

TN253

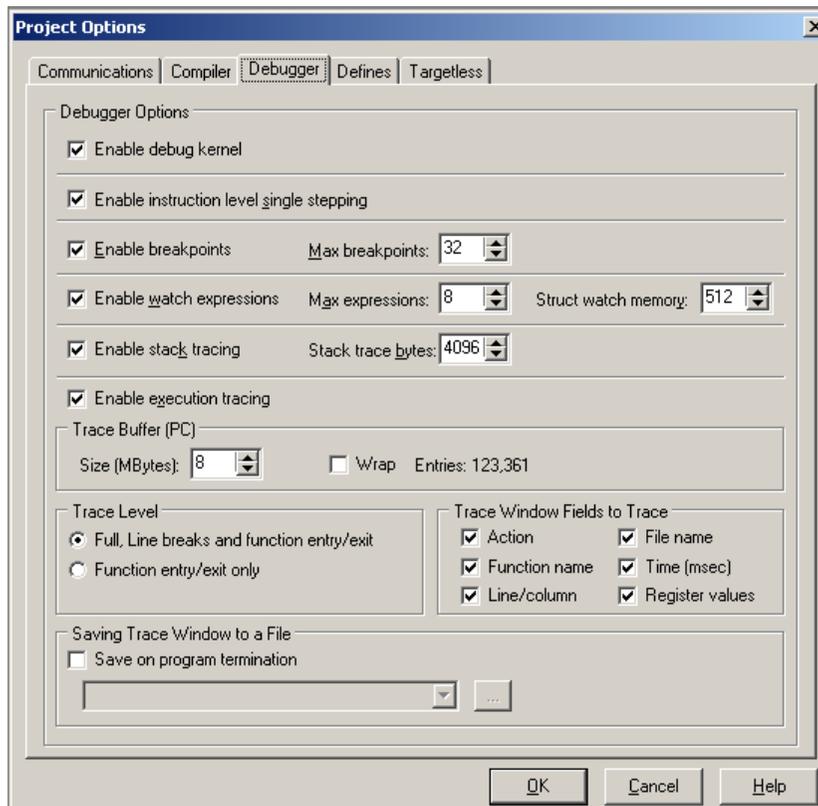
Execution Tracing

Execution tracing is an advanced debugging feature available with Dynamic C 9 through 10.11. It is not supported in later versions of Dynamic C. Execution tracing allows examination of the flow of a program’s execution in real time instead of single stepping through it. The Trace window can show which statement was executed, what type of action occurred, when the action was executed, and the contents of the registers after executing it. The contents of the Trace window can also be saved to a file for later access.

Enabling Execution Tracing

To use execution tracing it may be selected on the Debugger tab of the Project Options menu. If “Enable execution tracing” is checked, the target will send trace information back to Dynamic C when you turn on tracing by choosing Inspect | Start Tracing or when your program does so by executing a `_TRACEON` macro. Unchecking this box will disable the menu command and macro.

Figure 1. Dynamic C Menu: Options | Project Options



The macro `_TRACE` is another option for enabling execution tracing, albeit in a limited way. Read more about the `_TRACE` macro in the sections titled [“Starting and Stopping Execution Tracing”](#) and [“Using Trace Macros”](#).

Configuration Options

The Debugger tab is also where you may configure execution tracing parameters such as the buffer used to hold trace entries.

Trace Buffer (PC)

This section is where you specify how much memory is allocated on the host PC for trace entries received from the target. The default is 64 MB. If you check the “Wrap” box, new trace entries overwrite existing ones when the buffer fills up, starting with the oldest. When “Wrap” is unchecked, any entries received after the buffer fills up are discarded.

The number of entries displayed is the maximum number of trace entries the buffer will hold given the size of the trace buffer you specify and the Trace window information fields you select.

Trace Level

This section is where you specify which events will be captured by the trace. Full tracing captures all debuggable statements plus function entries and exits. If you don’t want to include all statements, you can choose to capture each function entry and exit only.

Dynamic C statements are debuggable by default, while assembly code is not. You can toggle this with the `debug` and `nodebug` keywords for Dynamic C functions, and with the `debug` and `nodebug` options of the `#asm` compiler directive for blocks of assembly code.

Trace Window Fields to Trace

This section is where you specify the trace information that will be captured from the target and displayed in the Trace window. You can include the function name, file name, and line and column where each trace entry originated; the type of action being performed; the time stamp when the action was performed; and the contents of the registers after the action was performed. The more fields you select to be displayed in the Trace window, the larger each entry, and so the fewer entries the trace buffer can hold.

Saving Trace Window to a File

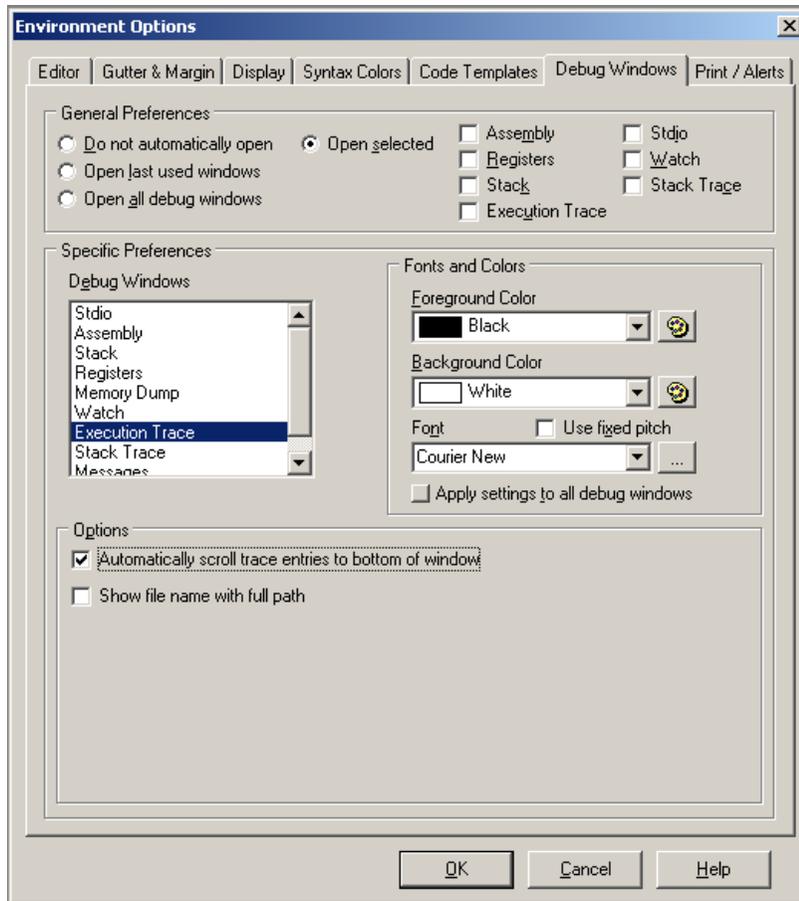
This section is where you specify the file that will contain the entries that are displayed in the Trace window. Checking the “Save on program termination” box on the Debug Windows tab will cause Dynamic C to write the contents of the trace buffer to a file when the program terminates. When this box is checked, the dropdown text box and browse button become active. You must specify a filename and its path. If a filename is entered that does not exist, the file will be created. If the file exists, it will be overwritten.

Note that this feature saves the contents of the trace buffer at the time your program terminates, so if the buffer fills up while your program is running not all trace entries received will be written to the file. If you want to save trace entries before they are lost, you can do so at any time from the Trace window.

Trace Window Configuration

The Debug Windows tab is where you select configuration options for the Trace window. Highlight “Execution Trace” in the list debug windows as shown in the screenshot below.

Figure 2. Dynamic C Menu: Options | Environment Options



The available options for the selected window are shown in the bottom half of the dialog box. The options for the Trace window are: automatic scrolling of the entries and whether the full path is displayed when the file name is displayed. These options can be set on the Debug Windows tab at compile time or toggled at runtime from a right-click pop-up menu accessible via the Trace window. The section titled [“The Trace Window”](#) has more details on the Trace window and its right-click menu.

Starting and Stopping Execution Tracing

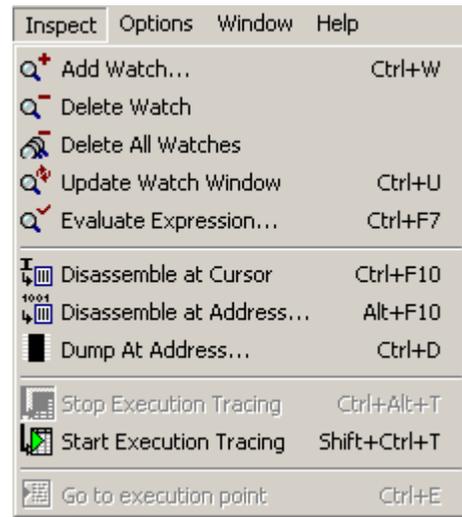
There are three ways to toggle tracing during program execution.

The first two require that tracing be enabled in the Debugger tab of the Project Options dialog and they do not trace in nodebug functions.

1. GUI options: Opening the Inspect menu, you will see the “Stop Execution Tracing” and the “Start Execution Tracing” commands, along with their keyboard shortcuts and toolbar buttons. Use any of these methods to start and stop execution tracing while the program is running or while stopped at a breakpoint.
2. `_TRACEON` and `_TRACEOFF`: Macros that are the equal to the GUI options

The third way does not require tracing to be enabled and it can be done in nodebug functions.

3. `_TRACE`: A macro that causes itself, and only itself, to be traced.



The sample program `Dem04.c` uses trace macros. A description of this sample can be found in the section titled “Using Trace Macros”.

Enabling execution tracing causes more code to be compiled into the BIOS, meaning there is less memory available on the target for your program, so if you get insufficient memory errors with your program, disabling tracing might help. Also, when you turn on tracing from the menu or a macro, your program will suffer a performance hit because of the extra communication required between Dynamic C and the target. If your program requires precise timing, tracing may interfere.

The Trace Window

Trace entries received are displayed in the Trace window. This window is only available if tracing is enabled in Project Options and Dynamic C is in run mode.

Figure 3. Execution Trace Window

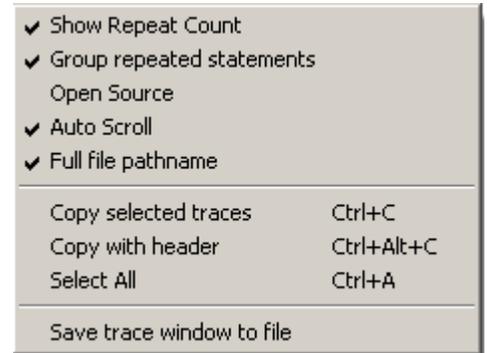
Action	Function	File Name	Line/Col	Timestamp	A : 0xB3	C : 0
Execute	main	E:\DCINPROG\SAMPLES\DEMO3.C	19,4	22141	BC : 0x0000	x : 0
Execute	main	E:\DCINPROG\SAMPLES\DEMO3.C	20,4	23391	DE : 0xC330	0 : 1
Execute	main	E:\DCINPROG\SAMPLES\DEMO3.C	20,12	24500	HL : 0x1F72	x : 0
Execute	main	E:\DCINPROG\SAMPLES\DEMO3.C	26,3	25727	AF : 0xF830	x : 1
Execute	main	E:\DCINPROG\SAMPLES\DEMO3.C	27,4	27242	BC : 0x0000	x : 1
Execute	kbhit	E:\DCINPROG\LIB\STDIO.LIB	194,2	28438	DE : 0x1000	Z : 0
					HL : 0xDFFB	S : 1
					IX : 0xC53B	
					IY : 0x1F07	
					SP : 0xDFF9	
					PC : 0x1FC6	
					XPC : 0x0030	

The Trace window has a right-click pop-up menu. An option on this menu controls the display of an additional column in the Trace window. If “Group repeated statements” is selected, then “Show Repeat Count”

may also be selected and will display values in the rightmost column of the Trace window. A value displayed under “Show Repeat Count” is the number of times the corresponding statement has been executed and, therefore, traced. The Timestamp column is not updated for subsequent traces of a repeated statement

The “Group repeated statements” option is useful when tracing statements inside a loop.

The rest of the pop-up menu options are more or less self-explanatory. You can choose to open the source code for any function in the Trace window by selecting the function and choosing “Open Source.” In Figure 3, note that a trace statement for `kbhit()` is selected in the Trace window. Choosing “Open Source” in this situation would open a window for `STDIO.LIB`, the library file that contains the function `kbhit()`.

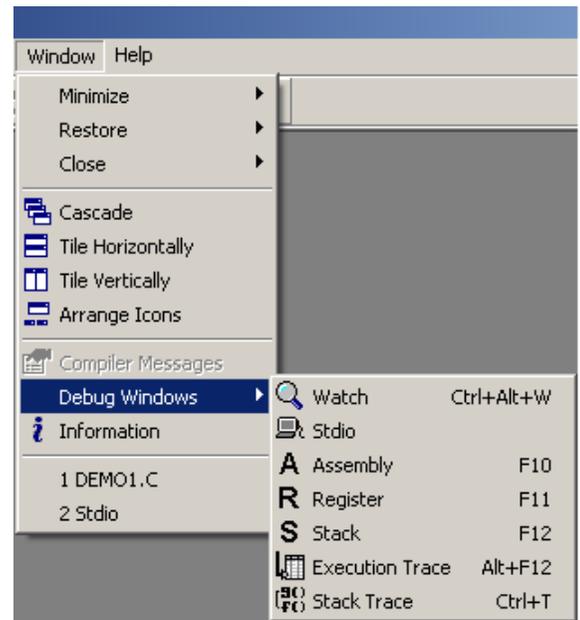


You can also toggle auto scroll, as well as decide whether to display the complete path in the File Name column. The last three menu options are for saving Trace window contents to another file. You can select trace statements in the window and then using “Copy selected traces” or “Copy with header” you can paste the selected traces anywhere you can perform a paste operation. You can also choose to copy the entire contents of the current Trace window to a named file. This is similar to the option in the Debugger tab of the Project Options dialog, which allows saving the Trace window to a file upon program termination.

Activating the Trace Window

The Trace window may be activated or deactivated using the Windows menu as shown in the screen shot below. The screen shot also shows the keyboard shortcut (Alt+F12) and the toolbar button that toggles the Trace window.

The fields displayed in the Trace window are specified in the Debugger tab of the Options | Project Options menu.



Using Trace Macros

The sample program `Demo4.c` demonstrates execution tracing using trace macros. Trace macros provide a finer grain of control than the menu options.

The sample program `Demo4.c` is in `TN253.zip`, which is available with this technical note. The sample program can also be found in the Samples folder with Dynamic C versions 9 through 10.11.

To run `Demo4.c`, open Dynamic C, go to the Debugger tab of the Project Options dialog, select “Enable execution tracing” and then choose “Full” for the Trace Level. Click “OK” to save and close the dialog, then compile and run `Demo4.c`.

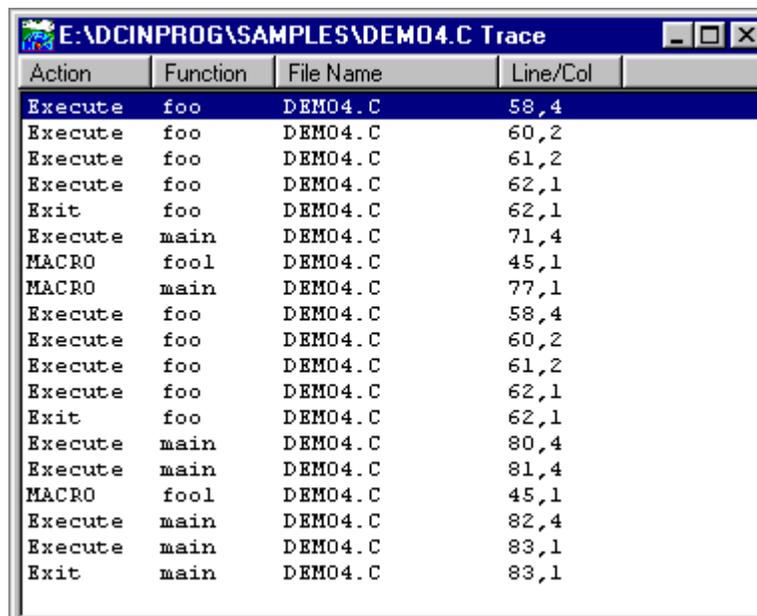


When the program finishes, the Trace window will open and you can examine its entries. If the Trace window does not open automatically when `Demo4.c` exits, open it using one of the options described in the section entitled [Activating the Trace Window](#). The Trace window can be opened anytime after the program is compiled, but execution speed is slightly affected if the window is open while the program is running.

_TRACE

The `_TRACE` macro creates one entry in the trace buffer containing the program state information at the time the macro executes. It is useful if you want to monitor one statement closely rather than follow the flow of part of a program. In `Demo4.c`, `_TRACE` is executed at lines 45 and 77, as you can see in [Figure 4](#).

Figure 4. Trace Window Contents after Running Demo4.c



Action	Function	File Name	Line/Col
Execute	foo	DEMO4.C	58,4
Execute	foo	DEMO4.C	60,2
Execute	foo	DEMO4.C	61,2
Execute	foo	DEMO4.C	62,1
Exit	foo	DEMO4.C	62,1
Execute	main	DEMO4.C	71,4
MACRO	fool	DEMO4.C	45,1
MACRO	main	DEMO4.C	77,1
Execute	foo	DEMO4.C	58,4
Execute	foo	DEMO4.C	60,2
Execute	foo	DEMO4.C	61,2
Execute	foo	DEMO4.C	62,1
Exit	foo	DEMO4.C	62,1
Execute	main	DEMO4.C	80,4
Execute	main	DEMO4.C	81,4
MACRO	fool	DEMO4.C	45,1
Execute	main	DEMO4.C	82,4
Execute	main	DEMO4.C	83,1
Exit	main	DEMO4.C	83,1

The `_TRACE` macro does not affect the `_TRACEON` and `_TRACEOFF` macros, and likewise is not affected by them. It will execute regardless of whether tracing is turned on or off. An interesting thing to note about `_TRACE` is that it generate a trace statement even when it appears in a `nodebug` function.

`_TRACEON`

The `_TRACEON` macro turns on tracing. This does not cause any information to be recorded by itself like the `_TRACE` macro, but rather causes a change of state within the debug kernel so that program state information is recorded for program and library statements executed thereafter, until the `_TRACEOFF` macro is executed or by menu command. Dynamic C captures the information you specified in the Project Options dialog and displays it in the Trace window.

In `Demo4.c`, `_TRACEON` is executed in the function `foo()`. Note that tracing is turned on in the second call to `foo1()` in `main()`, but that except for the `_TRACE` statement there are no trace statements for `foo1()`. This is because statements in nodebug functions are not traceable.

`_TRACEOFF`

The `_TRACEOFF` macro turns off tracing, starting with the next statement after it executes. Instances of the `_TRACE` macro will still execute, but tracing remains off until it is turned on by the `_TRACEON` macro or by menu command.\

Summary

Execution tracing is a good data gathering tool to use when you are not sure what is happening. The large amount of tracing information that may be saved on the host PC is available even if the program crashes.