEWER_MENU_NEW	113	
IDS_VIEWER_MENU_ADD	114	
IDS_VIEWER_MENU_MAKE	115	
IDS_VIEWER_FILE_EXTENSION	116	.mdb
IDS_VIEWER_DATABASE_TEMPLATE_PATH	117	23 Viewer
IDS_VIEWER_TEMPLATE_PATH	118	NONE

Figure 7: String Table of Workspace

Writing Custom Code

When adding custom code to the newly created .dll file, it is strongly recommended that you only edit the "InsertableCode.cpp" file. This file has all of the functions available neatly laid out with helpful comments. A description of each function is supplied here as well.

OpenDocument

Parameters: Document Path (string), Child Project Index (long)

Return Value: Success/Failure

Description:

Given the path to a document, and the index of the child project (in the order they were added in the wizard) the user will need to supply code that will launch the appropriate product.

CanVerifyOwnership

Parameters: Document Path (string), Child Product Index (long)

Return Value: True/False

Description:

If a user wishes to have the capability of having ProjectWorX32 automatically identify documents as being part of the custom product, then this function should return true. The ability to identify documents as belonging to a product is useful when a file is manually added to the project folder outside of ProjectWorX32 and is then detected by ProjectWorX32 upon opening a project. If the product has the ability to verify ownership, then the document can be added (given that the VerifyOwnership function is implemented. See below).

VerifyOwnership

Parameters: DocumentPath (string), Child Product Index (long)

Return Value: True/False

Description:

If "CanVerifyOwnership" returns true, then this function must be implemented. The user will have to write some code here which, based on the document path, can verify that the document belongs to the appropriate child product. By default, this function will simply compare the file extension of the document to the file extension associated with the child product. Suppose, however, that a product opens HTML files. Simply checking the file extension would not prove ownership. If all of the HTML files, however, contained the text "<Meta NAME='company' CONTENT='SomethingSpecific'>" then the code for VerifyOwnership could do a search on the text in the HTML file. If there is no easy way to verify ownership, then the CanVerifyOwnership function should return false.

CanSearch

Parameters: none

Return Value: True/False

Description:

This function returns true if the user wishes to have the ability to search for a string in a document for the custom product.

CanReplace

Parameters: none

Return Value: True/False

Description:

This function returns true if the user wishes to have the ability to search for a string and replace it with another in a document for the custom product.

Search

Parameters: Document Path (string), Search String (string), Child Product Index (long)

Return Value: True/False

Description:

If CanSearch returns true, then this function will need to be implemented. The code for this function will need to search the document whose path is given and determine if the search string is in that document. For products whose documents are more than mere text files, the function can be difficult to write. For example, for GraphWorX32 display files, it is necessary to use the Automation interface supplied by GraphWorX32 to dig through a file and perform the search..

Replace

Parameters: Document Path (string), Search String (string), Replace String (string), Child Product Index (long)

Return Value: success/failure

Description:



Adding 3rd Party Applications to ProjectWorX32



APPLICATIONS NOTE

August 2004

Works almost identically to search, except in this case, once the string is found, it needs to be replaced by the “Replace string.” Also, code for saving the document once this replace is complete is necessary.

MakeMainDocument

Parameters: Document Path (string), Child Product Index (long)

Return Value: success/failure

Description:

If, in the Wizard, the menu for setting a document to be the main document was enabled, then this function must be implemented. Based on the document path and the child product, the user must supply code that will in some way notify the appropriate product that this document is the “main” document.

InitializeSearch

Parameters: Child Product Index (long)

Return Value: success/failure

Description:

When ProjectWorX32 performs a search or replace action, it does all the files for one application before going onto the next. For the sake of efficiency, InitializeSearch and UnInitSearch are supplied. Before calling search or replace on each of the documents that are associated with the given child product, ProjectWorX32 will call your InitializeSearch function. After it is through will all of the documents, it will call UnInitSearch. (One way these functions may be used is to create an instance of your application on InitializeSearch, use that instance on each of the search or replace calls, and close the instance of your application in the UnInitSearch function.)

UnInitSearch

Parameters: Child Product Index (long)

Return Value: success/failure

Description:

(See InitializeSearch)

CanStoreActiveDB

Parameters: Child Product Index (long)

Return Value: success/failure

Description:

Some products that support the notion of an “active” document also have the ability to internally store the location of the active document. If this is so, this function returns true, and ActiveDB should be implemented.

ActiveDB

Parameters: Child Product Index (long)

Return Value: location of the Active Database (string)

Description:

Return the string containing the location of the current active document.

NewDocument

Parameters: Child Product Index (long), Document Path (string), OpenAfterCreate (Boolean)

Return Value: location of the Active Database (string)

Description:

If, in the wizard, the application corresponding to the child index did not rely on a template file for creating a new document, code for creating a new document for the application must be supplied here, and saved in the location specified by “Document Path”. If “OpenAfterCreate” is true, then launch the application with the newly created document.

VerifyInstalled

Parameters: Child Product Index (long)

Return Value: true/false

Description:

In ProjectWorX32, items that exist in the Application tree that are not properly installed display an appropriate error message. Have this function return false if you are able to detect that the product your .dll file maps to be not properly installed.

GetParentOEMName

Parameters: New Name (reference to string)

Return Value: true/false

Description:

The name of the custom application, seen in the parent node, hard-coded into the .dll file’s string resource, can be over-written by having this function return true. If the function returns false, then ProjectWorX32 will show the default, hard-coded value. If the function returns true, then the parent node name will be over-written by the value placed into the input/output string parameter. This is useful if a product is being licensed to an OEM (original equipment manufacturer) who is using the product under a different name. Under normal use, this function will most likely read a string from the registry and return true if the registry string exists.