How do I pass an array of variable-sized PROPSHEETPAGE structures to PropertySheet?

devblogs.microsoft.com/oldnewthing/20211125-00

November 25, 2021

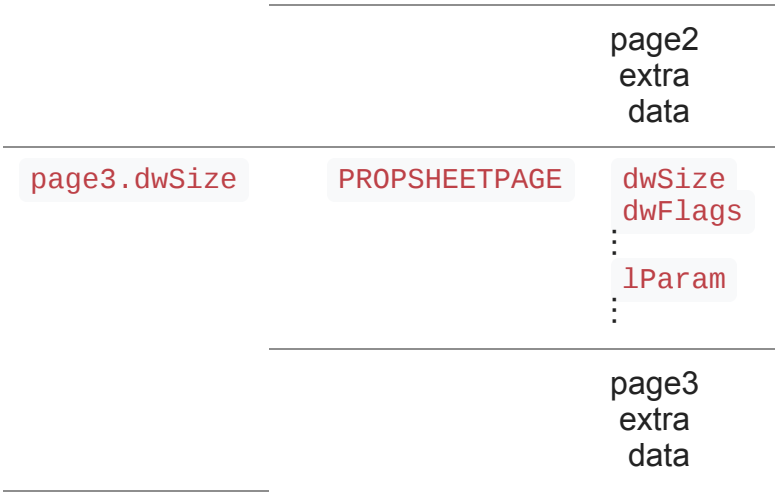
Raymond Chen

Last time, we noted that you can add your own custom data to the end of the `PROPSHEETPAGE`, and if you set the `dwSize` to include the custom data, then the system will copy that custom data into the `HPROPSHEETPAGE`.

This technique comes in handy if you need to create a property sheet page with `CreatePropSheetPage`, since it gives you a way to store more data than just the single `lParam` that comes with the `PROPSHEETPAGE` structure.

When you fill out a `PROPSHEETHEADER` structure, you can choose whether you're passing an array of `HPROPSHEETPAGE` handles (created by `CreatePropSheetPage`) or an array of `PROPSHEETPAGE` structures. Passing an array of `HPROPSHEETPAGE` handles isn't a problem, since all `HPROPSHEETPAGE` handles are the same size, regardless of the size of the `PROPSHEETPAGE` lurking inside them. But passing an array of variable-sized `PROPSHEETPAGE` structures is a trickier business.

What we want to do is lay out the memory like this:



We can do this by manufacturing a structure to hold the three extended `PROPSHEETPAGE` structures.

```
struct ThreePages
{
    Page1Data page1;
    Page2Data page2;
    Page3Data page3;
};

ThreePages pages;
```

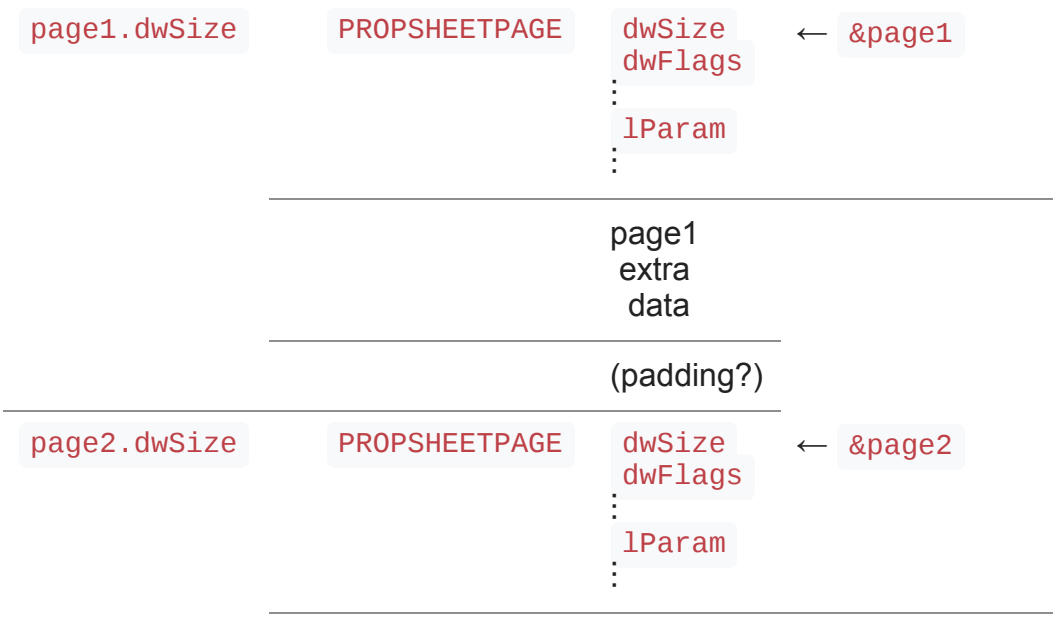
The naïve say of setting the `dwSize` members is to set each one to the size of the corresponding structure.

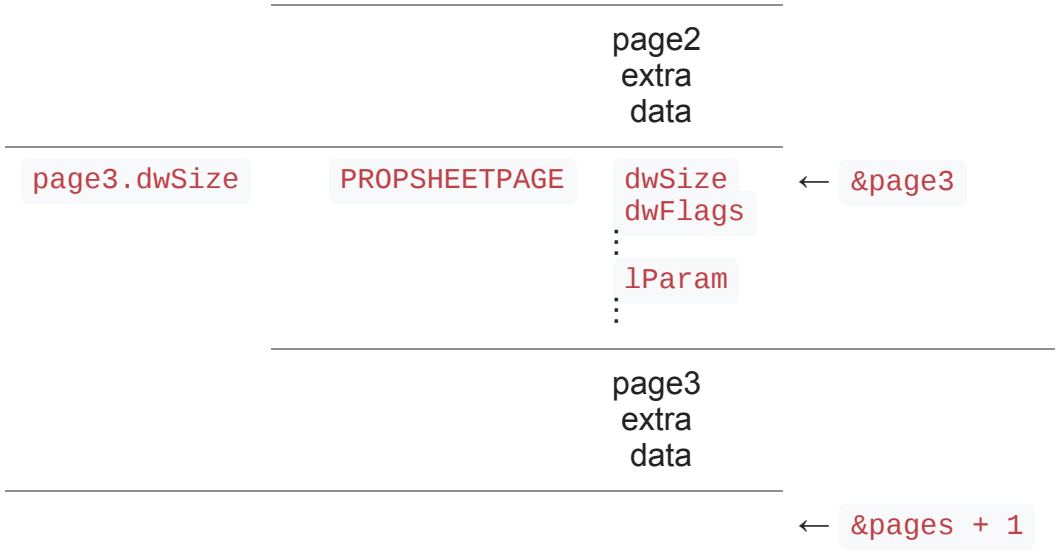
```
pages.page1.dwSize = sizeof(pages.page1);
pages.page2.dwSize = sizeof(pages.page2);
pages.page3.dwSize = sizeof(pages.page3);
```

This assumes that the three structures can be laid out next to each other without any inter-member padding. But that may not be true if the structures have different alignment requirements, say, if one of them contains a `__mi128`.

In the presence of padding, we have a shortfall between the size of each page and the start of the next page, resulting in an "oops" gap highlighted above.

In order to accommodate varying alignment requirements, the `dwSize` must include the padding so that the property sheet manager can find the next structure.[1] I've marked some key addresses in the diagram below:






```
pages.page1.dwSize = static_cast<DWORD>(
    reinterpret_cast<DWORD_PTR>(std::addressof(pages.page2)) -
    reinterpret_cast<DWORD_PTR>(std::addressof(pages.page1)));
pages.page2.dwSize = static_cast<DWORD>(
    reinterpret_cast<DWORD_PTR>(std::addressof(pages.page3)) -
    reinterpret_cast<DWORD_PTR>(std::addressof(pages.page2)));
pages.page3.dwSize = static_cast<DWORD>(
    reinterpret_cast<DWORD_PTR>(std::addressof(pages + 1)) -
    reinterpret_cast<DWORD_PTR>(std::addressof(pages.page3)));
```

This is quite a mouthful, but the idea is that we want to measure the distance to the next thing. We use `std::addressof` instead of the traditional `&` operator to protect against the possibility that the `&` operator has been overloaded.[2]

Yes, this is quite annoying, but it's also probably not something you're likely to be doing, because you could just use a pointer to a stack-allocated object which will remain valid until `PropertySheet` returns. The main value of the `PROPSHEETPAGE` payload is in the case where you need to produce an `HPROPSHEETPAGE`, since the `HPROPSHEETPAGE` is probably going to outlive any stack variables.

But it's there if you need it.

[1] Don't even think of using `#pragma pack(1)` to remove the padding. This will misalign the next structure and result in crashes on alignment-sensitive platforms.

[2] Overloading the `&` operator is something that annoys C++ library authors, although it's still nowhere as annoying as overloading the comma operator.

Raymond Chen

Follow