

Re: Convert C-Builder program to Delphi?

Source: <http://coding.derkeiler.com/Archive/Delphi/comp.lang.pascal.delphi.misc/2005-01/0527.html>

From: W. D. (NewsGroups_at_US-Webmasters.com)

Date: 01/28/05

Date: Fri, 28 Jan 2005 00:42:01 -0600

Maarten Wiltink wrote:

>
> "W. D." <NewsGroups@US-Webmasters.com> wrote in message
> news:41F8884A.539E@US-Webmasters.com...
>
>> [...] Right now this C++ code that I am working on is getting
>> bizarre. As far as I can tell, the following code is for
>> compatilby with DLLs.
>>
>> I Googled just about everyplace I can think of and still
>> haven't come up with the proper way to translate these
>> to Delphi:
>>
>> -----
>> #if !defined (DLLFUNC)
>> #define DLLFUNC(ret) extern __declspec(dllexport) ret __stdcall
>> #endif
>>
>> DLLFUNC(int) SomeDLL_Open();
>>
>> DLLFUNC(T) more or less wraps type T so that declaring a function
>> to return it is also usefully exported from a DLL this source is
>> linked into.
>>
>> It is a macro and depends on C preprocessor macros being textual;
>> also on the syntactical closeness of the relevant C keywords.
>> Delphi does not have them all next to each other and doesn't have
>> macros anyway.
>>
>> Marking functions as exported is done in the project file in Delphi,
>> quite separate from the function declaration itself. The calling
>> convention is given by a directive added after the declaration but
>> it marks up the declaration as a whole, not just the type. (That's
>> true also for C, but macros don't care.)
>>
>> Look for "exports" and "stdcall" in the help.
>>
>> Groetjes,

comp.lang.pascal.delphi.misc: Re: Convert C-Builder program to Delphi?

> Maarten Wiltink

At 11:53 1/27/2005, Stephen Posey wrote:

>> *Oh boy. This C++ code that I am working on is getting*

>> *bizzare. As far as I can tell, this code is for compatibilty with DLLs.*

>>

>> *I Googled just about everyplace I can think of and still haven't come up*

>with

>> *the proper way to translate these to Delphi:*

>> ~~~~~

>> *#if !defined (DLLFUNC)*

>> *#define DLLFUNC(ret) extern __declspec(dllexport) ret __stdcall #endif*

>>

>>

>> *DLLFUNC(int) SomeDLL_Open();*

>>

>> *DLLFUNC(void) SomeDLL_Close();*

>>

>> *DLLFUNC(int) SomeDLL_GetData(BYTE* pDataType, const PBYTE* pTheData,*

>*DWORD* pDataSize);*

>>

>> *DLLFUNC(int) SomeDLL_Request(BYTE byRequestType, DWORD dwRequestData);*

>>

>> *// The result returned is based upon the last function attempted within*

>*this DLL*

>> *DLLFUNC(int) SomeDLL_GetErrorCondition();*

>>

>> *typedef int (__stdcall *FUNC_OPEN)();*

>> *typedef void (__stdcall *FUNC_CLOSE)();*

>> *typedef int (__stdcall *FUNC_GET)(BYTE*,PBYTE*,DWORD*);*

>> *typedef void (__stdcall *FUNC_SHOWICON)(BOOL);*

>>

>>

>> *#endif*

>~~~~~

>>

>> *Can anybody spare a hint?*

>

>*This appears to me to be an overly cute approach of defining a*

>*consistent interface onto some exported DLL functions. I guess it might*

>*have some benefit if the calling signature is expected to change, this*

>*way the only place anything needs to be changed is in the #define macro.*

>

>*The thing to remember about C/C++ preprocessor #defines is that it's*

>*really just a TEXT substitution mechanism. Since Delphi doesn't have*

>*anything quite like this, the usual approach is to expand the #define*

>*into actual Delphi code; sometimes this comes out as a CONST*

>*declaration, sometimes it converts to a function, in this case it*

>*decorates some function declarations elsewhere in your code.*

>

>*So, what is this actually saying?:*

>

Re: Convert C-Builder program to Delphi?

comp.lang.pascal.delphi.misc: Re: Convert C-Builder program to Delphi?

> > #define DLLFUNC(ret) extern __declspec(dllexport) ret __stdcall
>
> It says whenever the preprocessor encounters the text string "DLLFUNC",
> replace it with the defined text. the "(ret)" acts as a parameter,
> saying whatever text appears in parentheses after "DLLFUNC" should be
> inserted wherever "ret" appears in the macro string.
>
> So lets get down to cases, first a step-by-step substitution so you can
> see how that works.
>
> > DLLFUNC(int) SomeDLL_Open();
>
> 1. We found DLLFUNC so insert its definition at that location in the source:
>
> extern __declspec(dllexport) ret __stdcall
>
> 2. Append the original call
>
> extern __declspec(dllexport) ret __stdcall SomeDLL_Open();
>
> 3. Now we passed in "int" as the parameter, so replace any instances of
> "ret" with "int":
>
> extern __declspec(dllexport) int __stdcall SomeDLL_Open();
>
> So, that's what the code looks like at after the first macro pass. Now,
> "__declspec" is a standard C++ macro, read about __declspec here:
>
> http://msdn.microsoft.com/library/default.asp?url=/library/en-us/vccore/html/_core_export_from_a_dll_using___declspec.28.dllexport.29.asp
>
> But given what it does, I don't think you need to worry too much about
> its exact expansion. My first guess at a Delphi translation:
>
> function SomeDLL_Open(): Integer; stdcall; name 'SomeDLL_Open';
>
> This declaration would go in a "Exports" section of your DLL.
>
> Similarly:
>
> > DLLFUNC(void) SomeDLL_Close();
>
> extern __declspec(dllexport) void __stdcall SomeDLL_Close();
>
> > DLLFUNC(int) SomeDLL_GetData(BYTE* pDataType, const PBYTE* pTheData,
> > DWORD* pDataSize);
>
> extern __declspec(dllexport) int __stdcall SomeDLL_GetData(
> BYTE* pDataType, const PBYTE* pTheData, DWORD* pDataSize);
>
> > DLLFUNC(int) SomeDLL_Request(BYTE byRequestType, DWORD dwRequestData);

```
>  
> extern __declspec(dllexport) int __stdcall SomeDLL_Request(  
> BYTE byRequestType, DWORD dwRequestData);  
>  
> > DLLFUNC(int) SomeDLL_GetErrorCondition();  
>  
> extern __declspec(dllexport) int __stdcall SomeDLL_GetErrorCondition()  
>  
>HTH
```

Yes, it sure did—Thanks Stephen. (thanks Maarten, and Eugenijus too!)

What I came up with that compiled is:

```
// Trashed the macro, and manually constructed the statements...  
  
// C++: DLLFUNC(int) SomeDLL_Open();  
Function SomeDLL_Open(): Integer; StdCall;  
Exports SomeDLL_Open Name 'SomeDLL_Open';  
  
// Apparently, the 'Exports' keyword can be placed anywhere, so  
// why not just after the Function is declared?  
  
// C++: DLLFUNC(void) SomeDLL_Close();  
Procedure SomeDLL_Close(); StdCall;  
Exports SomeDLL_Close Name 'SomeDLL_Close';  
  
// C++: DLLFUNC(int) SomeDLL_GetData(BYTE* pDataType,  
// const PBYTE* pTheData, DWORD* pDataSize);  
Function SomeDLL_GetData(pDataType: Byte; const pTheData: PBYTE;  
    pDataSize: LongInt): Integer; StdCall;  
Exports SomeDLL_GetData NAME 'SomeDLL_GetData';  
  
// C++: DLLFUNC(int) SomeDLL_Request(BYTE byRequestType, DWORD  
dwRequestData);  
Function SomeDLL_Request(byRequestType: Byte; dwRequestData: LongInt):  
Integer; StdCall;  
Exports SomeDLL_Request NAME 'SomeDLL_Request';
```

Now I have a few more questions: Even though this format compiles, will I have any problems with the pointers that I pretty much ignored in these translations? If so, how should I alter this code?

Also, Type definitions 1, & 2, below seem to make sense, but #3 has multiple data types as arguments. And #4 has that extra parameter, 'BOOL'. What should I do in this case?

```
// (1) C++: typedef int (__stdcall *FUNC_OPEN)();  
Type FUNC_OPEN = Function(): Integer; StdCall;
```

comp.lang.pascal.delphi.misc: Re: Convert C-Builder program to Delphi?

```
// (2) C++: typedef void (__stdcall *FUNC_CLOSE)();  
Type FUNC_CLOSE = Function(): Integer; StdCall;
```

```
// (3) C++: typedef int (__stdcall *FUNC_GET)(BYTE*,PBYTE*,DWORD*);  
Type FUNC_GET = Function({ ???? }): Integer; StdCall;
```

```
// (4) C++: typedef void (__stdcall *FUNC_SHOWICON)(BOOL);  
Type FUNC_SHOWICON = Procedure(); StdCall;
```

Thanks for all the light you guys are shining my way!