Providing a custom autocomplete source for an edit control

devblogs.microsoft.com/oldnewthing/20130923-00

September 23, 2013

Raymond Chen

Today's Little Program shows a custom source for autocomplete. It's nothing exciting, but at least's it's something you can use as a starting point for your own customizations.

We start with a dialog template, whose edit control will be the target of a custom autocomplete.

```
// scratch.rc
#include <windows.h>
1 DIALOGEX DISCARDABLE  32, 32, 200, 56
STYLE DS_MODALFRAME |  WS_POPUP |
      WS_VISIBLE | WS_CAPTION | WS_SYSMENU
CAPTION "Sample"
FONT 8, "MS Shell Dlg"
BEGIN
 LTEXT "What is your favorite Seattle restaurant?",-1,7,8,184,10
 EDITTEXT 100,7,18,184,14
 PUSHBUTTON "OK",IDOK,146,38,50,14
END
```

Just for fun, I wrote the program in ATL. Instead of complaining that my code is hard to understand because I didn't use an application framework, people can now complain that my code is hard to understand because I used the *wrong* application framework.

```
// scratch.cpp
#include <windows.h>
#include <ole2.h>
#include <windowsx.h>
#include <shlobj.h>
#include <atlbase.h>
#include <atlcom.h>
CComModule _Module;
```

To save some typing, I define a shorthand name for "the predefined ATL object for enumerating strings via `IEnumString`."

```
typedef CComEnum<IEnumString,
                 &IID_IEnumString,
                 LPOLESTR,
                 _Copy<LPOLESTR> > CComEnumString;
```

To initialize the dialog, we do the following things:

- Create a predefined ATL object for implementing `IEnumString`.
- Tell the predefined ATL object to enumerate a hard-coded list of restaurant suggestions.
- Create an autocomplete object.
- Connect the autocomplete object to the edit control in the dialog and to the `IEnumString` object.
- Just for fun, change some of the default settings for autocomplete.

```
LPOLESTR c_rgpszSuggestions[] = {
    L"Brave Horse Tavern",
    L"Cuoco",
    L"Dahlia Bakery",
    L"Dahlia Lounge",
    L"Etta's",
    L"Lola",
    L"Palace Kitchen",
    L"Seatown",
    L"Serious Pie",
    L"Ting Momo",
};
void OnInitDialog(HWND hdlg)
{
  CComPtr<IAutoComplete2> spac;
  CComObject<CComEnumString> *pes;
  HRESULT hr = CComObject<CComEnumString>::CreateInstance(&pes);
  CComPtr<IEnumString> spes(pes);
  if (SUCCEEDED(hr) &&
      SUCCEEDED(pes->Init(&c_rgpszSuggestions[0],
                          &c_rgpszSuggestions[ARRAYSIZE(c_rgpszSuggestions)],
                          NULL)) &&
      SUCCEEDED(spac.CoCreateInstance(CLSID_AutoComplete)) &&
      SUCCEEDED(spac->Init(GetDlgItem(hdlg, 100), spes, NULL, NULL)) &&
      SUCCEEDED(spac->SetOptions(ACO_AUTOSUGGEST | ACO_UPDOWNKEYDROPSLIST))) {
  }
}
```

The rest is just boilerplate.

```
INT_PTR CALLBACK DlgProc(HWND hdlg, UINT uMsg, WPARAM wParam, LPARAM lParam)
{
  switch (uMsg) {
  case WM_INITDIALOG:
    OnInitDialog(hdlg);
    return TRUE;
  case WM_COMMAND:
    switch (GET_WM_COMMAND_ID(wParam, lParam)) {
    case IDOK:
      EndDialog(hdlg, 0);
      break;
    }
  }
  return FALSE;
}
int WINAPI WinMain(HINSTANCE hinst, HINSTANCE hinstPrev,
                   LPSTR lpCmdLine, int nShowCmd)
{
  if (SUCCEEDED(CoInitialize(NULL))) {
    DialogBox(hinst, MAKEINTRESOURCE(1), NULL, DlgProc);
    CoUninitialize();
  }
  return 0;
}
```

Now, one of the reasons for using a framework is that it hides a lot of details from you. But if you are trying to understand how to port code from one framework to another, those hidden details become an obstacle to progress rather than a convenience. You may port the overall structure from one framework to another, but if the two frameworks behave differently in the hidden parts, your conversion was incorrect.

For example, one subtlety hidden in the above code is how the strings are returned by the `IEnumString::Next` method. Recall that COM interfaces use the task allocator to pass memory between objects, so the string returned by `IEnumString::Next` is allocated by `CoTaskMemAlloc`, with the expectation that the caller will call `CoTaskMemFree` to free it.

Unless you happen to be familiar with this detail of ATL, you would never have guessed it from the code above. You might have thought that the enumerator handed out the literal string pointers used to initialize it, and then you'll start wondering why your program crashes at random times (because you introduced a heap corruption bug).

Raymond Chen

Follow