



Wind River Diab Compiler Utilities Reference, 5.9.7

31 January 2020

WHEN IT MATTERS, IT RUNS ON **WIND RIVER**

Copyright Notice

Copyright © 2019 Wind River Systems, Inc.

All rights reserved. No part of this publication may be reproduced or transmitted in any form or by any means without the prior written permission of Wind River Systems, Inc.

Wind River, Simics, Tornado, and VxWorks are registered trademarks of Wind River Systems, Inc. Helix, Pulsar, Rocket, Titanium Cloud, Titanium Control, Titanium Core, Titanium Edge, Titanium Edge SX, Titanium Server, and the Wind River logo are trademarks of Wind River Systems, Inc. Any third-party trademarks referenced are the property of their respective owners. For further information regarding Wind River trademarks, please see:

www.windriver.com/company/terms/trademark.html

This product may include software licensed to Wind River by third parties. Relevant notices (if any) are provided for your product on the Wind River download and installation portal, Wind Share:

<http://windshare.windriver.com>

Wind River may refer to third-party documentation by listing publications or providing links to third-party websites for informational purposes. Wind River accepts no responsibility for the information provided in such third-party documentation.

Corporate Headquarters

Wind River
500 Wind River Way
Alameda, CA 94501-1153
U.S.A.
Toll free (U.S.A.): +1-800-545-WIND
Telephone: +1-510-748-4100
Facsimile: +1-510-749-2010

For additional contact information, see the Wind River website:

www.windriver.com

For information on how to contact Customer Support, see:

www.windriver.com/support

Wind River Diab Compiler Utilities Reference, 5.9.7

31 January 2020

TABLE OF CONTENTS

1. Utilities.	1
1.1. Utilities.	1
2. dar Archiver.	2
2.1. dar Archiver.	2
2.1.1. dar Archiver Examples.	4
3. dbcnt Profiling Basic Block Counter.	6
3.1. dbcnt Profiling Basic Block Counter.	6
3.1.1. dbcnt Profiling Basic Block Counter Examples.	7
4. ddump File Dumper.	9
4.1. ddump File Dumper.	9
4.1.1. ddump File Dumper Examples.	16
5. dmake Makefile Utility.	20
5.1. dmake Makefile Utility.	20
6. WindISS Simulator and Disassembler.	21
6.1. Windiss Simulator and Disassembler.	21
6.1.1. Simulator Mode.	
6.1.2. Batch Disassembler Mode.	21
6.1.3. Interactive Disassembler Mode.	22
6.1.4. Windiss Simulator and Disassembler Examples.	23

1. UTILITIES

1.1. Utilities

The following chapters describe utility tools that accompany the compiler suite; this chapter details the common command-line options.

Common Command-Line Options

All tools in the Wind River suite accept the following command-line options where meaningful. They are repeated here for convenience.

Show Option Summary (-?)

-?, -h, --help

Show synopsis of command-line options.

Read Command-Line Options from File or Variable (-@name, -@@name)

-@name

Read command-line options from either a file or an environment variable. When **-@name** is encountered on the command line, the tool first looks for an environment variable with the given name and substitutes its value. If an environment variable is not found then it tries to open a file with given *name* and substitutes the contents of the file. If neither an environment variable or a file can be found, an error message is issued and the tool terminates.

-@@name

Same as **-@name**; also prints all command-line options on standard output.

Redirect Output (-@E=file, -@E+file, -@O=file, -@O+file)

-@E=file

Redirect any output to standard error to the given file.

-@O=file

Redirect any output to standard output to the given file.

Use of "+" instead of "=" will append the output to the file.

2. DAR ARCHIVER

2.1. Dar Archiver

Create and maintain an archive of files of any type, with special features for object files.

Syntax

```
dar command [position-name] archive-file [name] ...
```

Description

The dar command maintains files in an archive.

Archives can contain files of any kind. However, object files are handled in a special way. If any of the included files is an object file, the archiver will generate an invisible symbol table in the archive. This symbol table is used by the linker to search for missing identifiers without scanning through the whole archive.

An archive file consisting only of object files is also called a library, and so the archiver is often referred to as a *librarian*.

command is composed of a hyphen (-) followed by a command letter. One or more optional modifier letters for some commands may either be concatenated to the command letter, or may be given as separate option arguments (see below for examples).

position-name is the name of a file in the archive used for relative positioning with the -r and -m commands.

archive-file is the archive file pathname.

name is one or more files in the archive. Multiple *name* arguments are separated by whitespace.

dar Commands

dar commands and modifiers are as follows. Modifiers are shown in brackets. See also [Utilities on page 1](#), *Common Command-Line Options*.

-d [*lv*]

Delete the named files from the archive.

-m [*abiv*]

Move the named files. If any of the [*abi*] modifiers are employed, the *position-name* argument must be present and the files will be positioned in the same manner as with the -r command. Otherwise the files are moved to the end of the archive.

-p [*sv*]

Print the contents of the named files on the standard output. This is useful only with text files in an archive; binary files, e.g., object files, are not converted and so are not normally printable.

-q [*cflv*]

Quickly append the named files at the end of the archive without checking whether the files already exists. If the archive contains any object files, the symbol table file will be updated. If the [*f*] modifier is used, the files will be appended without updating the symbol table file, which is considerably faster. Use the -s command when all files have been inserted in the archive to update the symbol table.

-r [abciluv]

Replace the named files in the archive. New files are placed at the end of the archive unless one of the **[abi]** modifiers is used. If so, *position-name* must be given to specify a position in the archive. With the **[bi]** modifiers, the named files will be positioned before *position-name*; with the **[a]** modifier, after it.

If the archive does not exist, create it.

If the **[u]** modifier is specified, then only files with a modification date later than the corresponding files in the archive will be replaced.

-s [lR]

Update the symbol table file in the archive. Used when the archive is created with the **-qf** command.

-t [sv]

List a table of contents for the archive on the standard output.

-V

Print the version number of **dar**.

-x [lsv]

Extract the named files from the archive and place them in the current directory. The archive is not changed.

	Use With Commands	
a	-m -r	Insert the named files in the archive after the file <i>position-name</i> .
b	-m -r	Insert the named files in the archive before the file <i>position-name</i> . Same as " i " modifier.
c	-q -r	Does not display any message when a new archive <i>archive-file</i> is created.
Dpathname		
	-q -r	When adding to or replacing files in an archive, prefix <i>pathname</i> to name of each file to be stored to access it in the file system (but do not store the additional <i>pathname</i> in the symbol table).
f	-q	Append files to the archive, without updating the symbol table file. If any of the files already exist, multiple copies will exist in the archive. The next time the -s command is used dar will delete all copies but the last of the files with the same name.
i	-m -r	Insert the named files in the archive before the file <i>position-name</i> . Same as " b " modifier.
j	-q -r	Store a path prefix if given with an object file in the archive symbol table instead of just the base filename.

	Use With Commands	
		NOTE: The path prefix becomes part of the name in the archive. Thus, if a single file x.o is added once as x.o and a second time as lib/x.o using the " j " option, it will be stored twice in the archive.
l	-d -q -r -s -x	Place temporary files in the current directory instead of the directory specified by the environment variable TMPDIR , or in the default temporary directory.
s	-p -t -x	Same as the -s command.
u	-r	Replace those files that have a modification date later than the files in the archive.
v	-d -m -p -q -r -t -x	Verbose output.
R	-s	Sort object files in the archive so that the linker does not have to scan the symbol table in multiple passes.

[dar Archiver Examples on page 4](#)

Some later examples build on earlier examples.

2.1.1. dar Archiver Examples

Some later examples build on earlier examples.

New Archive

Create a new archive **lib.a** and add files **f.o** and **h.o** to it (the **-r** command could also be used):

```
dar -q lib.a f.o h.o
```

Modify Above Archive: Replace File, Add File

Replace file **f.o**, and insert file **g.o** in archive **lib.a**, and also display the version of **dar**. Without the "**a**" modifier, the new file **g.o** would be appended to the end of the archive. With the "**a**" modifier and the first **f.o** acting as the *position-name* in the command, new file **g.o** is inserted after the replaced **f.o**:

```
dar -rav f.o lib.a f.o g.o
```

Alternative command for Example 2

The previous examples can also be given in the following form with the modifier letters given as separate options. The first item following **dar** must always be the command from *dar Commands*.

```
dar -r -a -v f.o lib.a f.o g.o
```


Quick Append to Archive

Quickly append **f.o** to the archive **lib.a**, without checking if **f.o** already exists. This operation is very fast and can be used as long as the archive is later cleaned with the **-sR** command (see below):

```
dar -qf lib.a f.o
```

Cleanup Archive After Quick Appends

Cleanup archive **lib.a** by creating a new sorted symbol table and removing all but the last of files with the same name. This is useful after many files have been added with the **-qf** option:

```
dar -sR lib.a
```

Extract File from Archive Without Changing Archive

Extract **file.c** from archive **source.a** and place it in the current directory. The archive is unchanged.

```
dar -x source.a file.c
```

Delete File from Archive Permanently

Delete **file.c** files from archive **source.a**. The file is deleted without being written anywhere:


```
dar -d source.a file.c
```

Parent topic: [dar Archiver on page 2](#)

3. DBCNT PROFILING BASIC BLOCK COUNTER

3.1. Dbcnt Profiling Basic Block Counter

Display profile data collected from one or more runs of a program.

 **Note:** Profiling using -Xblock-count and -Xfeedback is not supported for VxWorks applications.

Syntax

```
dbcnt [-f profile-file] [-h n] [-l n] [-n] [-t n] source-file , ...
```

Description

The dbcnt command displays the number of times each line in a source program has been executed.

It can also be used to show “coverage” information (see [Coverage](#)).

The files to be measured must be compiled with the -Xblock-count option. By definition, a basic block is a segment of code with exactly one entrance and one exit. Thus, all statements in a basic block will have the same count. Compiling with -Xblock-count causes the compiler to insert code into each basic block to record each execution of the block. Each time the resulting program is run, the profile data is stored in the file named in the environment variable **DBCNT**. If **DBCNT** is not set, the file **dbcnt.out** will be used. If the program is executed more than once, the new profile data will be added to the existing **DBCNT** file.

After the profile data has been collected and returned to the host, to display one or more source files together with their line counts, enter the command:

```
dbcnt [options] source-file1 , source-file2 , ...
```

If the name of the **DBCNT** file is not **dbcnt.out**, use the -f option to provide the pathname of the actual file with the line counting information. See below for examples.

dbcnt options are as follows. See also [Utilities on page 1](#), *Common Command-Line Options*.

dbcnt Options

Refer to the following list of dbcnt commands.

-f file

Read profile data from *file* instead of **dbcnt.out**.

-h n

Do not print lines executed more than *n* times.

-l n

Do not print lines executed fewer than *n* times.

-n

Print the line number of every source line.

-tnPrint the *n* most frequently executed lines.**-V**Print the version number of **dbcnt**.

Files

Files processed by **dbcnt** must be unique in their first 16 characters.

Output File for Profile Data

dbcnt.out

Default output file for profile data.

DBCNT

Environment variable giving the name of the profile data file.

[dbcnt Profiling Basic Block Counter Examples on page 7](#)

Refer to the following example of the **dbcnt** command.

3.1.1. dbcnt Profiling Basic Block Counter Examples

Refer to the following example of the **dbcnt** command.The file **file.c** (shown annotated below) is compiled with:

```
dcc -Xblock-count -o file file.c
```

When executed, the following output is produced:

```
47 numbers are multiples of 3 or 5.
```

dbcnt is used to show how many times each line is executed:

```
dbcnt file.c
```

dbcnt produces the following output:

```


file.c (1 run(s)):
    main()
    {
1      int i = 100, n = 0;
1
101     while(i > 0) {
100         if ((i % 3) == 0 || (i % 5) == 0) {
67

```

```

47         n++;
47     }
100     i--;
100 }
1     printf("%d numbers are multiples of 3 or 5.\n",n);
    }

```

 **Note:** When a source line contains more than one basic block, such as the if statement above, empty lines are added to show the count of the basic blocks after the first.

The following will find the 100 most frequently executed source lines in a program:

```
dbcnt -n -t100 *.c
```

Coverage

The following will find all source lines which did not execute in a program:

```
dbcnt -h0 -l0 -n *.c
```

(The second option, -l0, is hyphen, lower-case L, 0.)

Notes

The functions `__dbini()` and `__dbexit()` must exist in the standard library in order for the linker to be able to link the files compiled with the `-Xblock-count` option.

For information on support for file I/O and environment variables in an embedded environment, see the *Use in an Embedded Environment* chapter of the *Wind River Diab Compiler User's Guide* for your architecture.

Parent topic: [dbcnt Profiling Basic Block Counter on page 6](#)

4. DDUMP FILE DUMPER

4.1. Ddump File Dumper

Dump or convert all or parts of object files and archive files.

Syntax

```
ddump [command] [modifiers] file, ...
```

Description

An object file consists of several different parts which can be individually dumped or converted with the **ddump** command.

ddump accepts both object files and archive files; in the latter case, each file in the archive is processed by the **ddump** command. **ddump** can generate debugging information only for code that is fully bound at link time; it does not work on relocatable object files.

For COFF **ddump** can also be used to convert from COFF or ELF to other object formats. See the **-R** command for Motorola S-Records and the **-I** command for IEEE695 (COFF only).

command is composed of a hyphen (-) followed by one or more command letters. One or more optional modifier letters for some commands may either be concatenated to the command letter, or may be given as separate option arguments. Commands and options are all represented by unique letters and so may be mixed in any order. Typically modifiers consisting of a single letter are concatenated with commands, while modifiers taking a separate argument are given as separate options (e.g., **-Rv** versus **-R -o name**).

For COFF, **ddump** permits combining options that produce output files. For example:

```
ddump -RI a.out
```

will produce both a Motorola S-Record output file, **srec.out**, and an IEEE695 output file, **ieee.out** from the COFF input file **a.out**. When producing multiple output files in this way, do not specify an output filename with the **-o** option as both files would be written to the same output file.

ddump now supports dumping ELF64 binaries. In previous versions it only accepted ELF32 binaries. The user interface remains the same as when dumping ELF32 binaries.

See also [Utilities on page 1](#), *Common Command-Line Options*.

ddump commands

-A

An action to dump assembly symbol table.

The following modifier is available:

-O plain| xml

Specify output format in either plain text or XML. (plain is the default.)

-a

Dump the archive header for all the files in an archive file.

-B

Convert a hexadecimal file to binary format. Each pair of hexadecimal numbers is translated to one byte in the output file. Whitespace (spaces, tabs, and newlines) are ignored. Unless the -o modifier is used, the output file will be named **bin.out**.

-C

Generate an S-Record difference file from two ELF executable files. Usage:

```
ddump -C [modifiers]file1file2
```

The following special modifiers are available:

-h

Generate differences for read-only sections and a complete dump for writable sections. Useful when the original executable has already run on the target and has modified some writable information.

-v

Generate differences for initialized sections. Useful when the executable has initialized uninitialized data.

-c

Dump the string table in each object file.

-D

Dump the DWARF debugging information in each object file.

-d, +d

Command modifier. See [Table 1 on page 16](#).

-F

Demangle C++ names entered interactively, one per line (no files are processed). Enter **Ctrl-C** or the end-of-file character to terminate interactive mode. If combined with other options, prints demangled names.

For details on how names are mangled, see the *C++ Features and Compatibility* chapter of the *Wind River Diab Compiler User's Guide* for your architecture.

-f

Dump the file header in each object file.

-g

Dump the symbols in the global symbol table in each archive file.

-H

Display the contents of any file in hexadecimal and ASCII formats. The **-p** modifier will display hexadecimal only.

-h

Dump the section headers in each object file.


The following modifier is available

-O plain| xml


Specify output format in either plain text or XML. (plain is the default.)

-I

For COFF (not available for ELF files). Convert a COFF file to IEEE-695 object format. The output file will be named **ieee.out** unless the **-o** modifier is used. The **-pname** modifier can be used to specify the processor name.

 **Note:** The linker option **-Xextern-in-place** is required when creating an executable file to be converted to IEEE-695 format. For more information about **-Xextern-in-place**, see the *Wind River Diab Compiler Options Reference*.

ddump appends an extra underscore in front of symbols in the IEEE-695 file when there is no **main** function in the COFF executable.

 **Note:** If commands in the linker command file cause two or more input sections from the same module to be combined into a single output section, **ddump -I** will produce the following warning:

This is because COFF debug format includes section vs. file information, but cannot accommodate the case of more than one input section from a single file in an output section. This warning indicate that such information was lost (which may affect some debuggers), and may be otherwise ignored.

The startup module, **crt0**, includes multiple code sections, and so will produce this warning.

-l

Dump the line number information in each object file.

-m

Generate a metadata report for all provided object files, as the linker would do when linking the object files. Usage:

```
ddump -m
metadata-report-output-spec
```

For information about *metadata-report-output-spec* see the **-Xdump-metadata** entry in the *Wind River Diab Compiler Options Reference*

-N

Dump the symbol table information in each object file. Similar to the UNIX **nm** command. The following special modifiers are available. See also the **-t** option below for a more readable dump but without further options.

-e

Export symbol table as a colon separated list.

-g

Emulate GNU **nm** output.

-h

Suppress header.

-o

Display numbers in octal.

-p

Display symbols in BSD format.

-r

Display filename before symbol name.

-u


Display only undefined symbols.

-x

Display numbers in hexadecimal.

-nCommand modifier. See [Table 1 on page 16](#).**-o**

Dump the optional header in each object file.

 **Note:** **-o** is both a command and an option. If any of the commands **-B**, **-I**, **-R**, **-T**, **-Td**, or **-X** are encountered, then a following **-o** is assumed to specify the output file for the command. If **-o** is encountered first, then it operates as the command. See the **-o** modifier in [Table 1 on page 16](#).

-P

Invoke the Symbol Redirection and Section Renaming utility. This utility offers the ability to

- redirect references to global symbols within the scope of functions or C structures, and/or
- rename sections

for entire object files or archive files.

Usage:

```
ddump -P [modifiers] command-file input-file output-file
```

ddump -P takes an input file (either an object file or an archive file), processes it according to the commands listed in *command-file*, and sends the processed output to the named *output-file*.

Optional *modifiers* include::**-e**

Promote all warnings to errors.

-uPrefix all the symbol names referenced in the command file with an underscore (**_**).

-w

Demote all non-critical errors to warnings.

Commands in the command file are separated by a semicolon (;). However, if no more than one command is present on a given line, semicolon termination is optional.

Comments are allowed in the command file. A # token at any point on a line marks the beginning of a single-line comment, effective until the end of the line. Inline and multi-line comments are also possible, using the C-like construct `/* */`.

The command file supports two kinds of command:


Symbol Redirection

Symbol-redirection commands have the following syntax:

```
scope : original-variable = new-variable
```

where *scope* denotes the scope (function or **struct** instance) within which the redirection takes place, while *new-variable* is the symbol to which *original-variable* is redirected. If *scope* is not a valid symbol within the input ELF object, an error is emitted and no redirection is performed on either one.

If *new-variable* exists, references pointing to *original-variable* within the given *scope* are redirected to *new-variable*. Otherwise, a new undefined symbol with the name of *new-variable* is created and the redirection is performed against it. A warning is emitted if the latter instance.

 **Note:** Symbol redirection only works if references to the corresponding symbol are "explicit," in the form of "relocations" that occur in the given scope. Some cases in which this may not be so are:

- When `-Xlocal-data-area` is used (note that this option is enabled by default on some targets). This option causes references to some global variables to be replaced by references to the Local Data Area (LDA).
- On ARM the address of a global variable may be loaded via a PC-relative load from a "literal pool." However `ddump -P` does not know which literal pools are associated with which functions; for this reason symbol redirection within functions does not work on ARM, though symbol redirection within structures does work.

Section Renaming

Section-renaming commands have the following syntax:

```
section-name -> new-section-name
```

This command renames the section currently known as *section-name* to *new-section-name*. If multiple sections with the same name exist within the object file, this directive affects all of them.

-p [name]

Command modifier. See [Table 1 on page 16](#).

-R

Convert an executable or object file to different formats, especially Motorola S-Record format. The output file will be named **srec.out** unless the `-o` modifier is used (see [Table 1 on page 16](#)). Sections may be selected with the `-n` or `-d` and `+d` modifiers as usual.

The following special modifiers are available:

- mt

Write S-Records of the given type: 1 for 16-bit addresses, 2 for 24 bit-addresses, 3 for 32-bit addresses (the default). No space is permitted between "**m**" and *t*.

- p

Write a plain ASCII file in hexadecimal (not S-Record format).

- u

Write a binary file (not S-Record format). Inter-section gaps of size less than or equal to 10KB are filled with 0. The size may be changed with the **-y** option described in [Table 1 on page 16](#). A larger gap will cause an error.

The binary contains just the raw data from the original files. **ddump** does not perform any translation or relocation.

- v

Do not output the **.bss** or **.sbss** section (applies to all output formats).

Without **-v**, S-Records will be generated to set **.bss** and **.sbss** sections to 0. This will increase transmission or programming time when sending S-Records to PROM programmers or other devices and may not be desirable.

- wn

Set the line width of the S-records to represent *n* data characters. The actual line length is $2n$ plus the size of other fields such as the address field. The default value of *n* is 20. $2n$ is used instead of *n* because it takes $2n$ hex digits to represent *n* characters. No space is permitted between "**w**" and *n*.

-r


Dump the relocation information in each object file.

-S

Display the size of the sections. Similar to the UNIX **size** command. By using the **-f** modifier, the section names will be included in the output. By default, only the **.text**, **.data**, and **.bss** sections will be included. By using the **-v** modifier, all sections will be included.

-s

Dump the section contents in each object file.

 **Note:** Use of the **v** modifier, that is, **-sv**, is highly recommended.

-T

Remove symbol table information from each object file (similar to the UNIX **strip** command). Also remove relocation and debug information from ELF objects. If the ELF object is a relocatable object (ELF type **ET_REL**), do not strip relocation information. If the object is an ELF shared object (ELF type **ET_DYN**), only strip the non-dynamic relocation information.

-Td

Similar to **-T**, but remove only debug information (ELF only).

-t

Dump the symbol table information in each object file.

For COFF objects only, the **-v2** option will suppress debug information in the dump.

-t index

Dump the symbol table information for the symbol indexed by *index* in the symbol table.

+t *index*

Dump the symbol table information for the symbols in the range given by the -t option through the +t option. If no -t was given, 0 is used as the lower limit.

The following modifier is available:

-O *plain| xml*

Specify output format in either plain text or XML. (plain is the default.)

-U

An action to dump high level language symbol table.

The following modifiers are available.

-O *plain| xml*

Specify output format in either plain text or XML. (plain is the default.)

--expand-array-elem **NUMBER**

Allow user to control how many array elements should be expanded in the XML output. The default value of NUMBER is 1, that means only the first element of the struct/union array is dumped. --expand-array-elem -1 will expand all array elements.

-v

When used together with -O xml, dump struct/union members recursively.

-u

Command modifier. See [Table 1 on page 16](#).

-V

Print the version number of **ddump**.

-v

Command modifier. See [Table 1 on page 16](#).

-X

Convert input file to Intel hex format. This command accepts the -m and -w modifiers described under the -R command, as well as the -o modifier described in [Table 1 on page 16](#).

-y

Command modifier. See [Table 1 on page 16](#).

-z *name*

Dump the line number information for the function *name*.

-z *name number*

Dump the line number information in the range *number* to *number2* given by **+z** for the function *name*.

+z *number2*

Provide the upper limit for the -z option.

ddump Command Modifiers

Table 1 on page 16 displays the modifiers that work with multiple ddump commands.

Table 1. ddump Command Modifiers

Command Modifier	Use with Command(s)	Description
-d <i>number</i>	-h -l -R -s	Dump information for sections greater than or equal to <i>number</i> . Sections are numbered 1, 2, etc.
+d <i>number</i>	-h -l -R -R -s	Dump information for sections less than or equal to <i>number</i> .
-n <i>namelist</i>	-h -l -R -s -t	Dump the information associated with each section name in a comma-separated list of section names.
-o <i>name</i>	-B -l -R -T -Td -X	Specify an output filename for the respective commands. (See the -o command.)
-p	any but -l	Suppress printing of headers. Special meaning with -R .
-p	-l only	Set the processor name in the "Module Begin" record. If this option is not specified the processor name is taken from the magic number of the input file. A list of processor names and magic numbers can be found in the IEEE 695 specification.
-u	any	Underline filenames. Special meaning with -N , -P and -R .
-v	any	Dump information in verbose mode. Special meaning with -R .
-y <i>n</i> [, <i>c</i>]	-Ru	<p>Change the size of the gap allowed by the -Ru command to <i>n</i> and, optionally, fill it with the character <i>c</i>. (See the -R command.) For example:</p> <pre>ddump -Ru -y20000,0x20 ...</pre> <p>permits gaps from 1 through 20,000 bytes, and fills those gaps with the space character. By default, <i>n</i> is 10k and <i>c</i> is 0.</p>

ddump File Dumper Examples on page 16

Refer to the following examples of the ddump command.

4.1.1. ddump File Dumper Examples

Refer to the following examples of the ddump command.

Dump File Header and Symbol Table for Files in Archive

Dump the file header and symbol table from each object file in an archive in verbose mode:

```
ddump -ftv lib.a
```

Convert Executable File to Motorola S-Records

Convert an executable file named **test.out** to Motorola S-Record format, naming the output file **test.rom**. Use the **-v** option to suppress the **.bss** section (without **-v**, S-Records would be generated to fill the **.bss** section with zeros).

```
ddump -Rv -o test.rom test.out
```

Generate S-Records Only for "data" Sections

Similar as the prior example.

For architectures that support small data sections, but convert and output only sections **.data** and **.sdata** and call the result **data.rom**.

```
$ ddump -R -n .data,.sdata -o data.rom test.out
```

For others, convert and output only section **.data** and call the result **data.rom**.

```
$ ddump -R -n .data,-o data.rom test.out
```

Display Section Sizes

Use **-Sf** to show the size of all sections loaded on the target. See below:

```
$ ddump -Sf a.out
          9056(.text+.sdata2) + 772(.data+.sdata) + 428(.sbss+.bss) =
          10256
```

Demangle C++ Names

Demangle C++ names with **ddump -F**:

```
$ ddump -F
command entry mymain__FiPPc          user entry
mymain(int , char **)          demangled result init__7myclassFv          user entr
y
myclass::init(void )          demangled result
```

Display Metadata Information

```
$ ddump
-mx,object=objectFile,source=sourceFile,ctoaVersion,dasVersion bubble.o
crt0.o
<?xml version="1.0"?> <DiabMetadata>
  <File>      <object>objects/bubble.o</object>
               <source>../src/bubble.c</source>
               <ctoaVersion>5.9.2.0</ctoaVersion>
               <dasVersion>5.9.2.0</dasVersion> </File> <File>
               <object>objects/crt0.o</object>
               <source>../src/crt0.s</source>
               <ctoaVersion></ctoaVersion>
```

```

    <dasVersion>5.9.2.0</dasVersion> </File>
  </DiabMetadata>

```

Dump high level language symbol table as XML, with struct member hierarchy.

```

$ ddump -U -v -O xml a.out
    <HighLevelSymbolTable> <File name="a.out">
      <HLLSymbol name="var" type="struct foo" filename="test.c" aggregate="1"
      addr="0x100102b8" size="16" section=".bss"> <HLLSymbol name="a"
      type="unsigned char" filename="test.c" aggregate="0" addr="0x100102b8" size="1
"
      section=".bss"> </HLLSymbol> <HLLSymbol name="b" type="struct bar"
      filename="test.c" aggregate="1" addr="0x100102bc" size="8" section=".bss">
      <HLLSymbol name="a" type="unsigned char" filename="test.c" aggregate="0"
      addr="0x100102bc" size="1" section=".bss"> </HLLSymbol> <HLLSymbol
      name="b" type="short" filename="test.c" aggregate="0" addr="0x100102be" size="
2"
      section=".bss"> </HLLSymbol> <HLLSymbol name="c" type="int"
      filename="test.c" aggregate="0" addr="0x100102c0" size="4" section=".bss">
      </HLLSymbol> </HLLSymbol> <HLLSymbol name="c" type="int"
      filename="test.c" aggregate="0" addr="0x100102c4" size="4" section=".bss">
      </HLLSymbol> </HLLSymbol> </File>
    </HighLevelSymbolTable>

```

Dump assembly symbol table as XML

```

$ ddump -A -O xml a.out
    <AsmSymbolTable> <File name="a.out">
      <AsmSymbol name="main" type="function" filename="test.s" addr="0x100bc"
      size="24" section=".text" /> <AsmSymbol name="var" type="object"
      filename="test.s" addr="0x202a8" size="16" section=".bss" /> </File>
    </AsmSymbolTable>

```

-U Display either- or a combination of the following symbol types and attributes:

var func

const

undef undef

var

Show variables, including those with const attribute;

Examples:

```

ddump -Uvar dummy.a
ddump -Uvar,-const dummy.a #exclude constants, which are enabled by default
ddump -Uvar,+undef dummy.a #show variables, including undefined ones

```

const

Show constants only.

func

Show functions.

undef

Show undefined symbols only.

By default, it will show both undefined functions and variables; combine with symbol type specifier to limit the results to a specific symbol type, as follows:

```
ddump -Uvar,undef dummy.a #show only undefined variables
ddump -Ufunc,undef dummy.a #show only undefined functions
```

Parent topic: [ddump File Dumper on page 9](#)

5. DMAKE MAKEFILE UTILITY

5.1. Dmake Makefile Utility

Rebuilding the Wind River libraries requires the special make utility, **dmake**, by Dennis Vadura. **dmake** is shipped and installed automatically with the tools.

dmake supports the standard set of basic rules and features supported by most “make” utilities—see the documentation for other “make” utilities for details.

Installation

The **dmake** executable is shipped in the **bin** directory and requires no special installation.

Using dmake

Use **dmake** as a typical “make” utility. For example, enter **dmake** without parameters to cause it to look for a makefile named, on Windows, **makefile** (case-insensitive), and on UNIX, first **makefile** and then **Makefile**.

Enter **dmake -h** for a list of command-line options.

dmake requires a “startup” file unless the **-r** option is given on the command line, and will look for the file in the following locations in order:


- The value of the macro **MAKESTARTUP** if defined on the command line.
- The value of the **MAKESTARTUP** environment variable if defined.
- The file *versionDir/dmake/startup.mk* (supplied as shipped).

For more information about **dmake**, see the **dmake** command reference, which is provided with the rest of the Diab documentation set.

6. WINDISS SIMULATOR AND DISASSEMBLER

6.1. Windiss Simulator and Disassembler

WindISS, the Wind River Instruction Set Simulator, is a simulator for executables and a disassembler for object files and executables. The disassembler mode provides both batch and interactive disassembly. WindISS does not support multi-thread simulation.

 **Note:** WindISS is not supported for ARM Thumb-2, ColdFire hardware floating point, or x86 instructions.

Synopsis

The three modes of operation are selected by:

`windiss ...`

Simulation (with no `-i` option).

`windiss -i ...`

Batch disassembly.

`windiss -ir ...`

Interactive disassembly.

The modes of operation are described the following sections.

 **Note:** Windiss does not support disassembling or simulating PPC64 binaries.

[Simulator Mode on page](#)

In simulator mode, **windiss** can take command-line arguments, input from standard input, and send output to standard output.

[Batch Disassembler Mode on page 21](#)

In Batch Disassembler mode `windiss` can disassemble ELF object files and executables and write the assembly code to standard output.

[Interactive Disassembler Mode on page 22](#)

In Interactive Disassembler mode, `windiss` can print the disassembled ELF object code and executables interactively.

[Windiss Simulator and Disassembler Examples on page 23](#)

Refer to the following examples of the `windiss` command.

6.1.1. Batch Disassembler Mode

In Batch Disassembler mode `windiss` can disassemble ELF object files and executables and write the assembly code to standard output.

Syntax (Disassembler Mode)

```
windiss -i[o | e | l | A mask] [label] [-v target-name] [-1 start-address [-R2 end-address]]
[-R3 section] filename
```


Note the following:

- **A***mask* is for only for PowerPC.
- *label* is used only with the **I** modifier.
- **-v***target-name* is only for PowerPC and MIPS.
- For the **-ir** option, see [Interactive Disassembler Mode on page 22](#).

Description

Batch disassembly mode is selected by the **-i** option with no “**r**” modifier. In batch disassembler mode, **windiss** disassembles ELF object files and executables and writes the assembly code to standard output. The **-i** stands for instructions.

The modifiers **o**, **e**, and **I** are appended to the **-i** without an additional hyphen and with no spaces allowed. Modifiers may be used together in any order.

 **Note:** For ColdFire, by default the compiler prepends function names with an underscore (“_”).

To disassemble code use:

- **-i** alone to disassemble the whole file.
- [**e**] **-R1***start-address* [**-R2***end-address*] to specify code addresses. Use 0x for hex numbers. If part of a function is specified by a **-R1** and **-R2** options, the entire function is disassembled unless the “**e**” option is used to request exact addresses. A space is required between either the **-R1** or **-R2** option and the address.
- **-R3***section* to specify a section index in the object file. If the specified section has zero length, the option is ignored.
- **o** to also show machine code.
- **I***label* to specify the name of a function to be disassembled.
- **-iA***mask* (PowerPC only) specifies target-specific disassembler options. Currently the only supported mask is 0x1. Specifying **-iA** 0x1 tells the disassembler to display simplified instructions where possible; for example, **ori r0,r0,r0** would be displayed as **nop**.
- **-v***target-name* (MIPS and PowerPC only) to specify variant-specific processor behavior. See [Simulator Mode on page](#) , [Simulator Mode Command-Line Options](#) for details.
- **-v***target-name* (MIPS only) to specify variant-specific processor behavior. If **-v** is not present, the default MIPS target (generic MIPS instruction set) is simulated. Options include **MIPS** (same as default), **R3000**, **R4000**, and **MIPS64**. (**-vMIPS64** is the same as **-tMIPS64**.)

Parent topic: [Windiss Simulator and Disassembler on page 21](#)

6.1.2. Interactive Disassembler Mode

In Interactive Disassembler mode, **windiss** can print the disassembled ELF object code and executables interactively.

Syntax (Interactive Disassembler Mode)

```
windiss -ir[o] filename
```

Description

In interactive disassembler mode, **windiss** prints the disassembled ELF object code and executables interactively. The **-i** stands for instructions; the **r** modifier selects interactive mode; the **o** modifier shows hex machine code in addition to assembly language.


To disassemble code in interactive mode:

```
d[isasm]label | [-e]start-address [end-address]
```

If part of a function is specified, the entire function will be disassembled unless the -e option is given. The -e option requests that exact addresses be disassembled, without other code.

To quit interactive mode:

```
q[uit]
```

 **Note:** For ColdFire, by default the compiler prepends function names with an underscore ("_").

Parent topic: [Windiss Simulator and Disassembler on page 21](#)

6.1.3. Windiss Simulator and Disassembler Examples

Refer to the following examples of the windiss command.

Simulate Using All Defaults

Run **windiss** in simulator mode. The program output is 17.

```
windiss a.out

17
windiss: task finished, exit code: 83521, Instructions executed: 2118
windiss: interrupts were never enabled
```

Simulate with Specified Memory Sizes

Run **windiss** in simulator mode, specifying memory size as 20,000 bytes, and then 1 megabyte:

```
windiss -m 20000 a.out

windiss: loading outside of memory, EA=0x4c00 (increase by using -m <size>)

windiss -m 1M a.out

17
windiss: task finished, exit code: 83521, Instructions executed: 2118
windiss: interrupts were never enabled
```

Simulate Showing POSIX Calls

Run **windiss** in simulator mode, and use the debug option with a mask to show POSIX calls.

```
windiss -d 8 a.out
%% posix call 120: isatty(1), ret=1, errno=0
%% posix call 4: write(1, 0x6bfc, 4)
```

```
17
windiss: task finished, exit code: 83521, Instructions executed: 2118
windiss: interrupts were never enabled
```

Batch Disassemble Entire File

Disassemble **a.out**:

```
windiss -i a.out
```

Batch Disassemble One Function in File

Disassemble **main** in **a.out**:

```
windiss -il main a.out
```

Batch Disassemble Functions in Address Range

Disassemble all code in function which includes addressees from 0x9c to 0x4e:

```
windiss -i -R1 0x9c -R2 0x4e a.out
```

Disassemble only code from 0x9c to 0x4e:

```
windiss -ie -R1 0x9c -R2 0x4e a.out
```

Interactive Disassembly

Disassemble **a.out** in interactive mode, examine **main** and addresses 0xa0 to 0xa4 (or for TriCore, 0xa0000000 to 0xa0000004) using the following commands:

```
windiss -ir a.out
d main
```

Which produces the following output:

MIPS

```
0x0000009c: <main>:      addi      $29,$29,-24
```

```
0x000000a0: <main+4>:    addi      $2,$0,10
```

```
0x000000a4: <main+8>:    addi      $29,$29,24
```

```
0x000000a8: <main+12>: jr      $31
```

```
0x000000ac: <main+16>: sll      $0,$0,0
```

PowerPC

```
00000070: <main>:      stwu     r1,-8(r1)
```

```
00000074: <main+4>:    mfspr     r0,8
```

```
00000078: <main+8>:    stw      r0,12(r1)
```

```
0000007c: <main+12>:   addi     r3,r0,10
```

```
00000080: <main+16>:   lwz      r0,12(r1)
```

```
00000084: <main+20>:   mtspr     8,r0
```

```
00000088: <main+24>:   addi     r1,r1,8
```

```
0000008c: <main+28>:   bclr     20,0
```

RH850

```
001000bc <_main>:    0x00562a00   addi     42,r0,r10
```

```
001000c0 <_main+4>: 0x7f0      jmp      [r31]
```

TriCore

```
a0000124 <main>:      sub.a     sp, 8
```

```
a0000126 <main+2>:    mov      d14,130
```

```
a000012a <main+6>:    mov      d13,132
```

```
a000012e <main+10>:  extr.u    d15,d14,0,8
```

```
a0000132 <main+14>:  st.w      [sp]0,d15
```

```
a0000134 <main+16>:  movh.a   a15,40960
```

```
a0000138 <main+20>:  lea       a4,[a15]15348
```

```
a000013c <main+24>:  call     printf
```

```
a0000140 <main+28>:  extr.u    d15,d13,0,8
```

```
a0000144 <main+32>:  st.w      [sp]0,d15
```

```
a0000146 <main+34>:  movh.a   a15,40960
```

```
a000014a <main+38>:  lea       a4,[a15]15360
```

```
a000014e <main+42>:  call     printf
```

```
a0000152 <main+46>:  mov       d2,0
```

```
a0000154 <main+48>:  ret
```

```
a0000156 <main+50>:  nop
```

For the exact address range for TriCore use:

```
d -e 0xa0000000 0xa0000004
```

For the exact address range for other architectures use:

```
d -e 0xa0 0xa4
```

Which produces the following output:

MIPS

```
0x000000a0: <main+4>:   addi      $2,$0,10
```

PowerPC

```
000000a0: <exit+16>:   stw      r0,20(r1)
```

TriCore

```
a0000000 <_start>:   mov      d14,d4
```

```
a0000002 <_start+2>: mov.aa    a13,a4
```

To quit use the **q** command.

Simulate, Showing Instructions and Data as Executed

These examples illustrates use for MIPS, PowerPC, and RH850. In each case, **windiss** is run in simulator mode until the specified number of instructions are executed.

MIPS

For MIPS, run normally until 2293 instructions are executed.

```
windiss -Ds 2293 -q a.out
```

```
17
SR   00000000 EPC  00000000 $0   00000000 $1   00000000
$2   ffffffff $3   00000064 $4   00000003 $5   00006bfc
$6   00000004 $7   00000000 $8   00000000 $9   00000000
$10  00000000 $11  00006be0 $12  00006788 $13  00000001
$14  00006be0 $15  00000002 $16  00000000 $17  00000000
$18  00000000 $19  00000000 $20  001ffe34 $21  00000001
$22  00014641 $23  0000eb00 $24  ffffffff $25  00000000
$26  00000000 $27  00000000 $28  0000eb04 $29  001ffdc0
$30  0000099c $31  00000208
0x00000208: <exit+148>:      addu      $4,$22,$0
$4   00014641
0x0000020c: <exit+152>:      jal      <__exit>
$31  00000214
0x00000210: <exit+156>:      sll      $0,$0,0
0x000002fc: <__exit>:          j          <__trap>
0x00000300: <__exit+4>:          ori      $2,$0,1
$2   00000001
0x000002dc: <__trap>:          syscall
```

PowerPC

For PowerPC, run normally until 1944 instructions are executed.

```
windiss -Ds 1944 -q a.out
```

```
17
MSR 00000000 CTR 00002818 LR 0000018c SRR0 00000000
SRR1 00000000 R0 0000018c R1 001ffdf0 R2 0000d314
R3 ffffffff R4 00000000 R5 00000004 R6 00001678
R7 00000064 R8 00000000 R9 000053f8 R10 00000008
R11 00000006 R12 00000000 R13 0000d31c R14 00000000
R15 00000000 R16 00000000 R17 00000000 R18 00000000
R19 00000000 R20 00000000 R21 00000000 R22 00000000
R23 00000000 R24 00000000 R25 00000000 R26 00000000
R27 00000000 R28 001ffe34 R29 00000001 R30 00014641
R31 00000968
0x00000a4c: <_cleanup+228>:      bclr      20,0
0x0000018c: <_exit+116>:      addi      r3,r30,0
R3 00014641
0x00000190: <_exit+120>:      b          <__exit>
0x00000394: <__exit>:      addi      r0,r0,1
R0 00000001
0x00000398: <__exit+4>:      b          <__trap>
0x0000037c: <__trap>:      tw          0,r31,r31
```

RH850

For RH850, run normally until 8067 instructions are executed. This example uses canonical “hello world” code.

```
windiss -Ds 8067 -q a.out hello world
----- trace on (skip), instruction: 8060
PSW 00000021 EIPC 00000000 EIPSW 00000000 ECR 00000000
FEPC 00000000 FEPSW 00000000 FPSR 00000000
R0 00000000
R1 00000004 R2 00000000 R3 007ff8d8
R4 0010eb2c
R5 0010eaf6 R6 00000003 R7 00106c1c
R8 0000000c
R9 00106bb0 R10 ffffffff R11 00000000
R12 00000000
R13 00000000 R14 00000000 R15 00000000
R16 00000000
R17 0000000c R18 00000054 R19 00000000
R20 00000000
R21 00000000 R22 00000000 R23 00000000
R24 00000000
R25 00000000 R26 00000000 R27 007fff68
R28 007ff948
R29 00000000 R30 00106b3c R31 00100106
VR0 0000000000000000
VR1 0000000000000000 VR2 0000000000000000
VR3 0000000000000000 VR4 0000000000000000
VR5 0000000000000000 VR6 0000000000000000
VR7 0000000000000000 VR8 0000000000000000
VR9 0000000000000000 VR10 0000000000000000
VR11 0000000000000000 VR12 0000000000000000
VR13 0000000000000000 VR14 0000000000000000
VR15 0000000000000000 VR16 0000000000000000
```



```

VR17 0000000000000000 VR18 0000000000000000
VR19 0000000000000000 VR20 0000000000000000
VR21 0000000000000000 VR22 0000000000000000
VR23 0000000000000000 VR24 0000000000000000
VR25 0000000000000000 VR26 0000000000000000
VR27 0000000000000000 VR28 0000000000000000
VR29 0000000000000000 VR30 0000000000000000
VR31 0000000000000000
00101c92 <__cleanup+186>:      addi
    32,r3,r3
    PSW 00000020 R3 007ff8f8
00101c96 <__cleanup+190>:      jmp
    [r31]
00100106 <_exit+42>:          mov
    r29,r6
    R6 00000000
00100108 <_exit+44>:          jarl
    284,r31
    R31 0010010c
00100224 <____exit>:          movea
    1,r0,r1
    R1 00000001
00100228 <____exit+4>:        br
    -16
00100218 <__trap>:           trap
    31

```

Parent topic: [Windiss Simulator and Disassembler on page 21](#)