How can I detect from the preprocessor what a macro's definition is?

devblogs.microsoft.com/oldnewthing/20180628-00

June 28, 2018

Raymond Chen

It is common that a preprocessor macro chooses between multiple behaviors based on various other controlling macros.

```
#ifdef BUILD_DLL
#define CONTOSOAPI __declspec(dllexport)
#else
#define CONTOSOAPI __declspec(dllimport)
#endif

#ifdef USE_STDCALL
#define CONTOSOAPICALL __stdcall
#else
#define CONTOSOAPICALL __cdecl
#endif
```

Suppose you want to check at compile time how these macros are defined. Is there a way to do string comparisons in the preprocessor, something like this?

```
#if somethingsomething CONTOSOAPI == __declspec(dllexport)
```

I'll get it out of the way up front: Instead of trying to parse the value of a macro, you can just replicate the conditionals that led to the macro's definition. In other words, you can do this:

```
#ifdef BUILD_DLL
... stuff to do when CONTOSOAPI is __declspec(dllexport)
#else
... stuff to do when CONTOSOAPI is __declspec(dllimport)
#endif

#ifdef USE_STDCALL
... stuff to do when CONTOSOAPICALL is __stdcall
#else
... stuff to do when CONTOSOAPICALL is __cdecl
#endif
```

But let's say that this option isn't available. For example, maybe the logic that eventually leads to the definition of `CONTOSOAPI` is super-complicated and difficult to replicate. Or the header file is not under your control and you want your code to adapt to newer versions of the header file that may use different logic to decide what definition to use.

The C and C++ preprocessors do not do string comparisons. All they can do is evaluate constant integral expressions. So things don't sound good.

But wait, maybe we can trick them into evaluating constant integral expressions!

```
#define __declspec
#define dllexport 1
#define dllimport 2

#if CONTOSOAPI == __declspec(dllexport)
... stuff to do when CONTOSOAPI is __declspec(dllexport)
#elif CONTOSOAPI == __declspec(dllimport)
... stuff to do when CONTOSOAPI is __declspec(dllimport)
#else
#error I don't know what CONTOSOAPI is defined as
#endif

#undef dllimport
#undef dllexport
#undef __declspec

#define __stdcall 1
#define __cdecl 2

#if CONTOSOAPICALL == __stdcall
... stuff to do when CONTOSOAPICALL is __stdcall
#elif CONTOSOAPICALL == __cdecl
... stuff to do when CONTOSOAPICALL is __cdecl
#else
#error I don't know what CONTOSOAPICALL is defined as
#endif

#undef __cdecl
#undef __stdcall
```

By redefining the words that appear in the `CONTOSOAPI` and `CONTOSOAPICALL` macros, you can turn the text into integer constant expression. After macro expansion, `__declspec(dllexport)` becomes `(1)`, and `__declspec(dllimport)` becomes `(2)`. These are integer constant expressions that can be evaluated by the preprocessor!

Why did I choose `1` and `2` as the integer constants rather than `0` and `1`? One of the rules of the C and C++ preprocessors is that after macro substitution, if there are any identifiers remaining whose values are not known, then they are treated as zero. This means

that `__declspec(magicbeans)` expands to `(magicbeans)`, and since there is no definition for `magicbeans`, the preprocessor treats it as zero. If I had defined `dllexport` as `0`, then I would misdetect `__declspec(magicbeans)` as `dllexport`.

This is extreme abuse of the C and C++ preprocessor. But desperate times may call for desperate measures.

Bonus chatter: Note that this trick requires that you find some way of defining symbols so that what remains is a valid integer constant expression.

Raymond Chen

Follow