Tornado® 2.2

MIGRATION GUIDE

WindRiver®

Corporate Headquarters
Wind River Systems, Inc.
500 Wind River Way
Alameda, CA 94501-1153
U.S.A.

toll free (U.S.): 800/545-WIND
telephone: 510/748-4100
facsimile: 510/749-2010

For additional contact information, please visit the Wind River URL:

http://www.windriver.com

For information on how to contact Customer Support, please visit the following URL:

http://www.windriver.com/support

Tornado Migration Guide, 2.2

15 Aug 02
Part #: DOC-14256-ZD-02

Contents

1 Introduction

Migrating to Tornado 2.2 and VxWorks 5.5

This guide contains information designed to assist VxWorks developers in moving to Tornado 2.2 and VxWorks 5.5.

This document is particularly aimed at developers moving from Tornado version 2.0.x and VxWorks version 5.4.x, but it also contains information useful for migrating from other versions of Tornado and VxWorks.

Migration Scenarios

While this guide is focused on providing a smooth migration for Tornado 2.0.x customers to Tornado 2.2, the information presented here is also useful if you are migrating from a different Tornado version. In brief, there are several different migration scenarios:

- **New project by Tornado 2.0.x developer(s).** This is the focus of this guide. The information collected here is meant to simplify moving an existing code base to work with Tornado 2.2 and VxWorks 5.5.
- **New project by Tornado 2.1 developer(s).** If your code base is written for Tornado 2.1, it requires less effort to move to Tornado 2.2/VxWorks 5.5 than in the Tornado 2.0.x migration illustrated in this guide. In particular, the GNU compiler has changed very little between 2.1 to 2.2. However, other topics, such as project migration, are relevant to Tornado 2.1 developers.
- **New project start by pre-Tornado 2.0 developer(s).** It is recommended that customers migrating from a version prior to Tornado 2.0 and VxWorks 5.4

make extensive use of the Tornado 2.2 documentation, especially the *Getting Started Guide*, the *Tornado User's Guide: Projects*, and the *Tornado 2.2 Release Notes*. Developers using networking facilities should also consult the *VxWorks Network Programmer's Guide*.

- **In-progress project by pre-Tornado 2.2 developer(s).** This migration guide contains useful information for this type of project, especially with regard to project migration and compiler changes. In addition, the *Tornado Release Notes, 2.2,* includes a list of new features that may be useful in project design.

Scope of This Document

This guide is primarily concerned with the differences between earlier versions of Tornado/VxWorks and this current release. It does not attempt to cover any new features introduced in this release, but instead focuses on changed or removed features that have the highest potential impact on customer code bases. For a detailed description of new features, see the *Tornado Release Notes, 2.2*.

This guide does not cover most optional and third-party products, but concentrates on features that are included with the standard Tornado 2.2/VxWorks 5.5 release. There is one exception to this scope, however: information is included on changes in the optional Diab compiler.

2
Binary Compatibility

2.1 Introduction

Binary compatibility with previous versions of VxWorks is not guaranteed by this release. Wind River strongly recommends that all source code be recompiled, and that any third-party code supplied in object form be upgraded. This chapter describes the enhancements made to the current version of VxWorks that can cause binary incompatibility and changes to the VxWorks binary archives.

2.2 Object Module Format

For most architectures, the object module format (OMF) that Tornado/VxWorks now supports is ELF. The debugging information format is DWARF 2. This change impacts some architectures more than others. Table 2-1 lists the formats that were used in older versions of Tornado 2, and the corresponding information for the new release.

For architectures in which the binary file format has changed since the last version, binary files in the old format must be recompiled.

Table 2-1 **Object Module Format and Debugging Information Format by Architecture and Tornado Version**

Architecture	**Tornado 2 / 2.0.2 / 2.1**		**Tornado 2.2**	
	OMF	**Debug Info Format**	**OMF**	**Debug Info Format**
68K/CPU32	**a.out**	STABS	**a.out**	STABS
ARM/StrongARM/XScale	COFF	STABS	ELF	DWARF 2
ColdFire	ELF	DWARF 2	ELF	DWARF 2
Hitachi SuperH	ELF	DWARF 2	ELF	DWARF 2
MIPS	ELF	STABS	ELF	DWARF 2
Pentium	**a.out**	STABS	ELF	DWARF 2
PowerPC	ELF	STABS	ELF	DWARF 2
Simulator (NT)	PE-COFF	STABS	PE-COFF	STABS
Simulator (Solaris)	ELF	STABS	ELF	STABS

Some systems may have old boot ROMs installed, which cannot be upgraded to a VxWorks 5.5 boot ROM. For these, conversion of the fully linked ELF bootable images into the old format is possible using the **objcopyarch** utility. Because this solution has some limitations (listed below), it should only be considered if the boot ROM cannot be upgraded. The known limitations are as follows:

- Relocatable and **.o** files cannot be converted. Only fully linked images can be converted.
- Because no conversion of the debugging information is possible, this information is lost in the conversion.
- Because the target server cannot read the old OMF, it must be explicitly given the ELF version of the file as its core file.

2.3 Changes in Object Archive Layout

The arrangement of object file archives has been reorganized in VxWorks 5.5. This change was made to support both GNU- and Diab-compiled archives, as well as to simplify the addition of CPU variants within a processor family (such as PowerPC or MIPS). The following sections describe the pre-VxWorks 5.5 archive layout, the new hierarchical layout, and the link path that determines precedence in link operations.

NOTE: Direct access to the VxWorks archives is not required for most users. In fact, it is recommended that the VxWorks archives be backed up before any direct operations are performed on them.

2.3.1 Archive Layout Prior to VxWorks 5.5

In VxWorks versions prior to 5.5, the object archive layout was relatively flat, with archives and objects found in **../target/lib/**.[1] With the CPU type defined by the macro **CPU**, and the toolchain defined by the macro **TOOL**, this directory was organized shown in Table 2-2:

Table 2-2 **Pre-VxWorks 5.5 Object Archive Directories**

Directory	Notes
../target/lib/lib(CPU)(TOOL)vx.a	VxWorks archives
../target/lib/lib(CPU)(TOOL)gcc.a	For **TOOL=gnu**
../target/lib/lib(CPU)(TOOL)dcc.a	For **TOOL=diab**
../target/lib/obj(CPU)(TOOL)vx/	VxWorks objects directory
../target/lib/obj(CPU)(TOOL)test/	Test routine objects directory

As an example, for **CPU=PPC604** and **TOOL=gnu**, **../target/lib** would have the following contents:

../target/lib/libPPC604gnuvx.a
../target/lib/libPPC604gnugcc.a
../target/lib/objPPC604gnuvx/
../target/lib/objPPC604gnutest/

With this earlier method, the BSP makefile included the file **../target/h/make/defs.bsp**, which defined the macro **LIBS** to point at the appropriate archives in the **../target/lib** directory.

1. All paths are expressed relative to the base of the installed Tornado tree.

2.3.2 VxWorks 5.5 Object Archive Layout

In VxWorks 5.5, the archives have been arranged hierarchically, with a tree beneath **../target/lib** characterized by CPU architecture family, CPU variant, and tool (GNU or Diab). The old archive layout is also preserved for backward compatibility.

Three macros define the location of all archives and object directories:

- **VX_CPU_FAMILY .** The CPU architecture family, such as **pentium** or **arm**.
- **CPU .** The specific CPU variant within this family, such as **PENTIUM4**.
- **TOOL .** For example, **gnu** or **diab**.

Table 2-3 describes the new archive directory tree (not including the pre-VxWorks 5.5 files and directories described in the previous section).

Table 2-3 **VxWorks 5.5 Object Archive Directory Structure**

Directory	Description
../target/lib/*VX_CPU_FAMILY***/**	CPU family (for example, **ppc**, **mips**, and **arm**). Currently, this directory only contains subdirectories. For example, **../target/lib/mips**.
../target/lib/*VX_CPU_FAMILY***/***CPU***/**	CPU variant files. Typically, this directory only contains per-toolchain and common subdirectories. For example, **../target/lib/ppc/PPC440**.
../target/lib/*VX_CPU_FAMILY***/***CPU***/***commonConfig***/**[*]	Contains most VxWorks libraries, in archives called **lib***LibBaseName***.a**; and objects, in subdirectories **obj***LibBaseName***/**. For example, **libos.a** contains OS objects compiled with the default toolchain for your CPU family.
../target/lib/*VX_CPU_FAMILY***/***CPU***/***gnuConfig***/**[*]	Contains VxWorks libraries and objects that make use of C++, compiled with the GNU toolchain.
../target/lib/*VX_CPU_FAMILY***/***CPU***/***diabConfig***/**[*]	Contains VxWorks libraries and objects that make use of C++, compiled with the Diab toolchain.

* The *commonConfig*, *gnuConfig*, and *diabConfig* values depend on the compiler configuration you are using. For example, **commonle** indicates a compiler configured for little-endian compilation.

The last two directories in Table 2-3 require further elaboration. Diab C++ support and GNU C++ support are not 100% interoperable; therefore, when that support is available for an architecture, VxWorks facilities that require C++ include separate archives and objects for both toolchains. The remaining objects, found under the *commonConfig* directory, are built with the default toolchain for your architecture. In cases where Diab and GNU are completely interoperable, applications can be linked with the common archives, regardless of your choice of compiler, because they are written in portable C and assembler. Table 2-4 lists the current availability of the GNU and Diab toolchains for each architecture, as well as the default compiler used for each architecture.

Table 2-4 **Toolchain Support Per CPU Family**

CPU Family	GNU Support?	Diab Support?	Default Toolchain
PowerPC	yes	yes	GNU
Pentium	yes	no	GNU
ARM/StrongARM/XScale	yes	yes	Diab
MIPS	yes	yes	Diab
SH	yes	yes	GNU
MC68k/CPU32	yes	no	GNU
Coldfire	no	yes	Diab
Win32 Simulator	yes	no	GNU
Solaris Simulator	yes	no	GNU

2.3.3 Link Precedence

Because of the complexity of the new archive layout and the possibility of libraries and objects inhabiting different levels in the directory hierarchy, a link macro **LD_LINK_PATH** has been defined. **LD_LINK_PATH** sets a precedence for linking libraries. As the linker executes during a VxWorks build, it looks for the appropriate objects starting with the first directory in the link path and will only link the first correct object it finds. **LD_LINK_PATH** is defined in **../target/h/make/defs.link**.

WARNING: Do not modify **LD_LINK_PATH**.

3
VxWorks API Changes

3.1 Introduction

VxWorks 5.5, wherever possible, avoids changes to public interfaces. Application code written for VxWorks 5.4 produces the same behavior with VxWorks 5.5 and compiles after only minor changes. This chapter provides details on these changes.

Types of API Changes

The API changes in VxWorks 5.5 are broken down into three categories, each covered a separate section of this chapter:

- *3.2 Modified Routines*, p.10, describes routines that have been modified in VxWorks 5.5.
- *3.3 Retired Libraries*, p.16, lists libraries that have been removed or deprecated in VxWorks 5.5.
- *3.4 Other API Changes*, p.18, notes several facilities that have changed since Tornado 2.0/VxWorks 5.4 and points to documentation on them.

This chapter does not describe any *new* routines or libraries in VxWorks 5.5; they do not create migration issues. These additions are documented in the *Tornado Release Notes, 2.2*.

3.2 Modified Routines

The modified routines are divided into the following categories, based on the severity of the change:

- ***Must change.*** Any use of these routines requires modification of your code.
- ***Recommended change.*** While these modified routines are backward-compatible, it is recommended that you change any code that uses them.
- ***Minor change.*** The changes to these routines are relatively small or cosmetic and, at worst, produce a compiler warning for pre-VxWorks 5.5 code.

For a full description of all of these routines, see the *VxWorks API Reference*.

3.2.1 Must-Change Routines

A small set of routines changed significantly in VxWorks 5.5. Unmodified use of these routines either produces different functionality, creates a compile-time error, or causes a runtime error. As a result, you *must* change application code that uses these routines. Complete information about the proper use of these functions in VxWorks 5.5 can be found in the library and routine entries in the *VxWorks API Reference*.

One library in particular has changed significantly: **telnetLib** has been replaced by **telnetdLib**, and the routines in this library and their usage have changed. If you are using this library, see the *VxWorks OS Libraries API Reference* for a complete description of the routines and their use. In addition, there is a code example for this library in **.../target/unsupported/telnet/echoShell.c**, which can be used as a template.

Table 3-1 lists these routines and their corresponding libraries.

Table 3-1 **Must-Change Routines**

Routine	Library
arpAdd( )	**arpLib**
bootpParamsGet( )	**bootpLib**
cacheR4kLibInit( )	**cacheR4kLib**
dhcpcBootInit( )	**dhcpBootLib**

Table 3-1 **Must-Change Routines**

Routine	Library
dhcpcLibInit( )	**dhcpcLib**
dhcpcOptionSet( )	**dhcpcCommonLib**
dhcpsLibInit( )	**dhcpsLib**
fppRestore( )[*]	**fppArchLib**
fppSave( )[†]	**fppArchLib**
loadModuleAt( )	**loadLib**
ripLibInit( )	**ripLib**
selectInit( )	**selectLib**

* x86 CPUs only.
† x86 CPUs only.

The changes to each routine are summarized below. For more definitive details on these routines, see the routine and library entries in the *VxWorks API Reference*.

- **arpAdd( ).** The **ATF_USETRAILERS** flag is no longer supported.
- **bootpParamsGet( ).** New parameters have been added to this function. The **bootpParams** structure has been changed.
- **cacheR4kLibInit( ).** New parameters have been added to this function.
- **dhcpcBootInit( ).** New parameters have been added to this function.
- **dhcpcLibInit( ).** The additional parameter *maxSize* is now passed to this function.
- **dhcpcOptionSet( ).** The parameter list has been reduced.
- **dhcpsLibInit( ).** The parameter list has been consolidated to single parameter.
- **fppRestore( ) and fppSave( ) [x86 only].** A new **FP_CONTEXT** definition has been added to support streaming SIMD technology on the Pentium 2, 3, and 4.

  There are two kinds of floating-point contexts, each with its set of routines:

  – 108-byte context, for older FPUs (i80387, i80487, and Pentium) and older MMX technology. This context uses **fppSave( )**, **fppRestore( )**, **fppRegsToCtx( )**, and **fppCtxToRegs( )** to save and restore the context, and to convert to or from the **FPPREG_SET**.

  - 512-byte context, for newer FPUs, newer MMX technology and streaming SIMD technology (PentiumII, III, 4). This context uses **fppXsave( )**, **fppXrestore( )**, **fppXregsToCtx( )**, and **fppXctxToRegs( )** to save and restore the context, and to convert to or from the **FPPREG_SET**. For more details on VxWorks support for Pentium 2, 3, and 4 processors, see the *VxWorks for Pentium Architecture Supplement*.

- **loadModuleAt( ).** This function now correctly returns NULL on failure.
- **ripLibInit( ).** A new *authType* parameter has been added to this function.
- **selectInit( ).** A new *numFiles* parameter has been added to this function.

3.2.2 Recommended-Change Routines

Some routines have changed in VxWorks 5.5, yet retain backward compatibility with earlier versions of VxWorks. While it is not crucial that code using these routines be modified in order to run correctly, it is recommended. In many cases, doing so will ease future migration. For example, the routine **muxPollReceive( )** is supported in VxWorks 5.5, but is deprecated and will be removed in a future release of VxWorks.

Table 3-2 lists these routines and their corresponding libraries.

Table 3-2 **Recommended-Change Routines**

Routine	Library
cd( )	**usrFsLib**
diskFormat( )	**usrFsLib**
diskInit( )	**usrFsLib**
fei82557EndLoad( )	**fei82557End**
motFecEndLoad( )	**motFecEnd**
muxPollReceive( )	**muxLib**
netDrv( )	**netDrv**
netLibInit( )	**netLib( )**
proxyNetCreate( )	**proxyArpLib**
rm( )	**usrFsLib**

Table 3-2 **Recommended-Change Routines**

Routine	Library
rmdir( )	**usrFsLib**
symFindByValue( ))	**symLib**
symFindByValueAndType( )	**symLib**

The changes to each routine are summarized below. For more definitive details on these routines, see the routine and library entries in the *VxWorks API Reference*.

- **cd( ).** The argument type has changed from **char** * to **const char** *, and the routine has been moved from **usrLib** to **usrFsLib**. The header file **usrLib.h** may still be included for backward compatibility; changing this file to **usrFsLib.h** is optional for now.
- **diskFormat( ).** The argument type has changed from **char** * to **const char** *, and the routine has been moved from **usrLib** to **usrFsLib**. The header file **usrLib.h** may still be included for backward compatibility; changing this file to **usrFsLib.h** is optional for now.
- **diskInit( ).** **The** argument type has changed from **char** * to **const char** *, and the routine has been moved from **usrLib** to **usrFsLib**. The header file **usrLib.h** may still be included for backward compatibility; changing this file to **usrFsLib.h** is optional for now. This routine is now obsolete; it is recommended that you use **dosFsVolFormat( )**.
- **fei82557EndLoad( ).** The *initString* now includes a field for *deviceId*. Although it is not required, it is recommended that you pass 0 as *deviceId* if you are not using this field.
- **motFecEndLoad( ).** The *initString* now includes a field for *clockSpeed*. Although it is not required, it is recommended that you pass 0 as *clockSpeed* if you are not using this field.
- **muxPollReceive( ).** This routine has been deprecated. Use **muxTkPollReceive( )** instead.
- **netDrv( ).** The **include** macro for this driver has changed from **INCLUDE_NETWORK** to **INCLUDE_NET_DRV**.
- **netLibInit( ).** The **include** macro for this library has changed from **INCLUDE_NETWORK** to **INCLUDE_NET_LIB**.

- **proxyNetCreate( ).** This routine no longer returns the **errno** value **S_proxyArpLib_INVALID_INTERFACE**. Remove any code that checks for this value.
- **rm( ) and rmdir( ).** The argument type has changed from **char** * to **const char** *, and the routine has been moved from **usrLib** to **usrFsLib**. The header file **usrLib.h** may still be included for backward compatibility; changing this file to **usrFsLib.h** is optional for now.
- **symFindByValue( ).** This routine has been obsoleted. Use **symByValueFind( )** instead.
- **symFindByValueAndType( ).** This routine has been obsoleted. Use **symByValueAndTypeFind( )** instead.

3.2.3 Minor-Change Routines

The routines listed in Table 3-3 retain full backward compatibility with Tornado 2.0/VxWorks 5.4. The changes made to the functions in this category are minor and do not cause problems in migrating code. Some of these changes are enhancements for additional functionality (say, additional error reporting) that do not change the routine's previous behavior. Any code modifications you make to accommodate these changed routines are optional.

Table 3-3 **Minor-Change Routines**

Routine	Library
arpDelete( )	**arpLib**
copy( )	**usrFsLib**
copyStreams( )	**usrFsLib**
ll( )	**usrFsLib**
ls( )	**usrFsLib**
mkdir( )	**usrFsLib**
msgQCreate( )	**msgQLib**
msgQDelete( )	**msgQLib**
msgQSend( )	**msgQLib**
pwd( )	**usrFsLib**

Table 3-3 **Minor-Change Routines**

Routine	Library
select( )	**selectLib**
semBCreate( )	**semBLib**
semCCreate( )	**semCLib**
semGive( )	**semLib**
semMCreate( )	**semMLib**
setsockopt( )	**sockLib**

The changes to each routine are summarized below (summarized by library). For more definitive details on these routines, see the routine and library entries in the *VxWorks API Reference*.

- **arpDelete( ).** The additional **errno** macro **S_arpLib_INVALID_HOST** is now supported.
- **copy( ), copyStreams( ), ll( ), ls( ), mkdir( ), pwd( ).** These routines have been moved from **usrLib** to **usrFsLib**; this change is backward-compatible. The new **include** macro **INCLUDE_DISK_UTIL** and the new component *File System and Disk Utilities* have been added.
- **msgQCreate( ), msgQDelete( ), msgQSend( ).** New behavior has been added to support the event facility. (See the *VxWorks API Reference* library entries for **eventLib** and **msgQEvLib**.)
- **select( ).** The maximum value for **FD_SET** has been raised from 256 to 2048 bits.
- **semBCreate( ), semCCreate( ), semGive( ), semMCreate( ).** New behavior has been added to support the event facility. (See the *VxWorks API Reference* library entries for **eventLib** and **semEvLib**.)
- **setsockopt( ).** Several new socket options are now supported.

3.3 Retired Libraries

Table 3-4 lists the libraries that have been removed in VxWorks 5.5.

Table 3-4 **Retired Libraries**

Libraries	Description
cacheI960CxALib **cacheI960CxLib** **cacheI960JxALib** **cacheI960JxLib**	i960 architecture-specific cache libraries
cacheR3kALib	MIPS R3000 cache library
etherLib	Ethernet hook library
http*moduleName* (57 total libraries) For example: **httpFormutilTextfield**	Wind Web Server libraries
ideDrv	IDE/ATA driver
if_ulip	BSD-style driver for ULIP for simulators
ntEnd	END driver for Windows ULIP
ospfLib	OSPF library
saIoLib **snmpAuxLib** **snmpBindLib** **snmpEbufLib** **snmpIoLib** **snmpProcLib** **snmpdLib** **subagentLib**	Envoy SNMP Agent libraries
telnetLib	Telnet server library
unixSio **winSio**	Simulator serial driver libraries

The changes to each library group are summarized below. For more definitive details on these libraries, see the corresponding entries in the *VxWorks API Reference*.

- **cacheI960CxALib, cacheI960CxLib, cacheI960JxALib, cacheI960JxLib.** The i960 architecture is not supported in Tornado 2.2/VxWorks 5.5.
- **cacheR3kALib.** This library is no longer part of the VxWorks public API. The public interface for MIPS R3000 caching can be found in **cacheR3kLib**.
- **etherLib.** This library, which provided access to raw Ethernet frames, has been obsoleted. Several good alternatives exist:
  - **bpfLib**, the Berkeley Packet Filter library.
  - Use of a snarfing protocol to intercept all Ethernet frames.
  - Use of a user-written protocol to more selectively intercept frames.

  For details, see the *VxWorks Network Programmer's Guide*.
- **http*moduleName*.** The 57 libraries in this group made up an obsoleted version of the Wind Web Server. A up-to-date version is available as an upgrade for prior users of Wind Web Server. Please contact your Wind River account representative for details on the availability of this upgrade.
- **ideDrv.** This library has been obsoleted by **ataDrv**. See the corresponding entry in the *VxWorks API Reference* for details.
- **if_ulip, ntEnd, unixSio, winSio.** These driver libraries are used by the Windows and Solaris simulators, but are no longer part of the public VxWorks API.
- **ospfLib.** This library is an older version of OSPF and has been obsoleted by WindNet OSPF 2.0. This newer version of OSPF is available as an upgrade for existing OSPF customers. Please contact your Wind River account representative for details on the availability of this upgrade.
- **saloLib, snmpAuxLib, snmpBindLib, snmpEbufLib, snmpIoLib, snmpProcLib, snmpdLib, subagentLib.** These libraries are an older version of the Envoy SNMP agent. A newer version of Envoy for VxWorks 5.5 is available as an upgrade for existing Envoy customers. Please contact your Wind River account representative for details on the availability of this upgrade.
- **telnetLib.** This library has been replaced by **telnetdLib**. See the *VxWorks API Reference* for details.

3.4 Other API Changes

Several sizeable facilities have experienced sweeping changes since the introduction of Tornado 2.0/VxWorks 5.4. Code using earlier versions of the following facilities requires extensive revision.

- **dosFsLib.** A new version of this library, DosFS 2.0, was released in 1999. If you use an earlier version of this library, it is recommended that you upgrade. DosFS 2.0 has support for FAT32 and retains compatibility with more recent Microsoft operating systems.
- **True Flash File System (TrueFFS).** The underlying configuration of this optional product has changed significantly in this release. See the *VxWorks API Reference* for further details.
- **USB Developer's Kit.** Version 1.1.2 of the USB Developer's Kit is an upgraded version for developers writing code for USB devices. See the *USB Developer's Kit Programmer's Guide* and the *USB Developer's Kit Release Notes* for details.

4
Compiler Migration

4.1 GNU Migration

If you are migrating from a Tornado version earlier than 2.1, the GNU toolchain version has changed. The latest GNU toolchain version (identified by the compiler version number) is 2.96+.

This change is most noticeable for C++ users. C++ code compiled with the Tornado 2.0 compiler is not compatible with code produced by the current version of the compiler. Existing C++ code must be rebuilt with the new compiler.

For detailed information on using the GNU toolchain 2.96+ with Tornado 2.2, see the *GNU Toolchain Release Notes for Tornado 2.2*.

4.2 Diab 5.0a Migration

The 5.0x release of the Diab Compiler introduces a new C++ front end and new C++ libraries. With this release, the Diab compiler will be compatible with the latest ISO/IEC 14882:1988(E) C++ standard. In moving to this new release, current users will notice several changes related to providing enhanced C++ support. This chapter outlines the differences between the 4.4x releases and the new 5.0x release.

Related Documentation

This chapter is only intended to provide a brief overview of the expected behavior of the new compiler version. It is strongly recommended that the user consult the following manuals and papers for a complete explanation of options and differences:

- The *Diab C/C++ Compiler User's Manual* is included with the compiler and is available in print, PDF and HTML formats.
- The *Diab 5.0 Release Notes* is included with the compiler. It can be found in the file *installDir***/diab/5.0a/relnote.htm**.

Migrating C Code to Diab 5.0a

C users will find few differences when moving from 4.4x to the 5.0a release. The existing C-language front end remains unchanged from previous versions. C++ users will find numerous differences when moving from 4.x to 5.0a. The changes made to the C++ compiler greatly improve its ANSI compliance and also fix a few serious deficiencies in the previous versions of the compiler.

Backward Compatibility

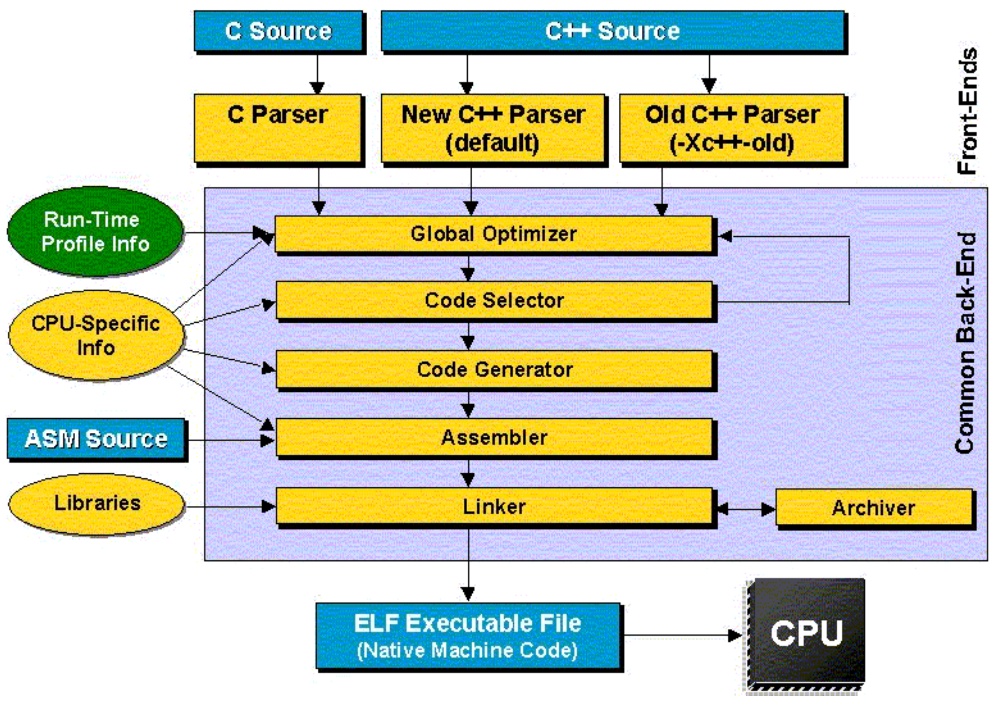
Current C++ users who do not need or want the new C++ front end can continue to use the front end from the older 4.4x version by using the following compiler command line option:

```
-Xc++-old
```

Diab Optimization Technology

Figure 4-1 illustrates that Wind River's optimization technology and support for current back end target architectures remains intact.

Figure 4-1 **Diab Migration**



First Impressions

When you move to the 5.0a release, you will likely see warnings and error messages on code that compiled cleanly with previous versions of the compiler. In general, the new compiler is stricter in both type-checking and syntax-checking, which results in more diagnostic, warning, and error messages. While at first these messages may be troublesome, they can prove quite helpful and instructive on closer inspection.

As stated above, the 5.0a C++ front end expects and demands adherence to ANSI C++ coding standards.

4.2.1 New Features

The 5.0a release includes the following new features:

- Updated compliance with the latest ANSI C++ standard, especially in the areas of templates, STL, and exception handling.
- New C++ libraries including STL.
- Support for pre-compiled headers. (See the Diab documentation for instructions on using pre-compiled headers.)
- Support for DWARF 2:
  - In 5.0a, DWARF 1.1 is the default.
  - Use compiler option **-Xdebug-dwarf2** to generate DWARF 2.
  - In a later release of the 5.0x compiler, DWARF 2 will be the default.

4.2.2 Changes from Previous Versions

- Version 5.0a checks for illegal access to protected and private class members.
- For template processing, comdat is now the default.
  - Previous versions of the compiler required **-Xcomdat**.
  - In the comdat approach, the compiler instantiates every template instance.
  - The linker removes all but one of the multiple instantiations.
- Exceptions and runtime type information are enabled by default.
  - Previous versions of the compiler required the following options:

    ```
    -Xexceptions (-X200)
    -Xrtti (-X205=1)
    ```

  - Exceptions and RTTI can be disabled by using the following options:

    ```
    -Xexceptions-off (-X200=0)
    -Xrtti-off (-X205=0)
    ```

- The new front end compiler is called **etoa**. The 4.4x version of the compiler (**dtoa**) is still present and is used if the **-Xc++-old** option is specified. The driver names (**dplus**, **dcc**) remain unchanged.

NOTE: Do not invoke **ctoa**, **dtoa**, or **etoa** directly. Use the driver programs **dcc** or **dplus**.

- The C++ compiler no longer supports direct assembler functions.
  - Assembly macros, which are more functional, are still supported.
- The C++ libraries do not support locales, wide characters, or the **long double** type.
- Some header files have new names. For example, **exception.h** replaces **except.h**.
- The C++ compiler uses a new error message output format. For example, the following line of code generates an error message:

  ```
  cout << "Hello, world" << endl     //missing semicolon
  ```

  For the above line of code, the 4.4b error format produced the following message:

  ```
  "main.cc", line 6: error (dplus:1247): syntax error after endl, expecting
  ;
  ```

  For the same line of code, the 5.0a error format produces the following message:

  ```
  "main.cc", line 6: error #4065: expected a ";"
      }
      ^
  ```

  Although the wording has changed and a caret ("^") is used to identify the location of the error, the filename and line number remain the same; therefore, 5.0a can continue its interoperability with other programs.

4.2.3 New Compiler Options

The options listed in Table 4-1 are new to the 5.0x compiler:

Table 4-1 **New Options to Diab**

Option	Language(s)	Description
-?	C and C++	Shows commonly used compiler options.
-??	C and C++	Shows less frequently used options.
-?X	C and C++	Shows compiler X options.
-?W	C and C++	Shows compiler W options.
-Xcomdat	C++	Not a new option, but it is now the default.
-Xpch-automatic	C++	Generates and uses pre-compiled headers (PCHs).
-Xpch-create=*filename*	C++	Generates a PCH with the specified name.
-Xpch-diagnostics	C++	Generates a message for each PCH found but unable to be used.
-Xpch-directory=*directory*	C++	Looks for PCHs in the specified directory.
-Xpch-messages	C++	Generates a message each time a PCH is created and used.
-Xpch-use=*filename*	C++	Uses the specified PCH file.
-Xusing-std-on	C++	Operates as if **using namespace std;** had been specified in the code.
-Xusing-std-off	C++	Searches in global scope.

4.2.4 Deprecated Keywords, Options, and Directives

Support for several keywords, compiler options, and directives that are rarely, if ever, used in C++ programming has been dropped in the 5.0a C++ compiler. All the keywords, options, and directives are still supported in the 5.0a C compiler and in C++ if the **-Xc++-old** option is used.

While the items below are listed as deprecated, they may be reinstated in future releases, depending upon customer demand.

Deprecated Keywords

Table 4-2 shows the keywords that are not supported in the 5.0a C++ compiler. See the "Alternative/Comment" column for equivalent solutions.

Table 4-2 **Keywords Deprecated in 5.0a C++ Compiler**

Keyword	Alternative/Comment
extended	Same as **long double**.
interrupt, **__interrupt__**	**interrupt**, **__interrupt__**, and **#pragma interrupt** are not supported in the 5.0a release.
packed, **__packed__**	**packed** and **__packed__** are not supported in the Tornado 2.2 release. (However, they are supported in Diab 5.0a.) **-Xmember-max-align** is supported.
pascal	Reverses the argument list.

Deprecated Options

Table 4-3 shows the options that are not supported in the 5.0a C++ compiler. Many of these options were originally offered to provide compatibility for older, outdated programming styles. In other cases, the options are now incompatible with the standards expected by the new front end. In consideration of the new front end's ANSI compliance, it is recommended that you analyze your application and build system to determine an alternative solution more in keeping with the C++ standard, rather than relying on possibly outdated compiler options.

The 5.0a compiler issues a warning whenever unsupported options are used. This makes it easier to locate them in makefiles or other build systems.

Table 4-3 **Options Deprecated in 5.0a C++ Compiler**

Option	Alternative/Comment
-I@	Searches for user-defined include files in the order specified. The **-I** option is still supported.
-Xold-function-decls-...	Permits the use of old-style function definitions.
-Xpostfix-inc-dec-...	Specifies that the parser should look for an **operator++( )** or **operator--( )** in either the prefix or postfix position.

Table 4-3 **Options Deprecated in 5.0a C++ Compiler**

Option	**Alternative/Comment**
-Xvtbl-...	Controls how vtables are implemented.
-Xcall-MAIN	Generates a call to **_MAIN( )** at the beginning of **main( )**.
-Xshow-inst	Prints all template instantiations to **stderr**.
-Xclass-type-name-visible	Uses old **for** scope rules.
-Xstruct-arg-warning	Issues a warning if a structure argument is larger than the specified number of bytes.
-Xbottom-up-init	Controls how structure and array initializations are made (ANSI specifies top-down).
-Xcpp-no-space	Does not insert spaces around macro names and arguments during preprocessing.
-Xswap-cr-nl	Swaps **\n** and **\r**.
-Xbit-fields-unsigned, **-Xunsigned-bitfields**	Treats bit fields as unsigned. In 5.0a, bit fields retain the sign of their defined type.
-Xbit-fields-signed, **-Xsigned-bitfields**	Treats bit fields as signed. In 5.0a, bit fields retain the sign of their defined type.
-Xbit-fields-compress, **-Xbitfield-compress**	If possible, changes the type of bit fields in order to save space.
-Xdouble-...	Controls the use of the **double** type.

Deprecated Directives

Table 4-4 shows the directives that are not supported in the 5.0a C++ compiler.

Table 4-4 **Directives Deprecated in 5.0a C++ Compiler**

Directive	**Alternative/Comment**
#ident	Inserts a comment into the object file.
#assert, **#unassert**	Supported, but with slight changes in syntax.

4.3 Differences Between GNU and Diab Compilers

The executable names for Diab tools do not vary according to the target as the GNU names do. Diab tools make use of table files to generate target-dependent code and other target-dependent actions. This greatly reduces the number of executables that must be built on a particular host. Table 4-5 lists the tool names for each compiler.

Table 4-5 **Tool Names**

GNU Tool Name	Diab Tool Name
cc*	**dcc** (use the **-t** switch to set a target)
as	**das** (use the **-t** switch to set a target)
ld	**dld** (the **-t** switch selects a target and libraries)
nm*	**ddump**
ar	**dar**

The command to munch C++ source varies between the two tools.

Table 4-6 **Munch Commands**

GNU Munch	Diab Munch
wtxtcl *	**ddump -M**

5
Migrating Projects

5.1 Introduction

Wind River provides a project migration tool, **prjMigrate**, to aid in updating your Tornado project to work with Tornado 2.2/VxWorks 5.5.

A Tornado 2.x project consists of a **.wpj** file, generated files (including a makefile and configuration files), and your source files. **prjMigrate** acts on the **.wpj** file and the generated files. Your source files should not require migration if the APIs they use have not changed. For a list of changed APIs, see *3. VxWorks API Changes*. If the APIs have changed, source code must be migrated. Migration of user source code is beyond the scope of the project migration tool.

Because user modification of makefiles is a common practice, the project migration tool determines whether the makefile has been modified. If so, the tool makes a backup of the makefile, and issues a message explaining that any changes must be manually replicated in the newly generated makefile. The other generated files are simply regenerated.

The project migration tool performs the following tasks:

- Copies the project into the context of the new Tornado tree.
- Makes a backup of the makefile, if it has been modified, and issues a message explaining the need to manually replicate its changes.
- Migrates the **.wpj** file.
- Regenerates the makefile.

- Regenerates the configuration files from the migrated **.wpj** file.
- Provides an interface for interactive query on build macros and components, if you created your project using the command line.

5.2 Using the Project Migration Tool

The **prjMigrate** tool is a data-driven tool that helps you migrate a project created in one version of Tornado to a project in another version of Tornado. It is built on the Tornado project facility.

The tool can work in two modes: **AUTO** and **QUERY**. When run in **AUTO** mode, the tool copies the project into the destination Tornado installation, backs up the makefile if it has been manually modified, migrates the **.wpj** file, and regenerates the makefile and configuration files. **QUERY** mode allows you to discover what equivalent value should be used for build macros and components in the new environment.

NOTE: Before using this tool, it is strongly recommended that you read the help information. You can access the help information at a command line by entering **prjMigrate -h**.

Synopsis

To use the project migration tool, enter the following at a command-line prompt:

```
% prjMigrate -h[elp]
```

For **AUTO** mode, enter the following:

```
% prjMigrate -windbase oldInstallDir {-type projectType [-newproject
prjDirMigrated] oldPrjDir }
```

For **QUERY** mode, enter the following:

```
% prjMigrate -bsp bspName {-tool toolchain [-component "componentList"] }
```

or

```
% prjMigrate -bsp bspName {-tool toolchain [-macro buildMacro -value
"macroValue"] }
```

The parameters used in the command-line sequences above are explained in *Parameters*, p.31, and *Examples*, p.32.

Parameters

The project migration tool takes the parameters described in Table 5-1.

Table 5-1 **prjMigrate Parameters**

Parameter	Description
-h[elp]	Displays **prjMigrate** usage information.
-windbase *oldInstallDir*	The installation directory of the old Tornado installation, from which the project is being migrated.
-type *projType*	The type of project being migrated. Using **-type** causes **prjMigrate** to work in **AUTO** mode. The possible types are **vxWorks** and **vxApp**.
-newproject *prjDirMigrated*	Optional. The destination directory of the project to be migrated. The default is *installDir***/targetproj/***oldName***Migrated**.
oldPrjDir	The directory of the project being migrated.
-bsp *bspName*	The BSP for the project being migrated.
-tool *toolchain*	The toolchain used; either GNU or Diab.
-macro *buildMacro*	Optional. The build macro you wish to query.
-value "*macroValue*"	Optional. The current value of the macro to query. This consists of your changes to the default value of the old Tornado installation's macro. The query result is the default value of the new Tornado installation's macro, with your changes applied. If this parameter is omitted, the query result is the default value of the new Tornado installation's macro.
-component "*componentList*"	A list of components. The query output is the list of equivalent components of the new Tornado installation.

Examples

Example 5-1 **Convert a Project Using AUTO Mode**

This command-line sequence converts a Tornado 2.0 project, **Project0**, located in *installDir***/target/proj**, to a Tornado 2.2 bootable project. *installDir* is **/tmp/t20**.

```
% prjMigrate -windbase /tmp/t20 -type vxWorks target/proj/Project0
```

As result of this operation, the directory **target/proj/Project0** is copied to the **target/proj/Project0Migrated** directory of the Tornado 2.2 installation. The makefile is backed up if it has been modified manually. **.wpj** is migrated to work with Tornado 2.2. The makefile and configuration files are regenerated, but the source files remain unchanged.

Example 5-2 **Generate a List of Build Macros Using QUERY Mode**

This command-line sequence generates a list of build macros that are available to be queried.

```
% prjMigrate -windbase /tmp/t20 -bsp ads860 -tool gnu
```

Example 5-3 **Build Macro Query Using QUERY Mode**

This command-line sequence generates a list of the values of the build macro **CFLAGS** for the **ads860** BSP with the GNU compiler in both Tornado 2.0 and 2.2.

```
% prjMigrate -windbase /tmp/t20 -bsp ads860 -tool gnu -macro CFLAGS
-value "-mcpu=860"
```

6
Migrating BSPs

This chapter discusses the migration of a BSP to VxWorks 5.5. It includes information on both architecture-independent and architecture-dependent issues.

NOTE: This chapter discusses changing a pre-VxWorks 5.5 BSP so that it will be compatible with VxWorks 5.5. The strategy implied here for BSP migration is to make small changes to an existing BSP to make it operational and compatible. In some cases, however, a different strategy may be called for.

VxWorks 5.5 includes enhanced CPU architecture support for several processor families, especially Intel architectures (see section *6.2.6 Pentium*, p.39). Very extensive migration may be required to make these architecture enhancements available to an existing pre-VxWorks-5.5 BSP. An alternate strategy is to use a VxWorks 5.5 BSP that most closely matches your target hardware as a base for your BSP. Board-specific changes could be rolled into this updated BSP, much as they were when the BSP was first written.

For many boards, this second strategy entails a more time-consuming process than the simple changes described below, and this cost should be weighed against the benefits of the VxWorks 5.5 enhancements. For a detailed description of the architecture-specific features available in VxWorks 5.5, see the VxWorks Architecture Supplement for your CPU architecture.

6.1 Architecture-Independent Changes to BSPs

BSP Makefile Changes and the bspCnvtT2_2 Tool

The BSP makefile has been simplified by minimizing the number of **include** statements needed. In particular, prior to Tornado 2.2, a makefile needed several **includes**, as shown in Example 6-1.

Example 6-1 **Makefile includes Prior to Tornado 2.2**

```
...
include $(TGT_DIR)/h/make/defs.bsp
include $(TGT_DIR)/h/make/make.$(CPU)$(TOOL)
include $(TGT_DIR)/h/make/defs.$(WIND_HOST_TYPE)
...
include $(TGT_DIR)/h/make/rules.bsp
include $(TGT_DIR)/h/make/rules.$(WIND_HOST_TYPE)
...
```

In Tornado 2.2, only two of these **include** statements are necessary:

```
...
include $(TGT_DIR)/h/make/defs.bsp
...
include $(TGT_DIR)/h/make/rules.bsp
...
```

The **defs.bsp** and **rules.bsp** files now include any other files necessary to build your BSP. To modify your BSP, simply comment out the unnecessary **includes** in your makefile.

In addition, a makefile conversion tool, **bspCnvtT2_2**, has been provided in the *installDir*/**host/$(WIND_HOST_TYPE)/bin** directory. You can invoke **bspCnvtT2_2** with the following syntax:

```
% bspCnvtT2_2 bspName1 bspName2 ...
```

bspCnvtT2_2 converts your makefile by performing the following actions:

- Saving your old makefile to a file named **Makefile.old** in your BSP directory.
- Commenting out the unnecessary **includes**.
- Warning you about the use of any hex build flags. (See *Hex Utilities and objcopy*, p.35.)

Hex Utilities and objcopy

The use of Wind River-provided hex and binary utilities, such as **aoutToBinDec** or **coffHexArm**, has been deprecated in favor of the GNU utility **objcopy**. See the *GNU Toolkit User's Guide* chapter on binary utilities for details. It is recommended that you modify your BSP build settings as necessary to use **objcopy**.

New Default Value of WDB_COMM_TYPE

The **WDB_COMM_TYPE** default value has been changed from **WDB_COMM_NETWORK** to **WDB_COMM_END**. If you plan to use a different communication mode, define it explicitly in **config.h**. For example, if by default your BSP sets up WDB communication on a serial line, you should include the following line in **config.h**:

```
...
/* make sure this appears after inclusion of configAll.h */
#define WDB_COMM_TYPE WDB_COMM_SERIAL
...
```

Changes in the Shared Memory Subsystem

A BSP for VxWorks 5.5 requires several modest changes to **config.h** and possibly to **sysLib.c** in order to support the shared memory network and the optional component VxMP. In past releases of VxWorks, you could ensure that the shared memory components were included simply by verifying the inclusion of the shared memory backplane network. But in VxWorks 5.5, VxMP can be configured without support for the shared memory network.

A new component with the inclusion macro **INCLUDE_SM_COMMON** has been added to VxWorks 5.5. Use this macro to test for shared memory support. The majority of BSPs that support shared memory use conditional compilation statements such as the following:

```
#ifdef INCLUDE_SM_NET
    /* shared memory-specific code */
#endif
```

For VxWorks 5.5, these statements must be updated to test for **INCLUDE_SM_COMMON**:

```
#ifdef INCLUDE_SM_COMMON
    /* shared memory-specific code */
#endif
```

When you modify **sysLib.c**, follow the simple rule of replacing all instances of **INCLUDE_SM_NET** with **INCLUDE_SM_COMMON**.

With a few exceptions, you can use the same rule in changing **config.h**. A test for **INCLUDE_SM_NET** is still valid in network-related statements, but is not valid as a test for the common shared memory parameters:

SM_ANCHOR_ADRS	**SM_ANCHOR_OFFSET**	**SM_CPUS_MAX**
SM_INT_ARG1	**SM_INT_ARG2**	**SM_INT_ARG3**
SM_INT_TYPE	**SM_MASTER**	**SM_MAX_WAIT**
SM_MEM_ADRS	**SM_MEM_SIZE**	**SM_OBJ_MEM_SIZE**
SM_OFF_BOARD	**SM_TAS_TYPE**	

These shared memory parameters do not require conditional compilation and can be left defined at all times. (Note that some are defined by default in **configAll.h** and must be undefined before being redefined in **config.h**.)

The definition of **INCLUDE_SM_NET** in **config.h** may also bring in components that have changed in VxWorks 5.5, such as **INCLUDE_NET_SHOW** and **INCLUDE_BSD**. These components are no longer needed; the **smNetShow( )** routine is now in a separate component, called **INCLUDE_SM_NET_SHOW**, and proper BSD or other network configuration is contained in other files.

Changes in Other Run-time Facilities

Several optional products for Tornado 2.2/VxWorks 5.5 have undergone changes that necessitate BSP modifications:

- **True Flash File System (TrueFFS).** For details, see the *VxWorks Programmer's Guide* chapter on the Flash Memory Device Interface. Also consult the library entries for **tffsConfig** and **tffsDrv** in the *VxWorks API Reference*.
- **DosFs 2.0.** This updated version of the DOS file system support for VxWorks necessitates changes to your BSP. Because **dosFsNLib** version 2.0 has been available since shortly after the Tornado 2.0 release, many Tornado 2.0.x and later BSPs may already support it, and do not require modification. Full details and examples of DOS version 2 can be found in the *VxWorks Programmer's Guide* and the *VxWorks API Reference*.

6.2 Architecture-Dependent BSP Issues

The following sections provide the specific steps required to upgrade your BSP to Tornado 2.2/VxWorks 5.5 as well as architecture-specific information related to upgrading a BSP. For additional architecture-specific information, refer to the appropriate *Architecture Supplement* manual for your target architecture (available on WindSurf).

6.2.1 Migration Changes Common to All Architectures

NOTE: Some material in this section overlaps with the information in *6.1 Architecture-Independent Changes to BSPs*, p.34.

The following changes are required for all BSPs, regardless of architecture:

- **Makefile update.** This step is required for all users. Use the **bspCnvtT2_2** script to update the BSP makefile. This script will comment out unnecessary include lines and any existing **HEX_FLAGS** value.
- **TFFS support.** This step is required for BSPs with TFFS support. Remove the inclusion of **sysTffs.c** from the **syslib.c** file.

6.2.2 68K/CPU32

All Wind River-supplied BSPs for the 68K/CPU32 architecture released with the Tornado 2.2 product have been upgraded for use with VxWorks 5.5. Custom VxWorks 5.4-based BSPs require only the modifications described in *6.2.1 Migration Changes Common to All Architectures*, p.37 to upgrade to VxWorks 5.5. No architecture-specific modifications are required.

For more information on using VxWorks with 68K/CPU32 targets, see the *VxWorks for 68K/CPU32 Architecture Supplement*.

6.2.3 ARM

NOTE: This section describes BSP migration from Tornado 2.1\VxWorks 5.4 to Tornado 2.2\VxWorks 5.5. For information on migrating a BSP from Tornado 2.0.x to Tornado 2.1, see the *Tornado for ARM Release Notes and Architecture Supplement* manual available on WindSurf.

In addition to the steps described in *6.2.1 Migration Changes Common to All Architectures*, p.37, the following ARM-specific migration changes are required:

- **For assembly files only.** The new macros **GTEXT**, **GDATA**, **FUNC**, and **FUNC_LABEL** have been added to assist in porting assembly files. The leading underscores in global assembly label names should be removed. Using these macros allows source compatibility between Tornado 2.1.x and Tornado 2.2.
- **Diab support.** Due to differences in assembler syntax between the GNU and Diab toolchains, you need to change any GNU assembly macros to Diab syntax. For more information on Diab assembly syntax, see the *Diab C/C++ Compiler for ARM User's Guide*.

For more information on using VxWorks with ARM targets, see the *VxWorks for ARM Architecture Supplement*.

6.2.4 ColdFire

In addition to the steps described in *6.2.1 Migration Changes Common to All Architectures*, p.37, the following ColdFire-specific issues should be considered when migrating your custom BSP:

- **Diab support.** This release of VxWorks for ColdFire includes the same basic layout and functionality included with the previous Tornado 2.1/VxWorks 5.4 release. However, the GNU toolchain is no longer supported.

For more information on using VxWorks with ColdFire targets, see the *VxWorks for ColdFire Architecture Supplement*.

6.2.5 MIPS

NOTE: This section describes BSP migration from Tornado 2.1\VxWorks 5.4 to Tornado 2.2\VxWorks 5.5. For information on migrating a BSP from Tornado 2.0.x to Tornado 2.1, see the *Tornado for MIPS Release Notes and Architecture Supplement* manual available on WindSurf.

In addition to the steps described in *6.2.1 Migration Changes Common to All Architectures*, p.37, the following MIPS-specific issues should be considered when migrating your custom BSP to Tornado 2.2/VxWorks 5.5:

- **CPU variants.** MIPS CPUs are now organized by CPU variant. This allows the VxWorks kernel to take advantage of the specific architecture characteristics of one variant without negatively impacting another. As a result, all MIPS BSPs must now include a **CPU_VARIANT** line in the Makefile after the **MACH_EXTRA** line. For example, CPUs which fall into the category of **Vr54xx** variants, use the following line:

```
CPU_VARIANT  =_vr54xx
```

See the *VxWorks for MIPS Architecture Supplement* for a list of MIPS CPUs and their respective **CPU_VARIANT** values.

- **MIPS64 Libraries.** The MIPS64 libraries (**MIPS64gnu**, **MIPS64diab**, **MIPS64gnule**, and **MIPS64diable**) now support 64-bit MIPS devices with ISA Level III and above. In previous versions of VxWorks, these libraries only supported MIPS devices with ISA Level IV and above. For more information on compiler options for MIPS libraries, refer to the *Architecture Supplement*.
- **Alchemy Semiconductor BSPs.** The Alchemy Semiconductor BSP, **pb1000**, has been altered to provide additional support to the **pb1500** BSP. As a result, some changes have been made to the API of the common support for these two BSPs. All macro, driver, and file names previously using **au1000** have been changed to simply **au**. For example, the cache library **cacheAu1000Lib** is now known as **cacheAuLib**. For more details on these changes, refer to the BSP and its supporting drivers.

For more information on using VxWorks with MIPS targets, see the *VxWorks for MIPS Architecture Supplement*.

6.2.6 Pentium

Support for Intel architectures has been greatly enhanced in VxWorks 5.5 as compared with previous versions. In order to take advantage of all of these changes, you may want to use a Wind River-supported VxWorks 5.5 BSP as the basis for your BSP, and backfit existing customizations (such as custom devices or driver extensions) to this new BSP. For a complete description of the architecture-specific enhancements, see the *VxWorks for Pentium Architecture Supplement* and the BSP reference documentation for a target similar to your board. (for example, pcPentium, pcPentium2, and so on).

The following steps outline the individual changes required to support VxWorks 5.5 on an existing BSP. This information enables you to migrate your BSP step by step, as opposed to the approach described above.

In addition to the steps described in *6.2.1 Migration Changes Common to All Architectures*, p.37, the following Pentium-specific issues should be considered when migrating your custom BSP to Tornado 2.2/VxWorks 5.5:

- The new CPU types **PENTIUM2**, **PENTIUM3**, and **PENTIUM4** have been added and **CPU_VARIANT** has been removed. Thus, **CPU_VARIANT** should be replaced with a new CPU type that is appropriate for your processor.
- Three new code selectors, **sysCsSuper**, **sysCsExc**, and **sysCsInt**, have been added for this release, and **sysCodeSelector** has been removed. In existing BSPs, **sysCodeSelector** should be replaced with **sysCsSuper**.
- The **ROM_IDTR**, **ROM_GDTR**, **ROM_GDT**, **ROM_INIT2** offset macros have been removed due to improvements in the GNU assembler (GAS). These macros are no longer used by **romInit.s**.
- For assembly files only: the new macros **GTEXT**, **GDATA**, **FUNC**, and **FUNC_LABEL** have been added to assist in porting assembly files. The leading underscores in global assembly label names should be removed. Using these macros allows source compatibility between Tornado 2.0.2 and Tornado 2.2.
- For assembly files only: replace **.align** with **.balign**.
- Boot images from earlier releases of VxWorks no longer function with VxWorks 5.5 images and must be rebuilt.
- The PC host utility **mkboot** now works with known VxWorks names, but may not work with user-provided names unless they are of type binary (***.bin**). For all other images, there are two options:
  - Rename your image to **bootrom.dat** before running **mkboot**.
  - Modify **mkboot.dat** to support your names. Follow the examples given in the **mkboot.bat** file.
- Power management is enabled by default. To disable it, modify **config.h**:

  ```
  #undef VX_POWER_MANAGEMENT
  ```

- The default console is now set to COM1. In prior versions of VxWorks, x86 targets set the default console to the VGA console. To use the VGA console, change **config.h**:

  ```
  #define INCLUDE_PC_CONSOLE
  ```

- The configuration parameters for the IDE driver, **ideDrv**, have been removed in favor of the ATA driver, **ataDrv**, that is already used as the default configuration in Tornado 2.0.
- The CPUID structure (**sysCpuId**) has been updated to support Pentium III and Pentium 4 processors. **sysCpuId.version**, **sysCpuId.vendor**, and **sysCpuId.feature** are replaced respectively with **sysCpuId.signature**, **sysCpuId.vendorId**, and **sysCpuId.featuresEdx**.
- **INT_VEC_GET( )/XXX_INT_VEC** have been replaced with **INT_NUM_GET( )/INT_NUM_XXX**, respectively. Although older macros are available in this release for backward compatibility, they will be removed in the next release.
- The routine **sysCpuProbe( )** now understands Pentium III and Pentium 4 processors.
- The routine **sysIntEoiGet( )** has been updated.
- The local and IO APIC/xAPIC drivers, **loApicIntr.c**, **ioApicIntr.c**, and **loApicTimer.c**, now support the xAPIC in Pentium 4. The show routines for these drivers have been separated and contained in **loApicIntrShow.c** and **ioApicIntrShow.c**, respectively.

For more information on using VxWorks with Pentium targets, see the *VxWorks for Pentium Architecture Supplement*.

6.2.7 PowerPC

In addition to the steps described in *6.2.1 Migration Changes Common to All Architectures*, p.37, the following PowerPC-specific changes are required to migrate your custom BSP to Tornado 2.2/VxWorks 5.5:

- Use of the **vxImmrGet( )** routine is deprecated. Existing BSPs implement this routine differently; some return the entire IMMR register, while others mask off the PARTNUM bits. BSPs' existing behavior is unchanged.

  The preferred replacements for this routine are **vxImmrIsbGet( )** and **vxImmrDevGet( )**, which are implemented in **vxALib.s** and should not be overridden by the BSP. Standard Wind River drivers use the new interface.

- Some early MPC74xx/AltiVec support included a routine, typically **vmxExcLoad( )**, to initialize the AltiVec exception vectors. For example:

  ```
  {
  bcopy ((char*)(LOCAL_MEM_LOCAL_ADRS + 0x0100),
          (char*)(LOCAL_MEM_LOCAL_ADRS + _EXC_VMX_UNAVAIL),
          SIZEOF_EXCEPTION);
  bcopy ((char*)(LOCAL_MEM_LOCAL_ADRS + 0x0100),
          (char*)(LOCAL_MEM_LOCAL_ADRS + _EXC_VMX_ASSIST),
          SIZEOF_EXCEPTION);
  }
  ```

  Such code must be removed. AltiVec exception vectors are initialized by **altivecInit( )**.

The following change is optional for PowerPC BSPs:

- Assembly files can be converted to use the Wind River standard macros defined in *installDir***/target/h/arch/ppc/toolsPpc.h**:

  FUNC_EXPORT
  FUNC_IMPORT
  _WRS_TEXT_SEG_START
  FUNC_BEGIN
  FUNC_LABEL
  FUNC_END

  Converting assembly files in this way is not generally required. However, conversion (especially to **_WRS_TEXT_SEG_START**) occasionally fixes a silent bug.

For more information on using VxWorks with PowerPC targets, see the *VxWorks for PowerPC Architecture Supplement*.

6.2.8 XScale/StrongARM

NOTE: This section describes BSP migration from Tornado 2.1\VxWorks 5.4 to Tornado 2.2\VxWorks 5.5. For information on migrating a BSP from Tornado 2.0.x to Tornado 2.1, see the *Tornado for StrongARM/XScale Release Notes and Architecture Supplement* manual available on WindSurf.

In addition to the steps described in *6.2.1 Migration Changes Common to All Architectures*, p.37, the following XScale/StrongARM-specific changes are required to migrate your custom BSP to Tornado 2.2/VxWorks 5.5:

- For assembly files only: the new macros **GTEXT**, **GDATA**, **FUNC**, and **FUNC_LABEL** have been added to assist in porting assembly files. The leading underscores in global assembly label names should be removed. Using these macros allows source compatibility between Tornado 2.1.x and Tornado 2.2.
- Diab support. Due to differences in assembler syntax between the GNU and Diab toolchains, you need to change any GNU assembly macros to Diab syntax. For more information on Diab assembly syntax, see the *Diab C/C++ Compiler for ARM User's Guide*.

For more information on using VxWorks with XScale/StrongARM targets, see the *VxWorks for Intel XScale/StrongARM Architecture Supplement*.

6.2.9 SuperH

In addition to the steps described in *6.2.1 Migration Changes Common to All Architectures*, p.37, the following SuperH-specific changes are required to migrate your custom BSP to Tornado 2.2/VxWorks 5.5:

- **Power management setup.** This step is required for BSPs where processor power management is enabled. In the **sysHwInit( )** routine in **sysLib.c**. initialize the **vxPowerModeRegs** structure depending on the SuperH processor used.
- **Diab support.** This step is only required if the BSP will be built with the Diab toolchain. Assembler files should be updated to use the **.short** directive instead of the **.word** directive.
- **Use of NULL.** In previous releases, NULL was defined as integer zero. This definition has been changed to match the C standard to a void pointer. To avoid compiler warnings, make sure NULL is only used for pointer assignments.

For more information on using VxWorks with SuperH targets, see the *VxWorks for Hitachi SuperH Architecture Supplement*.

A
Writing Portable C Code

This chapter describes how to write compiler-independent portable C code. The goal is to write code that does not require changes in order to work correctly with different compilers.

Wind River has conducted an analysis of its own VxWorks code base, with an eye to compiler independence issues. This chapter is based on the findings of that analysis.

The code changes proposed in this chapter do not cover the native host tools, including VxSim and any other tools that are compiled both by native compilers and by cross-compilation.

This chapter also does not cover C++ portability; there are fundamental differences in the GNU and Diab C++ implementations. Future editions of this document will address the portability of C++ code.

Background

While the ANSI C and compiler specifications are detailed, they still allow each compiler to implement the standard differently. The result is that much source code is not truly portable among compiler systems. In order for the compiler to generate code in a specific manner, engineers must insert special non-ANSI-defined instructions into the code.

The information in this chapter is part of Wind River's effort to support multiple compiler systems without requiring major source changes, and to add support for new compiler systems in the future without creating compatibility conflicts.

Analysis of Wind River's existing code base reveals three main areas where non-ANSI compiler features are used to generate runtime code:

- packed structure definitions
- in-line assembly code
- alignment of data elements

NOTE: This chapter is limited in scope to insuring that code is portable specifically between the GNU and Diab compilers.

NOTE: The scope of this chapter is further limited to specify GCC version 2.96 and Diab version 5.0 as the baseline compilers. Earlier versions are not truly compatible with each other and may not have all the necessary support to implement the conventions introduced here.

A.1 Portable C Code

A.1.1 Data Structures

Some structure definitions are bound by external restrictions. It is common practice to use a structure definition to document the register layout of hardware devices. Using a structure to define the layout of data packet contents received from another system is also common. This can create problems because the ANSI specification allows compilers to insert padding elements within structure definitions in order to optimize data element accesses. In these situations, doing so would make the structure declaration incompatible with the expectations and restrictions of the outside world. The compiler offsets from the start of the structure to the start of the data element would not match what is expected. A method is required for identifying certain structures as requiring special compiler treatment, while allowing other structures to be optimized for efficiency.

The common term for a structure definition without any padding is *packed*. Each of the major compilers has a means to specify packing for a structure, but there is no single recognized standard. GNU and Diab use an attribute statement as part of the declaration. Microsoft compilers use **#pragma** statements.

To specify packing in a compiler-independent manner a macro has been created for use when defining the structure. The **_WRS_PACK_ALIGN(***x***)** macro is used as an attribute declaration. It is placed after the closing curly brace, but before any instance declarations or initializers. By including the **vxWorks.h** header file in your compilation, a toolchain specific header file is included to define this macro appropriately for the compiler tool being used.

For example:

```
struct aPackedStruct {
    UINT32 int1;
    INT16  aShort;
    INT8   aByte;
} _WRS_PACK_ALIGN(1);

struct aPackedStruct {
    UINT32 int1;
    INT16  aShort;
    INT8   aByte;
} _WRS_PACK_ALIGN(1) anInstance = {0x1, 2,3};

typedef  struct {
    UINT8  aByte;
    UINT16 aShort;
    UINT32 aLong;
} _WRS_PACK_ALIGN(1)  myPackedStruct;
```

Specify Field Widths

Always use specific field widths within a packed structure declaration. The basic data type **int** does not have a specific size and should be avoided. The same rule applies for **long**, **unsigned**, and **char**. The **vxWorks.h** header file defines basic data types with explicit sizes. Use these data types in any packed structure (INT8, INT16, INT32, UINT8, and so on). In general, think ahead to architectures with more than a 32-bit native integer.

Avoid Bit Fields

Do not include bit field definitions within a packed structure. Compilers are permitted to start labeling bits with either the least significant bit or the most significant bit. This issue can be dealt with easily by using macro constants to define the specific bit pattern within the data field.

For example:

```
struct aPackedStruct {
    UINT32 int1;
    INT16  aShort;
    INT8   aByte;
} _WRS_PACK_ALIGN(1) anInstance={0x1,2,3};

/* Bits in the aByte field */

#define ABYTE_ERROR  0x01
#define ABYTE_OFLOW  0x02
#define ABYTE_UFLOW  0x04
#define ABYTE_DMA    0x08
#define ABYTE_POLL   0x10
```

A.1.2 In-Line Assembly

In-line assembly is a more difficult issue because it involves both the compiler and the assembler. In Wind River's case, the code base fortunately uses MIT assembler syntax throughout, which many assemblers are compatible with. In-line assembly in portable C code is by its nature not portable across architectures. Thus, the real question for in-line assembly is compiler portability.

The current compilers differ significantly about how to include assembly instructions without interfering with, or suffering interference from, the compiler's optimization efforts. In the absence of an ideal solution, in-line assembly should only be used if it does not interact with the surrounding C code. This means that acceptable in-line assembly will not interact with C variables or return values. Code that cannot meet this limitation should be written entirely in assembly language.

The **vxWorks.h** header file defines a **_WRS_ASM** macro to be used to insert in-line assembly instructions in a C function.

For example (PowerPC):

```
VOID foo (void)
    {
    routineA (args);
    _WRS_ASM(" eieio; isync;");
    routineB (args);
    }
```

Assume that the compiler is not free to optimize or reorder in-line assembly code with respect to surrounding C code.

A.1.3 Static Data Alignment

Sometimes it is necessary to align a global static data element or structure on a specific boundary type. This typically happens with CPU-specific data structures that need cache boundary alignment.

To handle this situation, another toolchain-specific macro has been introduced. The macro **_WRS_DATA_ALIGN_BYTES(***bytes***)** aligns the following data element with the byte alignment specified.

For example:

```
_WRS_DATA_ALIGN_BYTES(16) int myData = 0xFE00235F;
```

This alignment macro should only be used with global data that has been initialized. Uninitialized data may not be placed in the data section, and the macro may not have the desired effect. Uninitialized data can be handled at runtime using **memalign( )** or other suitable functions.

A.1.4 Runtime Alignment Checking

Checking the Alignment of a Data Item

You may need to know the alignment of a particular data item at runtime. Most compilers provide an extension for accessing this information, but there is no recognized standard for it. In the case of Wind River source code, the macro **_WRS_ALIGNOF(***x***)** is used to return the alignment of an item in byte units.

```
if (WRS_ALIGNOF(itemA) < 4)
    {
    printf ("Error: itemA is not longword aligned");
    }
```

Verifying Pointer Alignment

Pointers can be deliberately cast to be a pointer to a different type of object with different alignment requirements. Strict type checking at compile time is beneficial, but there are situations in which this checking must be performed at runtime. For this purpose, the macro **_WRS_ALIGN_CHECK(***ptr***,** *type***)** is provided. This macro evaluates to either TRUE or FALSE. TRUE is returned if the pointer **ptr** is aligned sufficiently for an item of type *type*. The test is normally done by examining low-order bits of the pointer's value.

```
void * pVoid;

if (!WRS_ALIGN_CHECK(pVoid, long))
    {
    printf ("Error: pVoid is not longword aligned");
    }
```

Unaligned Accesses and Copying

You may need to access data that may not be correctly aligned at runtime. It is recommended in this situation that you copy the data to a structure or other area that is properly aligned. After the data has been copied, it can be accessed without the possibility of causing unaligned access exceptions. The macro provided for this purpose is **_WRS_UNALIGNED_COPY(***pSrc***,** *pDst***,** *size***)**. While the standard VxWorks **bcopy( )** function could be used for this purpose, most compilers can do short copies more efficiently on their own. Using the macro is therefore desirable for performance reasons.

The following example shows both **_WRS_ALIGN_CHECK** and **_WRS_UNALIGNED_COPY** used together to check an unknown pointer. If it is sufficiently aligned, the pointer can be cast to some other type and the item can be accessed directly. If the pointer is not aligned, the unaligned macro is used to copy the data element to a usable variable location.

```
struct structA {
    long item1;
    long item2;
    char item3;
} itemA;

void * pVoid;
long aLong;

if (WRS_ALIGN_CHECK(pVoid, (struct structA)))
    {
    /* Alignment is okay, reference directly */
    aLongItem = ((struct structA *)pVoid)->item2;
    }
else
    {
    /* alignment is not okay, use unaligned copy */
    _WRS_UNALIGNED_COPY(pVoid,&aLong,sizeof(aLong));
    }
```

A.1.5 Other Issues

Follow Strict ANSI Compilation

Compilation units should be compiled with strict ANSI protocols in effect. This requires detailed prototype declarations. For the GNU compiler system, the compiler flags should include the following:

```
-Wall -W -Wmissing-declarations -Wstrict-prototypes -Wmissing-prototypes
```

Remove Compiler Warnings

Portable code must be as free of warnings as possible. Apply the strictest possible code checking and fix all reported warning situations. Compilers identify non-portable code issues well when doing error checking.

Avoid Use of Casts

Each and every cast represents a potential error. It is a common practice to use a cast to fix warnings reported by the compiler. However, each instance must be examined carefully to insure that a cast is the appropriate action. Often a warning indicates an actual error in argument passing that must be corrected. Using a cast overrides the compiler's ability to detect an actual error in data usage that may prove to be significant.

Avoid inline Keyword

The C **inline** keyword is to be avoided until the GNU and Diab compilers can implement it in a consistent manner. The current ANSI Specification, C99, does call for an **inline** keyword. However, at this time, neither compiler fully supports this specification. Although each accepts the **inline** keyword, there are subtle but significant differences in the implementation. An update to this document will be issued when support for the **inline** keyword is available.

Avoid alloca( ) Function

Many compilers support the **alloca( )** function as an extension to the C language. This normally allocates storage from the stack, as any declared variable would. Since the storage area is on the stack, this area does not need to be freed, as the stack is restored upon exiting the function.

While **alloca( )** is widely used in code from other OS programming models, it does not suit VxWorks very well. While other OSs may support automatic stack expansion and stack checking, VxWorks does not. In embedded programming, predictable timing and stack usage can be very important. Code for VxWorks should definitely avoid the use of the **alloca( )** function. Allocate the storage directly on the stack, or use **malloc( )** and **free( )** if necessary.

Take Care with void Pointer Arithmetic

It is common to use pointer arithmetic on **void** pointer data types. Because the size of a **void** item should be unspecified, the ANSI standard does not allow this practice. The GNU compiler, however, did allow it and assumed the data size to be one byte, the same as a **char** data type.

For example, the following code fragment is faulty:

```
{
void * pVoid;

pVoid += 1;                    /* WRONG */
pVoid++;                       /* WRONG */
pVoid = pVoid + sizeof(char); /* WRONG */
}
```

The example above is faulty because ANSI pointer arithmetic is based on the size of the object pointed to. In the case of a pointer to a **void**, the size is undefined. For the first faulty statement in the example above, the only correct implementation is as follows:

```
{
void * pVoid;

pVoid = (char *)pVoid + 1; /* RIGHT */

(char *)pVoid++;           /* WRONG */
(char *)pVoid += 1         /* WRONG */
}
```

The last two statements in the example above are still faulty because ANSI does not allow casts to be used for an **lval** type expression.

Use volatile and const Attributes

Proper use of the **volatile** and **const** attributes can result in better error detection by the compiler. The **volatile** keyword is essential for all data elements whose value can be changed by an agent outside of the current thread of execution. (For example, a device performing DMA, another task sharing the data, or an interrupt routine sharing the data.) Failure to tag a shared data element with **volatile** can generate intermittent system faults that are difficult to track.

VxWorks 5.4 and earlier always used the **–fvolatile** compiler option to force all pointers to be treated as pointing to a **volatile** data item. Portable code should not rely on this mechanism in the future. The compiler can optimize much more effectively when the correct attributes of all data elements are known.

The **const** attribute should be used to indicate to the compiler that an argument is strictly an input argument, and that its value is unchanged by this routine. This helps the compiler to perform error detection and allows it to better optimize the code.

Misuse of the register Attribute

The misuse of the **register** (that is, **FAST**) attribute is quite common. In Wind River's case, analysis of the code base revealed that a number of subroutines have been coded where the input arguments were all labeled with the **FAST** attribute, as well as a number of local routine values. The current compilers are able to do very good optimization of routines without requiring any use of the **FAST** attribute in the code base. Overuse of the attribute can actually prevent the compiler from performing effective optimization of the code. For the large majority of code now, the **FAST** attribute is simply unnecessary. It should only be used sparingly in situations where there is a large number of local variables, only a few of which are referenced often.

Avoid vector Name

A Motorola extension to the ANSI C specification makes the word **vector** a keyword. Runtime code should not use **vector** as a variable name. This change in the compilers supports the Altivec processor and the built-in DSP functions it provides. Your code should avoid the use of **vector** as a variable name.

Statement Labels

The ANSI specification requires labels to be associated with statements. It is not uncommon to place labels at the end of a code block without any associated statement at all. With strict ANSI checking this is an error. In the code below the default label is not connected to a statement. To correct this problem, either remove the label or add a null statement to give the label a proper point of connection.

```
switch (xxx)
    {
    case X: statement;
    case Y: statement;
    default:    /* WRONG - no statement here */
    }
```

Summary of Compiler Macros

For each supported C toolchain, the macros described in this section are defined when **vxWorks.h** is included in your compilation.

_WRS_PACK_ALIGN(n)

This macro is used to specify the packing and alignment attributes for a structure. Packing insures that no padding will be inserted and that the minimum field alignment within the structure is one byte. The user can specify the assumed alignment for the structure as a whole with the argument *x*, in bytes. The value *x* is expected to be a power of two value (1,2,4,8,...). The size of the structure is then a multiple of this value. If the overall structure alignment is 1, the compiler assumes that this structure, and any pointer to this structure, can exist on any possible alignment. For an architecture that cannot handle misaligned data transfers, the compiler is forced to generate code to access each byte separately and then to assemble the data into larger **word** and **longword** units.

The macro is placed after the closing brace of the structure field description and before any variable item declarations, or **typedef** name declarations.

Always specify fields with explicit widths, such as UINT8, UINT16, INT32, and so on. Do not use bitfields in a packed structure.

_WRS_ASM("X")

This macro is used to insert assembly code within a C function declaration. The inserted code must not interact with C variables or try to alter the return value of the function. The code uses the MIT assembly-language mnemonics and syntax.

It is assumed that the compiler does not optimize or reorder any specified in-line assembly code. The insertion of a null in-line assembly statement can be used to prevent the compiler from reordering C code before the statement with C code that follows the statement.

_WRS_DATA_ALIGN_BYTES(n)

This macro is used in prefix notation to declare an initialized C data element with a special alignment. The argument *n* is the alignment in byte units (1, 2, 4, 8, 16, and so on). This is normally used only with initialized global data elements. Use this macro with caution: overuse of this macro can result in poor memory utilization. If large numbers of variables require special alignment, it may be best to declare them in separate sections directly in assembler. The linker loader could then fit them together in an optimal fashion.

_WRS_GNU_VAR_MACROS

The GNU compiler system created a means to pass a variable number of arguments to pre-processor macro functions, and there are a few special instances in the VxWorks code base that use the GNU-defined syntax. Since then, the ANSI standards committee has defined an ANSI standard that is different from this practice. Currently, the GNU compiler 2.96 does not yet support the ANSI standard, and the Diab compiler supports *only* the ANSI standard.

Code that does use variadic macros should define them both for GNU and for the ANSI standard (Diab). Rather than select upon the toolchain or compiler name, a new macro feature name, **_WRS_GNU_VAR_MACROS**, has been created. The GNU toolchain defines this macro; the Diab toolchain does not.

If you want to port your code to another toolchain, you must choose between supporting the GNU-style syntax or the ANSI standard syntax. For example, the following code fragment demonstrates the use of an **#ifdef** statement to make this choice between GNU and ANSI:

```
#ifdef _WRS_GNU_VAR_MACROS /* GNU Syntax */
#define MY_MACRO(x, y, args...) printf (x, y, args)

#else /* ANSI Syntax */

#define MY_MACRO(x, y, ...) printf (x, y, __VA_ARG__)

#endif
```

In the future, when the GNU toolchain does support the ANSI standard, the **#ifdef**s based on this macro can be removed from the code base.

_WRS_ALIGNOF(item)

This macro returns the alignment of the specified item, or item type, in byte units. Most structures have alignment values of 4, which is the normal alignment of a **longword** data value. Data items or types with greater alignment values return an appropriate alignment value, which is expected to be a power of two (1, 2, 4, 8, 16, and so on).

_WRS_ALIGN_CHECK(ptr, type)

This macro returns a boolean value, either TRUE or FALSE. A return of TRUE indicates that the pointer value is sufficiently aligned to be a valid pointer to the data item or type. The expected implementation is to examine the low-order bits of the pointer value to see whether it is a proper modulo of the alignment for the given type.

_WRS_UNALIGNED_COPY(pSrc, pDst, size)

This macro is a compiler-optimized version of the standard Wind River **bcopy** operation. It moves a data block from the source location to the destination location. This macro allows the compiler to optimize the copy operation based on the data types of the pointers and the size of the block. This macro is designed to be used in high-performance situations; the size of the block is expected to be small. Misuse of the macro for other situations should be avoided.

A.2 Tool Implementation

This section discusses how the current VxWorks runtime code is organized to facilitate the addition and maintenance of new compiler tools.

A.2.1 New Tool Macros File

The definition of the new tool-based macros must be placed into the compilation units. This has been achieved by modifying the **target/h/vxWorks.h** file to include a new **toolMacros.h** file. This file defines the new macros and any other tool-based options that apply globally to runtime compilations. All runtime-compiled files must include **vxWorks.h** so that they will also include the new tool macro definitions.

A.2.2 New Tool Directories

Each toolchain provides the **toolMacros.h** file in a separate directory. Changes have been made in **vxWorks.h** to find the **toolMacros.h** file based on the preprocessor variable **TOOL_FAMILY**. For backward compatibility, if **TOOL_FAMILY** is not explicitly defined, its value is generated from the value of the build macro **TOOL**.

This new toolchain-specific directory structure is intended to make it easy for all tool-related files to be located in a common directory separate from other tool system files. It also makes it unnecessary to modify any common files in the system just to add a new tool system. Add the new tool directory **target/h/tool/***newTool* and then perform a system build; this triggers the system to reanalyze all toolchains and rebuild the toolchain database.

A.2.3 BSP Makefile Changes

All VxWorks 5.5 (Tornado 2.2) BSP makefiles require modification to replace the old **make** tool include file path with the new tool-based path. For this release, all changes are made to the platform release files.

Comment out the following lines from each BSP makefile by inserting a **#** in front of them. The remaining **include** lines pull in all the needed make files.

```
#include $(TGT_DIR)/h/make/make.$(CPU)$(TOOL)
#include $(TGT_DIR)/h/make/defs.$(WIND_HOST_TYPE)
#include $(TGT_DIR)/h/make/rules.$(WIND_HOST_TYPE)
```

A.2.4 Macro Definitions for GNU and Diab

The current implementation of the required macros for the GNU toolchain is as follows (this assumes GCC 2.96):

```
#define _WRS_PACK_ALIGN(x)        __attribute__((packed,aligned(x)))
#define _WRS_ASM(x)               __asm__ volatile (x)
#define _WRS_DATA_ALIGN_BYTES(x)  __attribute__((aligned(x)))
#define _WRS_GNU_VAR_MACROS
#define _WRS_ALIGNOF(x)           __alignof__(x)
#define _WRS_ALIGN_CHECK(ptr,type) \
    (((int)(ptr) & (WRS_ALIGNOF(type) - 1)) == 0 ? TRUE : FALSE)
#define WRS_UNALIGNED_COPY(pSrc,pDst,size) \
    (__builtin_memcpy((pDst, (void *)(pSrc), size))
```

The current implementation of the required macros for the Diab toolchain is as follows (this assumes Diab 5.0):

```
#define _WRS_PACK_ALIGN(x)        __attribute__((packed,aligned(x)))
#define _WRS_ASM(x)               __asm volatile (x)
#define _WRS_DATA_ALIGN_BYTES(x)  __attribute__((aligned(x)))
#undef  _WRS_GNU_VAR_MACROS
#define _WRS_ALIGNOF(x)           sizeof(x,1)
#define _WRS_ALIGN_CHECK(ptr,type) \
    (((int)(ptr) & (WRS_ALIGNOF(type) - 1)) == 0 ? TRUE : FALSE)
#define _WRS_UNALIGNED_COPY(pSrc,pDst,size) \
    (memcpy((pDst), (pSrc), size))
```