

CodeWarrior for Nintendo DSi Technical FAQ

Rev. 1.4 7 June 2010

Table of Contents

1	FAQ	1
1.1	Setup	1
1.1.1	CodeWarrior cannot find access paths.....	1
1.1.2	The IDE or Command Line Tools have suddenly started running slowly	1
1.2	C / C++ Compiler	2
1.2.1	The pragma "options align=" does not work	2
1.2.2	If the compiler gives priority to pragmas, why is an inline function not inlined when using "#pragma inline_depth", while the preference panel is set to "Don't Inline"	2
1.2.3	My disassembled code shows register r3 being unnecessarily saved to the stack.....	3
1.2.4	Notes for using the profiler	3
1.2.5	Inline function may not be inlined	4
1.2.6	Unexpected behavior when compiler option "Replace 8-bit memory access" in the "Nintendo CodeGen" panel is enabled	4
1.2.7	The compiler option "Use Instance Manager" does not work.....	4
1.2.8	Compiler crashes when an old or illegal pre-compiled header file is used.....	4
1.2.9	Compiler for Nintendo DSi does not support the IPA Program mode	4
1.3	Assembler	5
1.3.1	What is the register which should be saved when making an assembler routine which is called in a C source code?	5
1.4	Linker.....	6
1.5	Libraries.....	6
1.5.1	Using malloc() with the Nitro SDK runtime	6
1.5.2	How to use iostream.h	6
1.5.3	Rebuilding Support Libraries.....	7
1.6	IDE	8
1.6.1	Japanese characters are garbled when I input them.....	8
1.6.2	How to open a source file with the internal editor temporarily when the "use external editor" setting is enabled?.....	9
1.6.3	How to add / remove button on the toolbar?.....	9
1.7	Debugger.....	10
1.7.1	The behavior of Debugger is not correct at an overlay area.	10
1.7.2	How can I prevent the locking of the .nef file by the IDE when debugging?	10
2	TIPS.....	11
2.1	Tips about C/C++ programming.....	11
2.1.1	How to access to a member variable of packed structure.....	11
2.2	TIPS relevant to Windows environment.....	12
2.2.1	A project cannot be opened when the folder icon has changed.....	12

Revision history

Rev.	Date	Outline
1.0	09 April 2008	New
1.1	15 April 2008	mac: adjust wording for TWL variables, adjust formatting for readability
1.2	28 September 2009	<ol style="list-style-type: none">1. Compiler crashes when an old or illegal pre-compiled header file is used2. CodeWarrior for DS does not support the feature IPA File mode.
1.3	19 March 2010	Added sections 1.2.4, 1.2.6, and 1.5.1 from <i>CodeWarrior for NINTENDO DS Technical FAQ</i> to be in parallel with Japanese version of this document.
1.4	7 June 2010	Added section 1.5.2, "How to use iostream.h" Added section 1.5.3, "Rebuilding Support Libraries"

Freescall, the Freescall logo, and CodeWarrior are trademarks of Freescall Semiconductor, Inc., Reg. U.S. Pat. & Tm. Off. All other product or service names are the property of their respective owners.

© 2008-2010 Freescall Semiconductor, Inc. All rights reserved.

1 FAQ

1.1 Setup

1.1.1 CodeWarrior cannot find access paths

Select "Preferences..." in the "Edit" menu. Then check the "Source Trees" panel.

The settings for `TWLSDK_ROOT` and `IS_TWL_DEBUGGER_DIR` are necessary for the stationery and the example projects.

Name	Type	Value	Comment
<code>TWLSDK_ROOT</code>	Environment Variable	<code>CW_TWLSDK_ROOT</code>	The full path name for TWL SDK should be set as a value of the Environment Variable <code>CW_TWLSDK_ROOT</code> .
<code>IS_TWL_DEBUGGER_DIR</code>	Environment Variable	<code>IS_TWL_DEBUGGER_DIR</code>	The full path name for the IS DEBUGGER should be set as a value of the Environment Variable <code>IS_TWL_DEBUGGER_DIR</code> .

1.1.2 The IDE or Command Line Tools have suddenly started running slowly

FlexLM is used to manage the license in CodeWarrior for Nintendo DSi. This issue may arise if another product is also using the FlexLM environment variable `LM_LICENSE_FILE`. The slowdown may be the result of a change in the search order.

Please try the following procedure if the license path (server name or file path) of another product is also set as a value of the environment variable `LM_LICENSE_FILE`.

- 1) Check whether the license server for another product of another company specified with the environment variable `LM_LICENSE_FILE` is running.
- 2) Place the CodeWarrior application higher in the search order before the path of other products.
A license path is divided by a semicolon (;).
For example:

```
LM_LICENSE_FILE = C:\Program Files\Freescale\CW for NINTENDO TWL
V1.0\license.dat;(license path of another company's product)
```

- 3) Make a backup of the contents of the variable `LM_LICENSE_FILE` by copying the text to a plain text file. Delete the environment variable `LM_LICENSE_FILE`.

The CodeWarrior IDE uses the `license.dat` file which exists in the installation folder, even when the environment variable `LM_LICENSE_FILE` is not set. If you use the command line tools, copy the `license.dat` into the same folder as the command line tools.

1.2 C / C++ Compiler

1.2.1 The pragma "options align=" does not work

`#pragma options align` is not supported by the ARM compiler, however, the compiler incorrectly does not display any warning when it is used.

The compiler has been modified such that `#pragma options align=native` and `#pragma options align=native` are still accepted for backwards compatibility, but have no effect. Other aligned options are flagged as warnings if the 'Illegal Pragmas' warning is enabled.

1.2.2 If the compiler gives priority to pragmas, why is an inline function not inlined when using "#pragma inline_depth", while the preference panel is set to "Don't Inline".

Pragmas generally receive a higher priority than preference panel compiler options, however, the pragmas and the preference panel compiler options for inlining have a slightly complicated relationship.

The compiler preference panel option "InlineDepth" encompasses the operations of both pragma "dont_inline" and pragma "inline_depth".

Value of compiler option	Equivalent pragma expression
"InlineDepth"	
Don't Inline	"dont_inline on"
Smart	"dont_inline off" AND "inline_depth(smart)"
numeric 1 to 8	"dont_inline off" AND "inline_depth(n)" (n=1 to 8)

The pragma "dont_inline" overrides the pragma "inline_depth".

The following is a ranking of these pragmas and preference panel options in terms of priority:

```

[higher priority]
|
| - pragma "dont_inline"
| - InlineDepth pref panel option set to "Don't Inline"
| - pragma "inline_depth"
| - InlineDepth pref panel option set to anything other than "Don't Inline"
|
[lower priority]

```

If you control inlining by using pragmas while the "InlineDepth" preference panel option is set to "Don't Inline", use the following order:

```

#pragma dont_inline off
#pragma inline_depth(smart)

```

1.2.3 My disassembled code shows register r3 being unnecessarily saved to the stack

The CodeWarrior for Nintendo DSi 1.0 compiler uses the register r3 as a dummy to which it can push even numbered registers in order to keep the alignment of stack area at 8-bytes boundary when the number of registers which must be saved to the stack is an odd number.

At this time, register r3 is used only as a dummy.

In the older compiler that shipped with CodeWarrior for Nintendo DS 1.2, you could use "adding / subtracting to SP", but there were problems with this method.

1.2.4 Notes for using the profiler

- The profiler keeps the working area on the memory area of the target (IS-NITRO-EMULATOR or Ensata). If too large a number is set as the value for the 3rd argument (maximum number of total of the functions which profiler analyzes) or the 4th argument (maximum depth of function calling which profiler analyzes) of the `ProfilerInit()` function, the target's memory is overflowed.
- The profiler does not support a thread and an interrupt. In the range which is analyzed by a profiler, if calling a thread and generating an interrupt occur frequently, the profiler may make a mistake in analyzing OR the data in the profiler's working area may be broken.

1.2.5 Inline function may not be inlined

The following functions are never inlined:

- Functions that return class objects that need destruction.
- Functions with class arguments that need destruction.
- Functions with variable argument lists.

1.2.6 Unexpected behavior when compiler option "Replace 8-bit memory access" in the "Nintendo CodeGen" panel is enabled

CodeWarrior for Nintendo DS 3.0 does not support the compiler option "Replace 8-bit memory access." Do not enable this compiler option. This option is in the Nintendo CodeGen panel, under the Code Generation category of the Target Settings panel.

This compiler option was supported in CodeWarrior for Nintendo DS 1.2 and earlier. The compiler option was used to cope with the restriction of the TEG board.

1.2.7 The compiler option "Use Instance Manager" does not work

The compiler option "Use Instance Manager" is not supported in CodeWarrior for Nintendo DSi.

1.2.8 Compiler crashes when an old or illegal pre-compiled header file is used

When you use a new version compiler, you must delete the old pre-compiled header files first.

In TWL SDK, the pre-compiled header files are located in the following directory:

```
{${TwlSDK}}\cache\include
```

1.2.9 Compiler for Nintendo DSi does not support the IPA Program mode

WARNING! Per-program IPA mode has not been fully-tested for ARM development. Use at your own risk.

1.3 Assembler

1.3.1 What is the register which should be saved when making an assembler routine which is called in a C source code?

The CodeWarrior for Nintendo DSi compiler is based on ATPCS (Arm Thumb Procedure Call Standards) which is by ARM Ltd.

- Use registers r0-r3 to pass parameter values into routines, and to pass result values out. Between subroutine calls you can use r0-r3 for any purpose. A called routine does not have to restore r0-r3 before returning.
- Use registers r4-r11 to hold the values of a routine's local variables. In Thumb state, in most instructions you can only use registers r4-r7 for local variables. A called routine must restore the values of these registers before returning, if it has used them.
- Register r12 is the intra-call scratch register, `ip`. Between procedure calls you can use it for any purpose. A called routine does not need to restore r12 before returning.
- Register r13 is the stack pointer, `sp`. You must not use it for any other purpose. The value held in `sp` on exit from a called routine must be the same as it was on entry.
- Register r14 is the link register, `lr`. If you save the return address, you can use r14 for other purposes between calls.
- Register r15 is the program counter, `pc`. It cannot be used for any other purpose.

Register	Synonym	Special	Role in the procedure call standard
r15	-	pc	Program counter.
r14	-	lr	Link register.
r13	-	sp	Stack pointer.
r12	-	ip	Intra-procedure-call scratch register.
r11	v8	-	ARM-state variable register 8.
r10	v7	s1	ARM-state variable register 7. Stack limit pointer in stack-checked variants.
r9	v6	sb	ARM-state variable register 6. Static base in RWPI variants.
r8	v5	-	ARM-state variable register 5.
r7	v4	-	Variable register 4.
r6	v3	-	Variable register 3.
r5	v2	-	Variable register 2.
r4	v1	-	Variable register 1.
r3	a4	-	Argument/result/scratch register 4.
r2	a3	-	Argument/result/scratch register 3.

r1	a2	-	Argument/result/scratch register 2.
r0	a1	-	Argument/result/scratch register 1.

Note:

Keep the alignment of stack area at 8-bytes boundary when you store register to stack area. We recommend the method that you store even-number registers at once by using r3 as a dummy when the number of registers which must be saved to the stack is odd-number.

1.4 Linker

(There is no FAQ information about the linker at this time.)

1.5 Libraries

1.5.1 Using malloc() with the Nitro SDK runtime

If you use the Nitro SDK runtime library, crt0.o, and want to use the malloc() function from the MSL C library, you should first call the __init_hardware() function, defined in the file:

```
{CodeWarrior install directory}\ARM_EABI_Support\Runtime\Runtime_ARM\
(Common_Source)\__NITRO_eabi_init.c
```

from the runtime library.

It performs the necessary hardware initialization required for malloc() to work.

1.5.2 How to use iostream.h

The iostream header is dependent on the use of malloc(). To use malloc(), you must call the runtime function __init_hardware() in

```
ARM_EABI_Support\Runtime\Runtime_ARM\ (Common_Source)\__NITRO_eabi_init.c.
```

You will likely want to customize __init_hardware() (e.g. resize heap memory). This code can be written into TwlStartUp() directly.

For example:

```

#include <twl.h>
#include <iostream.h>

extern "C" void __init_hardware(); // setup heap memory
extern "C" void TwlStartup();      // this func is called before C++ init:
                                   // (__call_static_initializers())

void TwlStartUp()
{
    __init_hardware();
}

void TwlMain ()
{
    OS_Init();
    OS_Printf("Welcome to CodeWarrior for NINTENDO TWL\n");
    OS_Terminate();
}

```

1.5.3 Rebuilding Support Libraries

The DSi installation comes with pre-built standard C/C++ and runtime libraries. It is not normally necessary for you to rebuild the libraries that were delivered with the product. It is not recommended that you rebuild the support libraries. However, if needed, you can rebuild the support libraries by using supplied makefiles. Only the cygwin `bash` command shell is supported for the command-line environment. To rebuild the support libraries, the environment variable, `CWINSTALL`, must be set by the user. In `bash`, this is usually done as in the following example:

```
> export CWINSTALL="C:/Program Files/Freescale/CW for NINTENDO DSi V1.3"
```

Quotes are required for paths containing spaces. `CWINSTALL` should be set to the root of the DSi installation.

There are five unique makefiles used to build the necessary libraries. For example, to rebuild the standard C libraries do the following:

```

> cd $(CWINSTALL)/ARM_EABI_Support/msl/MSL_C/MSL_ARM/Project
> make -f MSL_C.NITRO.mak clean
> make -f MSL_C.NITRO.mak

```

The remaining support libraries can be rebuilt similarly.

The C++ libraries can be rebuilt by changing directory to:

```
$(CWINSTALL)/ARM_EABI_Support/msl/MSL_C++/MSL_ARM/Project
```

and using the `MSL_C++.NITRO.mak` file. Precompiled headers are not used normally. If necessary, however, you can rebuild them with: `MSL_C++_Header.NITRO.mak`.

```
> cd $(CWINSTALL)/ARM_EABI_Support/msl/MSL_C++/MSL_ARM/Project
```

```
> make -f MSL_C++.NITRO.mak clean
```

```
> make -f MSL_C++.NITRO.mak
```

Additional MSL support libraries can be rebuilt by changing directory to:

```
$(CWINSTALL)/ARM_EABI_Support/msl/MSL_Extras/MSL_ARM/Project
```

and using the `MSL_Extras.NITRO.mak` file.

```
> cd $(CWINSTALL)/ARM_EABI_Support/msl/MSL_Extras/MSL_ARM/Project
```

```
> make -f MSL_Extras.NITRO.mak clean
```

```
> make -f MSL_Extras.NITRO.mak
```

Finally, the runtime support libraries can be rebuilt by changing directory to:

```
$(CWINSTALL)/ARM_EABI_Support/Runtime/Runtime_ARM/Runtime_NITRO/(Projects)
```

and using the `NITRO_Runtime.mak` file.

```
> cd $(CWINSTALL)/ARM_EABI_Support/Runtime/Runtime_ARM/Runtime_NITRO/(Projects)
```

```
> make -f NITRO_Runtime.mak clean
```

```
> make -f NITRO_Runtime.mak
```

1.6 IDE

1.6.1 Japanese characters are garbled when I input them

Select "Preferences..." in the "Edit" menu of the CodeWarrior IDE, then confirm the "Font & Tabs" panel. It is necessary to select a font (for example, in Japanese environment, "System" etc.) which can display Japanese.

For details, please refer to "IDE User Guide".

1.6.2 How to open a source file with the internal editor temporarily when the "use external editor" setting is enabled?

There are the following ways to open a source file with the internal editor temporarily to set a break point (etc...) when the "use external editor" setting is enabled.

- Use "Toggle external editor mode"

This is the icon at the rightmost (in default condition) of the tool bar of IDE. When this icon is pushed condition, a file is opened by the external editor. When this icon is not pushed condition, a file is opened by the internal editor. The default Key Binding for this function is "Ctrl + J".

- Use "Alt" key

If you double-click with "Alt" key or press "Alt + Enter" on a file name on the project window, the file is opened with the internal editor.

Moreover, there is the following operation to open the place of a definition or a declaration with the internal editor from the opened source file in the internal editor.

- 1) Right-click on the variable / type declaration / function / macro / class.
- 2) Left-click with "Ctrl" key or press "Ctrl + Enter" on the "Go to xxxx definition of xxxx" or "Go to xxxx declaration of xxxx" on the pop-up menu.

1.6.3 How to add / remove button on the toolbar?

The following procedure can be used to add a button to the toolbar.

- 1) Select **Commands and Key Bindings...** in the "Edit" menu at CodeWarrior IDE to open the **Customize IDE Commands** window.
- 2) Choose a function which you want to add to the toolbar, in the "Commands" pane or the "Toolbar Items" pane. Then, drag and drop the icon which is on the left of the function name to the arbitrary positions on the toolbar.
(A function without an icon cannot be added to the toolbar.)

The following procedure can be used to remove a button from the toolbar.

- 1) Right-click on the position of a button which you want to remove from the toolbar.
- 2) Select **Remove Toolbar** on the pop-up menu.

1.7 Debugger

1.7.1 The behavior of Debugger is not correct at an overlay area.

You must use the API in the TWL SDK for management of overlays for the CodeWarrior Debugger to work correctly. This is because the CodeWarrior Debugger sets internal breakpoints on the overlay management functions of the "ROM File System (FS) API" in TWL SDK.

The debugger watches the loading and unloading overlay modules. As a result the debugger monitors the condition of overlay modules for displaying the correct source file etc.

If you do not use the API, the following warning messages will be displayed when the debugger starts.

- [Unable to find needed symbol "FS_StartOverlay" in ARM9 Elf file.
Overlay debugging will not work correctly.]
- [Unable to find needed symbol "FS_UnloadOverlayImage" in ARM9 Elf file.
Overlay debugging will not work correctly.]

1.7.2 How can I prevent the locking of the .nef file by the IDE when debugging?

In the default state, when using the .nef file the IDE locks it. Therefore you cannot build from the command-line while the IDE is using the .nef.

To avoid this uncheck the check-box `Cache symbolics between runs` on the "Debugger Settings" panel in the target settings window.

In addition, you cannot immediately build after changing this setting, since the .nef file was locked already. It will be unlocked after you start your next debugging session.

2 TIPS

2.1 Tips about C/C++ programming

2.1.1 How to access to a member variable of packed structure

Usually, an integer variable should be aligned at 4-bytes boundary because 4-bytes access instruction is used to access to integer variable. If the alignment of the integer variable separates from 4-bytes boundary as a result of packing the structure (because, there is char variable or short variable before the integer variable, etc...), it is necessary to shift the top address of the packed structure to adjust the alignment of the integer variable. For example:

```
typedef struct {
    u8  dstMac[6]  __attribute__((aligned(1)));
    u8  srcMac[6]  __attribute__((aligned(1)));
    u16 type       __attribute__((aligned(1)));
    u32 n2         __attribute__((aligned(1)));
} TEST_PACKET;

{
    TEST_PACKET packet_a;
    TEST_PACKET packet_b;
```

It is necessary to shift the top address of the structure to the 12-bytes boundary for accessing to the variable "n2" correctly. However, the value which can be used for `__attribute__((aligned(x)))` is only the number of factorials of 2 (1, 2, 4, 8, 16, 32, 64, etc ...).

In such a case, there is the following method.

```
char packet_data_a[sizeof(u16) + sizeof(TEST_PACKET)]
__attribute__((aligned(4)));
char packet_data_b[sizeof(u16) + sizeof(TEST_PACKET)]
__attribute__((aligned(4)));
TEST_PACKET* packet_a = (TEST_PACKET*)&packet_data_a[sizeof(u16)];
TEST_PACKET* packet_b = (TEST_PACKET*)&packet_data_b[sizeof(u16)];
```

2.2 TIPS relevant to Windows environment

2.2.1 A project cannot be opened when the folder icon has changed.

In Windows XP (or later), you can change the icon of a folder. However, CodeWarrior cannot open the project if the icon of the folder which contains the project file has changed, because CodeWarrior interprets the attribute of this folder as read only. This is because the "Read-only" attribute flag of folder and the "System" attribute flag of folder are used as "customized" flag of folder in Windows XP.

Note: For more information, see <http://support.microsoft.com/kb/326549/en-us>

Do not do customizing (changing the icon of the folder, etc...) for folders which contain CodeWarrior project files. Execute the following command at the command prompt. to recover the attribute of the folder:

```
attrib -R -S [folder name]
```