Configuring the Microsoft Visual Studio IDE to build Custom Modules

Copyright © 2012, Microstar Laboratories, Inc.

If you typically use the Microsoft Visual Studio[®] environment to build applications, you might prefer to build DAPL custom modules in that same environment. The environment is designed to make it easy to build applications that run in the Windows[®] system, and it does a remarkable job of providing all of the settings you would normally need for this. However, since downloadable binary modules for the DAPL system do not run in the Windows host system environment, you will need to make many adjustments.

Preparing your system

First, install your compiler tools. Verify that the compiler system works, and you can configure and build the examples that come with the compiler.

Second, run the DAPtools Professional software installer and select the "DAPL Command Development" item. This installation process will look in your system, identify the compiler version that you are using, and set up a compatible set of library and configuration options. It will also set up a copy of several custom command examples. Even if you don't plan to use the Windows command line environment, you should run the test that follows.

To test the command line environment, go to the main Windows menu and select

Microstar Laboratories | DTD | MSL DTD command window

Use the CD command to locate the folder with your copy of the example files. The exact location of these files in the "user documents area" will depend on the version of the Windows system that you are using.

CD "\Users\<username>\Documents\Microstar Laboratories\DTD\Examples"

Then type in the following command line.

```
NMAKE -f MODMAKEM.MAK
```

This test verifies that your DTD interface code is correctly installed where your compiler system can find it. A downloadable module file should be successfully built for each of the example C++ files. If this test fails, there is very little chance that you will get useful results from the graphical IDE environment.

Running the "New Project" Setup wizard

From the Visual Studio main menu select File | New | Project... and pick the Project type Win32. Click to select the Win32 Project installed template. Specify a name and location for your project files.

New Project					? 🔀
Project types:		Iemplates:	[NET Framework 3 5	-
Visual C++ ATL CLR General MFC Test Instal/Shield Other Projects A project for creat	Express t Types ;	Visual Studio installed ten Win32 Console Win32 Project Application My Templates Search Online Templates	nplates		
Name:	<enter_name></enter_name>				
Location:	C:\Dev\JMD.2008\	DapTest8ed2		~	Browse
Solution Name:	<enter_name></enter_name>		Create girectory	y for solution Control	
				ОК	Cancel

Click OK and continue to the next screen. Specify that you are building a DLL starting from an empty project.

Win32 Application Wizard - DLM 🔹 👔			
Applic	ation Settings		
Overview	Application type:	Add common header files for:	
Application Settings	<u>W</u> indows application	ATL	
	Console application	L_ MFC	
	⊙ Brr		
	Static library		
	Additional options:		
	Empty project		
	Export sympols		
	Precomplied neader		
	< Previous	Next > Finish Cancel	

When the wizard is finished, you need to set up an initial C^{++} code file. Go to the main menu and pick File | New | File. Select Visual C++ language, and the Visual Studio C++ File template.

New File			<u>?</u> ×
Categories: General	Templates: Visual Studio installed templates C++ File (.cpp) Midl File (.idl) Cursor File (.cur) Resource Template File (.rct)	 h Header File (.h) 	<u><u>B</u></u> <u>B</u> <u>B</u> <u>B</u> <u>B</u> <u>B</u> <u>B</u> <u>B</u> <u>B</u> <u>B</u>
Creates a file containing C++ source coo	le		
		<u>O</u> pen •	Cancel

You will get an empty code page. Insert the following "generic" code in that page.

```
#include "DTDMOD.H"
#include "DTD.H"
#define COMMAND "CC"
#define ENTRY CC_entry
int __stdcall ENTRY (PIB **plib)
{
    return 0;
}
extern "C" __declspec(dllexport) int __stdcall ModuleInstall(void *hModule)
{
    return (CommandInstall(hModule, COMMAND, ENTRY, NULL));
}
```

For the "CC" items appearing in the code above, substitute the name of the custom command as you want it to appear in your DAPL configuration scripts later. The command name *must not* be the same as the name of the project, which will determine the name assigned to the final DLM file. If the names were the same, the DAPL system could not distinguish the module file from the command inside of it.

Save this file in your project folder.

Now add this file to the *Solution Explorer* by right-clicking in the *Solution Explorer* window (usually at the left margin of the display) and picking Add | New item. The file name will then appear in the *Solution Explorer* pane. Double click on the file name to open it.

General project settings

Open the project. Select the Project | Properties dialogs from the main menu. Click on Configuration Properties and General. The first screen will verify the general project properties that you entered in the wizard when first setting up the new project.

Configuration: Active(Release)	1	Platform: Active(Win32)	Configuration Manager
Common Properties		General	
Configuration Properties		Output Directory	\$(SolutionDir)\$ConfigurationName)
General		Intermediate Directory	\$(ConfigurationName)
Debugging		Extensions to Delete on Clean	*.obj;*.ilk;*.tlb;*.tli;*.tlh;*.tmp;*.rsp;*.pgc;*.pgd;*.meta;\$
C/C++		Build Log File	\$(IntDir)\BuildLog.htm
Linker		Inherited Project Property Sheets	
Manifest Tool		Enable Managed Incremental Build	Yes
XML Document Generator		Project Defaults	
Browse Information		Configuration Type	Dynamic Library (.dll)
Build Events		Use of MFC	Use Standard Windows Libraries
Custom build Step		Use of ATL	Not Using ATL
		Character Set	Not Set
		Common Language Runtime support	No Common Language Runtime support
		Whole Program Optimization	No Whole Program Optimization
	Ir Sp	ntermediate Directory becifies a relative path to the intermediate	file directory; can include environment variables.

Verify the DLL output file format. The DAPL system uses 8-bit characters, so you will most likely prefer to avoid multi-character or UTF8 character coding.

There is no Windows debugging for downloadable modules.

Click the Configuration Manager button in the upper right corner. In the drop-down dialog, click the Edit... option.

Configuration Manager			? ×
Active solution configuration:	Active solu	tion <u>p</u> latform:	
<edit></edit>	Win32		-
Debug Release	r deploy):		
<new></new>		Platform	Build
<edit></edit>	-	Win32 🔻	
			Close

In the dialog that follows, disable the debug configuration option - this does not apply to code that runs in the embedded DAPL environment.

Compiler settings

Select Project | Properties from the main application menu. In the pane at the left, select the Configuration Properties and then C/C++ - General.

Configuration: Active(Release)	Platform: Active(Win32)	Configuration Manager
Common Properties	Additional Include Directories	"C:\Program Files (x86)\Microstar Laboratories\DTI
 Configuration Properties 	Resolve #using References	
General	Debug Information Format	Disabled
Debugging	Suppress Startup Banner	Yes (/nologo)
⊡ C/C++	Warning Level	Level 3 (/W3)
General	Detect 64-bit Portability Issues	No
Optimization	Treat Warnings As Errors	No
Preprocessor	Use UNICODE Response Files	No
Code Generation		
Language		
Precompiled Headers		
Output Files		
Browse Information		
Advanced		
Command Line		
🛨 Linker		
Manifest Tool		
Regisso Information		
Build Events		
Oustom Build Step		
	Additional Include Directories	
	Specifies one or more directories to add to (/I[path])	the include path; use semi-colon delimited list if more than one.

- Warning level 3 is optional, but most downloadable files can compile free of warning messages at this rigorous level. Since debugging can become difficult for software downloaded to an embedded environment, you will probably want as many early warnings as possible.
- Do not use 64-bit Windows detection.

Click the Additional Include Directories item and then the button that appears at the right. Click the "open folder" icon to add a new line, then click the button that appears on the right to locate the directory. Uncheck the Inherit from parent or project defaults box at the lower left. Unless you have changed the default install location of the DAPL software, your will add a new line that specifies the location

<u>C:\Program Files\Microstar Laboratories\DTD\INCLUDE</u> as in the following illustration.

Additional Include Directories	<u>? ×</u>
C: \Program Files (x86) \Microstar Laboratories \DTD \INCLUDE	_
	<u> </u>
Inherited values:	
	<u>^</u>
	_
Inherit from parent or project defaults	Macros>>
	OK Cancel

When you click OK here, the line you added will be inserted into the C++ general configuration dialog.

Next, click on the Optimization item in the Configuration properties column.

Configuration: Active(Release)	Platform: Active(Win32)	Configuration Manager
Common Properties	Optimization	Maximize Speed (/02)
Configuration Properties	Inline Function Expansion	Default
General	Enable Intrinsic Functions	No
Debugging	Favor Size or Speed	Neither
□ C/C++	Omit Frame Pointers	No
General	Enable Fiber-safe Optimizations	No
Optimization	Whole Program Optimization	No
Code Generation		
Language		
Precompiled Headers		
Output Files		
Browse Information		
Advanced		
Command Line		
E Linker		
Manifest Tool		
XML Document Generator		
Browse Information		
Build Events		
Custom Build Step		
	Optimization Gelect option for code optimization; choose Cu Ox)	istom to use specific optimization options. (/Od, /O1, /O2,

- The Maximize Speed option is suggested, but other optimization choices may also work fine.
- There are generally no problems with inline function expansion.
- Do not allow "intrinsic functions". These will conflict with "standard" math library functions.

Click on the Preprocessor item in the Configuration properties column. Click on the Processor Definitions item and then on the button that appears on the right.

Configuration: Active(Release)	Platform: Active(Win32)	Configuration Manager
Common Properties	Preprocessor Definitions	WIN32;NDEBUG; WINDOWS;
Configuration Properties	Ignore Standard Include Path	No
General	Generate Preprocessed File	No
Debugging	Keep Comments	No
□ C/C++		
General		
Optimization		
Preprocessor		
Code Generation		
Language		
Precompiled Headers		
Output Files		
Browse Information		
Advanced		
Command Line		
Linker		
Manifest Tool		
XML Document Generator		
Browse Information		
Build Events		
Custom Build Step		
	Preprocessor Definitions	
	Specifies one or more preprocessor defines.	(/D[macro])

Click the Preprocessor Definitions item, and then the button that appears at the right.

Preprocessor Definitions	? ×
_CRT_SECURE_NO_DEPRECATE	1
Therited values:	V
WINDLL	<u> </u>
	T
Inherit from parent or project defaults	cros>>
OK C	Cancel

- Clear the check box Inherit from parent or project defaults in the lower left corner.
- Click in the text window and type in the preprocessor macro _CRT_SECURE_NO_DEPRECATE as shown. Though not absolutely required, this macro is strongly recommended. The purpose is to turn off stack intrusion checks. This hazard does not exist unless an attacker has already overtaken your host system. Under healthy operation, the usual unrestricted stack diagnostics can by themselves produce processor overload hazards.

Click on the Code Generation item in the Configuration properties column.

Common Properties Enable String Pooling No Framework and References Enable Minimal Rebuild No Configuration Properties Enable C++ Exceptions No General Smaller Type Check No	Configuration: Active(Release)	Platform: Active(Win32)	Configuration Manager
Debugging Default C/C++ Basic Runtime Checks Default Optimization Preprocessor Code Generation Language Precompiled Headers Output Files Browse Information Advanced Command Line Enable Floating Point Exceptions Image: Linker Manifest Tool Manifest Tool XML Document Generator Build Events Enable Floating Point Exceptions Image: Recent Struct Kuntime Library Build Events Runtime Library Struct Member Alignment Default Browse Information Advanced Command Line Enable Floating Point Exceptions Image: Recent Struct Floating Point Exceptions Build Events Floating Point Point Exceptions Image: Recent Struct Floating Point Po	 Common Properties Framework and References Configuration Properties General Debugging C/C++ General Optimization Preprocessor Code Generation Language Precompiled Headers Output Files Browse Information Advanced Command Line Linker Manifest Tool XML Document Generator Browse Information Build Events Custom Build Step 	Enable String Pooling Enable Minimal Rebuild Enable C++ Exceptions Smaller Type Check Basic Runtime Checks Runtime Library Struct Member Alignment Buffer Security Check Enable Function-Level Linking Enable Enhanced Instruction Set Floating Point Model Enable Floating Point Exceptions	No No No No Default Multi-threaded (/MT) Ves No Not Set Strict (/fp:strict) No Td, /MD, /MDd)

- Disable exceptions. The exception handling library functions are Windows system dependent.
- Use the /MT Multi-Threaded Application library option. This might seem peculiar using an "application library" to produce "DLL code" but DLL features that the Windows system uses are not appropriate for the embedded DAPL system. The DTD installation instructions might tell you to avoid multi-threaded libraries, but that advice applies only to some earlier compiler versions.
- Buffer security check. This is the equivalent of the /GS option, and it is allowable.
- Other floating point model options should also work.

Click on the Language item in the Configuration properties column.

Configuration: Active(Release)	Platform: Active(Win32)	Configuration Manager
Common Properties	Disable Language Extensions	No
Configuration Properties	Default Char Unsigned	No
General	Treat wchar_t as Built-in Type	Yes
Debugging	Force Conformance In For Loop Scope	Yes
□ C/C++	Enable Run-Time Type Info	No (/GR-)
General	OpenMP Support	No
Optimization		
Preprocessor		
Code Generation		
Language		
Precompiled Headers		
Output Files		
Browse Information		
Advanced		
Command Line		
Manifest Tool		
XML Document Generator		
Browse Information		
Build Events		
Custom Build Step		
	Disable Language Extensions	
	Suppresses or enables language extensions.	(/Za)

Disable the Runtime Type Info feature. This calls library functions that are Windows system dependent.

Next, some optional diagnostic listings are available under the Output Files item in the Configuration Properties column.

Configuration: Active(Release)	Platform: Active(Win32)	Configuration Manager
Common Properties	Expand Attributed Source	No
Configuration Properties	Assembler Output	Assembly With Source Code (/FAs)
General	ASM List Location	\$(IntDir)\
Debugging	Object File Name	\$(IntDir)\
C/C++	Program Database File Name	\$(IntDir)\vc90.pdb
General	Generate XML Documentation Files	No
Optimization	XML Documentation File Name	\$(IntDir)\
Preprocessor		
Code Generation		
Language Drocompiled Headorn		
Output Files		
Browse Information		
Advanced		
Command Line		
Linker		
Manifest Tool		
XML Document Generator		
Browse Information		
Build Events		
Custom Build Step		
	Assessblay Output	
	Assembler Output	
	specifies the contents of assembly language	ge output life. (FAC, FAC, FACS)

The optional Assembly with Source Code (/FAs) option produces a listing that is sometimes useful for finding the cause of unexpected external function calls.

 $Click \ on \ the \ \mbox{Advanced} \ item \ in \ the \ \mbox{Configuration} \ \ \mbox{properties} \ column.$

Configuration: Active(Release)	Platform: Active(Win32)	Configuration Manager
 Common Properties Configuration Properties General Debugging C/C++ General Optimization Preprocessor Code Generation Language Precompiled Headers Output Files Browse Information Advanced Command Line Linker Manifest Tool XML Document Generator Browse Information Build Events Custom Build Step 	Calling Convention Compile As Disable Specific Warnings Force Includes Force #using Show Includes Undefine Preprocessor Definitions Undefine All Preprocessor Definitions Use Full Paths Omit Default Library Names Error Reporting	r application (can be overridden by function). (/Gd, /Gr,

- The Developer's Toolkit for DAPL functions use cdecl calling conventions. For other calling conventions, you can use the __stdcall declarations in the code files where needed.
- Compile as C++ (/TP option). If you prefer C, code as if using the C compiler and you will rarely notice any difference. The Developer's Toolkit functions use no special C++ notations (no classes, no function overloading, etc.)
- Omit embedded library names using the / Zl option. Ordinarily, compiled code will contain a long list of library files, most of which can do nothing but cause trouble for your module

Linker properties

Next, you will need to configure the linker properties. In the Configuration Properties column, select Linker and then General.

Configuration: Active(Release)	Platform: Active(Win32)	Configuration Manager
Configuration Properties	Output File	\$(OutDir)\\$(ProjectName).dlm
General	Show Progress	Not Set
Debugging	Version	
C/C++	Enable Incremental Linking	No (/INCREMENTAL:NO)
General	Suppress Startup Banner	Yes (/NOLOGO)
Optimization	Ignore Import Library	No
Preprocessor	Register Output	No
Code Generation	Per-user Redirection	No
Language	Additional Library Directories	"C:\Program Files (x86)\Microstar Laboratories\DTI
Output Files	Link Library Dependencies	Yes
Browse Information	Use Library Dependency Inputs	No
Advanced	Use UNICODE Response Files	No
Command Line		
Linker		
General		
Input		
Manifest File		
Debugging		
System		
Optimization		
Embedded IDL		
Advanced	Use UNICODE Response Files	
Command Line 🗨	property to "Yes" when files in the project have UNICODE paths.	
•		

- Modify the name of the output file type to dlm ("downloadable module") rather than the usual dll.
- Incremental linking will typically be suppressed anyway because of other options disabling it will avoid warning messages.

While still in the Linker – General section, click the Additional Library Directory item and then click the button that appears to the right. In the dialog box, click on the "new folder" icon and then on the button that appears to the right. Locate the folder where the special Developer's Toolkit for DAPL library files are located. If you installed in the default location, your added line will look like the following.

<u>?</u> ×
<u>➡ × + +</u>
A
<u> </u>
V
Macros>>
OK Cancel

- Click to disable the check box at the lower left to Inherit from parent or project defaults
- Click the "new folder" icon at the top and navigate to the location shown. The location might be different if you did not install in the default location.

Select the Input item under Linker. Here you will make some very important adjustments. The linker *must* check the special Developer's Toolkit for DAPL libraries first, before trying to resolve function calls with any other libraries. If any library functions carry additional links to features of the Windows environment, your module will not link and you will be faced with some very mysterious messages reporting unresolved references.

Configuration: Active(Release)	Platform: Active(Win32)	Configuration Manager
 Common Properties Configuration Properties General Debugging C/C++ Linker General Innut Manifest File Debugging System Optimization Embedded IDL Advanced Command Line Manifest Tool XML Document Generator Browse Information Build Events Custom Build Step 	Additional Dependencies Ignore All Default Libraries Ignore Specific Library Module Definition File Add Module to Assembly Embed Managed Resource File Force Symbol References Delay Loaded DLLs Assembly Link Resource	DAPL.LIB MODCRTO.LIB \$(NOINHERIT) No
	Additional Dependencies Specifies additional items to add to the li	nk line (ex: kernel32.lib); configuration specific.

Click on the Additional Libraries item. Click on the button that appears at the right. This dialog tells the Linker about the existence of the two special Developer's Toolkit for DAPL libraries, and forces these libraries to be scanned before any other library files.

Additional Dependencies	? ×
DAPL.LIB MODCRT0.LIB	4
	V
Inherited values:	
kernel32.lib user32.lib gdi32.lib winspool.lib	
	_
Inherit from parent or project defaults	Macros>>
	OK Cancel

- Very important, click to uncheck the Inherit from parent or project defaults box. Any of the libraries you see listed in the central display pane can cause unresolvable link errors.
- Enter the two special library names as shown. The name MODCRT0 contains the letter 0 and ends with numerical digit 0.
- The compiler is allowed to resolve function calls by searching its run-time library, but be aware that depending on what is called, this could result in unresolvable and seemingly unrelated link problems.

The linker debugging is not interactive, and produces log files. These options are safe to use. Select the Input item under Linker.

Configuration: Active(Release)	Platform: Active(Win32)	Configuration Manager
Common Properties	Generate Debug Info	No
 Configuration Properties 	Generate Program Database File	
General	Strip Private Symbols	
Debugging	Generate Map File	Yes (/MAP)
	Map File Name	
Linker	Map Exports	No
General	Debuggable Assembly	No Debuggable attribute emitted
Input		
Manifest File		
Debugging		
System		
Optimization		
Embedded IDL		
Advanced		
Command Line		
Manifest Tool		
XML Document Generator		
Browse Information		
Custom Build Step		
	Generate Debug Info	
	Enables generation of debug information.	(/DEBUG)
1		

• The optional linker MAP file is sometimes useful to identify how library function calls were resolved when the module was built.

In the Configuration Properties column, select Advanced. You can verify that the code is generated for the MachineX86 architecture. It would be a good idea to disable the Manifest inclusion, since this is not used, and wastes memory.

Finishing and Compiling

After completing your project configuration, save your project. Now you can use the *Solution Explorer* to begin your normal code development work, starting from the initial code page that you set up, and adding any additional code files that you need. When your code is ready, use the main Build menu to build your project code and produce the downloadable command module.