Custom AppWizard Tools Component - Overview

This component is designed for use with a custom AppWizard project. A custom AppWizard is similar to the default AppWizard, except that you define the steps it performs and the files it generates, allowing you to create projects based on a custom template. See *Creating a Custom AppWizard* in *Beginning Your Program* in the *Visual C++ Programmer's Guide* for more information on custom AppWizards. The Details section of *Beginning Your Program* contains the AppWizard Programming Reference. You should be familiar with custom AppWizards before using this component or reading further.

Once you've created a custom AppWizard project, you can apply this component to add two pieces of functionality:

- A set of functions to make it easier to define new macros to be used by the AppWizard template parser.
- A new dialog box in your custom AppWizard that allows the custom AppWizard user to browse the classes and files you create. This dialog box is recommended for use only with a custom AppWizard based on your own custom steps.

After you have inserted the component, refer to the file AwxTools.txt that it generates. This file contains information on what to do next and directs you to `TODO` comments inserted in your code.

Custom AppWizard Tools Component - Specifics

Custom AppWizard Tools Component - Specifics

Macro Helper Functions

This component generates two new files: WizUtil.cpp and WizUtil.h, which implement macro helper functions. These functions simplify the task of setting your macros, without requiring you to directly access the dictionary inside your **CCustomAppWiz**-derived class.

Three Types of Macros

There are three functions you can use, depending on the type of data you want to store as the value of the macro: string, integer, and Boolean. The functions are called `DefineStringMacro`, `DefineIntMacro`, and `DefineBoolMacro`. (Recall that all your macros are actually strings and are stored in your **CCustomAppWiz**-derived class's dictionary, which is just a **CMapStringToString** object.) These helper functions make it easier to set the macros based on the type of data you want to store in the string.

String macros have values that are just strings of characters. The `DefineStringMacro` function sets the macro's value to be the string you pass it.

Integer macros have values that you treat as integers, although they're actually strings (for example, the string "4"). The `DefineIntMacro` function takes the integer you pass it and sets the macro's value to be the string equivalent of that integer.

Boolean macros are used in "$$IF" and "$$ELIF" directives. The only useful information about a Boolean macro is whether it's stored in the dictionary at all. The macro's actual value is unimportant. If the macro is in the dictionary at all, it's considered to be "true" in the context of an "$$IF" or "$$ELIF". If the macro is not in the dictionary, it's considered to be "false." The `DefineBoolMacro` function either puts the macro in the dictionary (if you pass **TRUE**) or removes the macro from the dictionary (if you pass **FALSE**).

Overloads for $$BEGINLOOP/$$ENDLOOP

Each of the three macro helper functions is overloaded to potentially take an `iItem` parameter. These versions of the functions are useful for setting the value of a macro to be used on the `iItem`th iteration of a loop. For example, consider the call

```
DefineStringMacro("MACRO", 4, "VALUE")
```

In this case `iItem` is 4, so the result is that "MACRO_4" in the dictionary gets set to "VALUE". Thus, in your template on the fifth iteration of a loop (because iterations are numbered starting with 0), $ $MACRO$$ will be expanded to VALUE.

Classes Browser Step

This component also makes the following additions to your custom AppWizard project to provide it with a new class-browser step:

- A dialog template for the new step, having CG_IDD_CLASSES as its resource ID.
- A dialog class `CClassesDlg`, implemented in the files ClassDlg.cpp and ClassDlg.h.
- A class `CNames` to store information for each item displayed in the dialog, implemented in the files Names.cpp and Names.h.

Dialog Template

This new step resembles the final step in AppWizard (step 6 of 6 when creating an MDI or SDI application, for example). It contains a list box at the top with an entry for each class your custom AppWizard creates. Below it has edit controls for the selected class's name, implementation filename, and header filename, and a static control to display its base class.

Dialog Class: CClassesDlg

The dialog class for this step has the following features to make it easy to use:

- Automatic validation of filenames and class names, which also warns against duplicate names.
- Auto-tracking, so that typing a class name will automatically fill in the names for the implementation and header file. This tracking stops for each field the user edits manually.
- A dynamically changeable array, to which you, the custom AppWizard writer, can add, remove, and edit entries corresponding to the classes (see `CNames` below).
- Automatic definition of template macros to store the class and file information edited by the user.

Class Information Storage: CNames

Each entry in the list box of classes on this step corresponds to an instance of the `CNames` class. All of these instances are stored together in a dynamic `CNamesArray` member variable of `CClassesDlg`. This array is public so that it can be manipulated by the other steps of your custom AppWizard when their options affect the classes your wizard generates.

A sample list of classes is provided to show you how to use `CNames`. Typically you'll call the following function:

```
void CNames::Init(LPCTSTR szMacroName, LPCTSTR szName,
     LPCTSTR szBaseClass, BOOL bPrefixWithRoot = TRUE)
```

For example, the call

```
Init("SAMPLEDOC", "SampleDoc", "CDocument");
```

causes this `CNames` object to represent a class called `C`<*projectName*>`SampleDoc` (where <*projectName*> is the name of the project your custom AppWizard user is creating). Its base class is **CDocument**. The initial header and implementation filenames are derived from the class name. Of course, the class name and filenames can all be changed when the custom AppWizard user edits their values in the class browser.

In addition, the following AppWizard macros are defined when your step is dismissed:

- SAMPLEDOC_CLASS
- SAMPLEDOC_BASE_CLASS
- SAMPLEDOC_HFILE
- SAMPLEDOC_IFILE

In the example above, the `bPrefixWithRoot` parameter is set to **TRUE**, which is why <*projectName*> will appear in the class name. If you explicitly pass **FALSE** as `bPrefixWithRoot`, then, for example, in this case the class name would be `CSampleDoc` (no <*projectName*>).

Limitations

This dialog box is intended to be used only with a custom AppWizard based on your own custom steps.

A custom AppWizard based on an existing project does not have a `CDialogChooser` class, which is a requirement for adding this dialog. The Custom AppWizard Tools component cannot be applied to such a project.

A custom AppWizard based on standard AppWizard steps already has the standard AppWizard equivalent of this dialog (step 6 of 6 when creating an MDI or SDI application, for example), which displays information about the classes it will generate. Do not use this component to replace that step.

Custom AppWizard Tools Component - Class Page

This component requires some information about your project. If the default values are incorrect, enter the correct ones. (Below, replace <*AppWiz*> with the name of the **CCustomAppWiz**-derived class defined in your project.)

Global variable of type <*AppWiz*>: Enter the name of the global instance of your **CCustomAppWiz**-derived class.

<*AppWiz*> header file: Enter the name of the header file in which your **CCustomAppWiz**-derived class is declared.

Click OK on this dialog to insert the component..