You can extend the PROPSHEETPAGE structure with your own bonus data

devblogs.microsoft.com/oldnewthing/20110318-00

March 18, 2011

Raymond Chen

... for when regular strength lParam just isn't enough.

A little-known and even less-used feature of the shell property sheet is that you can hang custom data off the end of the `PROPSHEETPAGE` structure, and the shell will carry it around for you. Mind you, the shell carries it around by means of `memcpy` and destroys it by just freeing the underlying memory, so whatever you stick on the end needs to be plain old data. (Though you do get an opportunity to "construct" and "destruct" if you register a `PropSheetPageProc` callback, during which you are permitted to modify your bonus data and the `lParam` field of the `PROPSHEETPAGE`.)

Here's a program that illustrates this technique. It doesn't do much interesting, mind you, but maybe that's a good thing: Makes for fewer distractions.

```
#include <windows.h>
#include <prsht.h>
HINSTANCE g_hinst;
struct ITEMPROPSHEETPAGE : public PROPSHEETPAGE
{
 int cWidgets;
 TCHAR szItemName[100];
};
```

`ITEMPROPSHEETPAGE` is a custom structure that appends our bonus data (an integer and a string) to the standard `PROPSHEETPAGE`. This is the structure that our property sheet page will use.

```
INT_PTR CALLBACK DlgProc(HWND hwnd, UINT uiMsg, WPARAM wParam, LPARAM lParam)
{
 switch (uiMsg) {
 case WM_INITDIALOG:
  {
   ITEMPROPSHEETPAGE *ppsp =
      reinterpret_cast<ITEMPROPSHEETPAGE*>(lParam));
   SetDlgItemText(hwnd, 100, ppsp->szItemName);
   SetDlgItemInt(hwnd, 101, ppsp->cWidgets, FALSE);
  }
  return TRUE;
 }
 return FALSE;
}
```

The `lParam` passed to `WM_INITDIALOG` is a pointer to the shell-managed copy of the `PROPSHEETPAGE` structure. Since we associated this dialog procedure with a `ITEMPROPSHEETPAGE` structure, we can cast it to the larger structure to get at our bonus data (which the shell happily `memcpy` 'd from our copy into the shell-managed copy).

```
HPROPSHEETPAGE CreateItemPropertySheetPage(
    int cWidgets, PCTSTR pszItemName)
{
 ITEMPROPSHEETPAGE psp;
 ZeroMemory(&psp, sizeof(psp));
 psp.dwSize = sizeof(psp);
 psp.hInstance = g_hinst;
 psp.pszTemplate = MAKEINTRESOURCE(1);
 psp.pfnDlgProc = DlgProc;
 psp.cWidgets = cWidgets;
 lstrcpyn(psp.szItemName, pszItemName, 100);
 return CreatePropertySheetPage(&psp);
}
```

It is here that we associate the `DlgProc` with the `ITEMPROPSHEETPAGE`. Just to highlight that the pointer passed to the `DlgProc` is a copy of the `ITEMPROPSHEETPAGE` used to create the property sheet page, I created a separate function to create the property sheet page, so that the `ITEMPROPSHEETPAGE` on the stack goes out of scope, making it clear that the copy passed to the `DlgProc` is not the one we passed to `CreatePropertySheetPage`.

Note that you must set the `dwSize` of the base `PROPSHEETPAGE` to the size of the `PROPSHEETPAGE` *plus* the size of your bonus data. In other words, set it to the size of your `ITEMPROPSHEETPAGE`.

```
int WINAPI WinMain(HINSTANCE hInst, HINSTANCE hPrevInst,
                   LPSTR lpCmdLine, int nCmdShow)
{
 g_hinst = hinst;
 HPROPSHEETPAGE hpage =
   CreateItemPropertySheetPage(42, TEXT("Elmo"));
 if (hpage) {
  PROPSHEETHEADER psh = { 0 };
  psh.dwSize = sizeof(psh);
  psh.dwFlags = PSH_DEFAULT;
  psh.hInstance = hinst;
  psh.pszCaption = TEXT("Item Properties");
  psh.nPages = 1;
  psh.phpage = &hpage;
  PropertySheet(&psh);
 }
 return 0;
}
```

Here is where we display the property sheet. It looks just like any other code that displays a property sheet. All the magic happens in the way we created the `HPROPSHEETPAGE`.

If you prefer to use the `PSH_PROPSHEETPAGE` flag, then the above code could have been written like this:

```
int WINAPI WinMain(HINSTANCE hInst, HINSTANCE hPrevInst,
                   LPSTR lpCmdLine, int nCmdShow)
{
 ITEMPROPSHEETPAGE psp;
 ZeroMemory(&psp, sizeof(psp));
 psp.dwSize = sizeof(psp);
 psp.hInstance = g_hinst;
 psp.pszTemplate = MAKEINTRESOURCE(1);
 psp.pfnDlgProc = DlgProc;
 psp.cWidgets = cWidgets;
 lstrcpyn(psp.szItemName, pszItemName, 100);
 PROPSHEETHEADER psh = { 0 };
 psh.dwSize = sizeof(psh);
 psh.dwFlags = PSH_PROPSHEETPAGE;
 psh.hInstance = hinst;
 psh.pszCaption = TEXT("Item Properties");
 psh.nPages = 1;
 psh.ppsp = &psp;
 PropertySheet(&psh);
 return 0;
}
```

If you want to create a property sheet with more than one page, then you would pass an array of `ITEMPROPSHEETPAGE`s. Note that passing an array requires all the pages in the array to use the same custom structure (because that's how arrays work; all the elements of an array are the same type).

Finally, here's the dialog template. Pretty anticlimactic.

```
1 DIALOG 0, 0, PROP_SM_CXDLG, PROP_SM_CYDLG
STYLE WS_CAPTION | WS_SYSMENU
CAPTION "General"
FONT 8, "MS Shell Dlg"
BEGIN
    LTEXT "Name:",-1,7,11,42,14
    LTEXT "",100,56,11,164,14
    LTEXT "Widgets:",-1,7,38,42,14
    LTEXT "",101,56,38,164,14
END
```

And there you have it. Tacking custom data onto the end of a `PROPSHEETPAGE`, an alternative to trying to cram everything into a single `lParam`.

Exercise: Observe that the size of the `PROPSHEETPAGE` structure has changed over time. For example, the original `PROPSHEETPAGE` ends at the `pcRefParent`. In Windows 2000, there are two more fields, the `pszHeaderTitle` and `pszHeaderSubTitle`. Windows XP added yet another field, the `hActCtx`. Consider a program written for Windows 95 that uses this technique. How does the shell know that the `cWidgets` is really bonus data and not a `pszHeaderTitle`?

Raymond Chen

Follow