Quick Reference for shmdefine

0898010-060
July 2018

The information contained in this document is believed to be correct at the time of publication. It is subject to change without notice. Concurrent Real-Time makes no warranties, expressed or implied, concerning the information contained in this document.

To report an error or comment on a specific portion of the manual, photocopy the page in question and mark the correction or comment on the copy. Mail the copy (and any additional comments) to Concurrent Real-Time, 2881 Gateway Drive, Pompano Beach, FL 33069-4324. Mark the envelope **"Attention: Publications Department**." This publication may not be reproduced for any other reason in any form without written permission of the publisher.

Preface

Scope of Manual

This guide is designed to assist you in getting started with use of the **shmdefine** utility.

Structure of Manual

This manual consists of one chapter.

- *Chapter 1* introduces you to the **shmdefine** utility, describes the syntax of the command as well as its options, and guides the user through an example demonstrating the steps necessary for programs to share data which targets a Linux system.

Syntax Notation

The following notation is used throughout this manual:

italic

Books, reference cards, and items that the user must specify appear in *italic* type. Special terms may also appear in *italics*.

list bold

User input appears in **list bold** type and must be entered exactly as shown. Names of directories, files, commands, options and system manual page references also appear in **list bold** type.

`list`

Operating system and program output such as prompts and messages and listings of files and programs appear in `list` type.

`[ ]`

Brackets enclose command options and arguments that are optional. You do not type the brackets if you choose to specify such options or arguments

Referenced Publications

The following publications are referenced in this document:

0890240	*hf77 Fortran Reference Manual*
0890497	*Concurrent C/C++ Reference Manual*
0890516	*MAXAda Reference Manual*

Contents

Chapter 1 Using shmdefine

Tables

1
Using shmdefine

The **shmdefine** utility is designed to facilitate the use of shared memory by a set of cooperating programs. Although you may have a number of programs that will cooperate in using one or more shared memory segments, it is necessary to invoke the utility only once. Because **shmdefine** produces object files that must be linked to the source object file, you must invoke it prior to linking.

shmdefine works with MAXAda and Concurrent Fortran as well as the GNU C, C++, and Fortran compilers for programs that will execute on RedHawk™ Linux® target systems.

The initialization file generated by **shmdefine** contains an executable function to access the shared memory services at program start-up time. On target systems, the function must be explicitly called before any shared memory regions are referenced.

The linker command script generated by **shmdefine** describes the shared memory regions to the linker.

Command Syntax

The format for executing the **shmdefine** utility is as follows:

shmdefine [**-CGHNZ**] [**-b** *base_name*] [**-t** *target*] [*files*]

The options are as follows:

-C
--case-sensitive

Pay attention to case when interpreting symbols in the input. **shmdefine** interprets symbols in a case-insensitive manner by default.

-G
--gnu-f77

Use GNU Fortran 77 naming conventions for common blocks. By default, **shmdefine** uses Concurrent Fortran naming conventions. If you are using GNU Fortran 77 to compile common block definitions, you should use this option to ensure that your common block data is shared properly.

-H
--help

Display the help screen and stop.

-N
--no-copy

Suppress calls to data initialization subprograms specified in the `INIT USING` clause in any **`shmdefine`** input file (see "Initialization" on page 1-5).

-Z
--allow-nubbins

Allow regions with a size <= 0.

-b *base_name*
--base *base_name*

Use *base_name* as the prefix of the generated output files. **`shmdefine`** will append **`.sm.c`** and **`.sm.ld`** to generate the output files. The default prefix is **`shm_init`**.

-t *os*
--target *os*

Generate output files suitable for the target operating system *os*. Valid values for *os* **`linux`**.

If no *files* are specified, **`shmdefine`** will read standard input. Otherwise, each file is parsed, in order, until all have been read. When all input has been consumed, **`shmdefine`** will generate a C source file and a matching linker command script describing the defined shared memory region(s).

The C source file generated for a target contains an executable function that accesses shared memory services at the time the user calls it. If an `INIT USING` clause in the **`shmdefine`** input file specifies a callback subprogram, the C source file also makes the call to the specified subprogram. Calls to subprograms specified in `INIT USING` clauses occur when a newly executed program attaches to the corresponding shared memory segment – if and only if it is the only program currently attached to that segment. Linux C, C++ and Fortran programs which desire access to shared memory regions defined using **`shmdefine`** must call `shm_init()` to initialize the shared memory before using it, and must call `shm_rm()` when finished to release and/or destroy the shared memory. The Ada compiler automatically calls `shm_init()` and `shm_rm()`.

WARNING

Targets do not provide automatic data initialization of the shared memory segment even if the original source code specified initial values for the variables associated with the shared memory segment. See "Initialization" on page 1-5 for information on initializing shared regions.

The default name for the generated C source file is **`shm_init.sm.c`**.

The following table describes how to call the `shm_init()` and `shm_rm()` subprograms from C, C++ and Fortran.

Language	Initialize / Destroy

C	`extern void shm_init(void);`
	`extern void shm_rm (void);`
	
	`shm_init();`
	`shm_rm();`

Fortran	`call shm_init` `call shm_rm`

The linker command script describes the shared memory segments to the linker. The default name for this file is **`shm_init.sm.ld`**.

NOTE

The **`-M`** option is *not* needed when linking the **`.ld`** link script. If the **`-M`** option is used on the compiler/linker command on targets, a lengthy memory map is produced.

For C, C++, and Fortran programs, you must use the "**`-ld:1200 shmdef.ld`**" option to pass the filename to **`a.link`**.

You can also place the string "`-ld:1200 shmdef.ld`" in a `linker_options` pragma, embedded in your source code. The value "`1200`" passed to the **`-ld`** option places the linker command script at an appropriate position in the **`ld(1)`** command line to ensure that it is evaluated properly.

When linking your programs using MAXAda, you must use the "**`-ld`**[**`:`***nnnn*]" option to pass the filename to **`a.link`**.

For an active partition named **`myprog`**, this can be done easily by issuing:

```
a.partition -oappend '-ld:1900 shm_init.sm.ld' myprog
```

You can also place the string "`-ld:1900 shm_init.sm.ld`" in a `LINKER_OPTIONS` pragma, embedded in your Ada source code. The value "`1900`" passed to the **`-ld`** option places the linker command script at an appropriate position in the **`ld(1)`** command line to ensure that it is evaluated properly.

NOTE

The **`-ld`**[**`:`***nnnn*] argument syntax requires MAXAda 3.4-004 or greater.

Shared Regions

Input to the **shmdefine** utility defines the shared memory segment or segments that are to be used by cooperating programs. You may define the segments using standard input, or you may specify one or more files that contain the definitions. Although input in either case may be free-form, the general format for defining a shared memory segment is as follows:

```
[TARGET os]
[FORTRAN COMPILER compiler]
SHARED REGION region_name
    [INIT USING subprogram_name]
    [attribute1, attribute2, ...]
    variable_clause1, variable_clause2, ...
END SHARED REGION
```

Note that blanks, tabs, and newlines are recognized only as separators. The hash character (#) can be used to indicate that the rest of the line is a comment.

Target

The optional TARGET clause specified in the shared memory definition (see "Shared Regions" on page 1-4) takes LINUX as a parameter.

If a TARGET clause is specified, it is checked for consistency with any command line '**-t** *os*' target option. If the specified targets do not match, the command line option will override the target specified in the TARGET clause and a warning will be issued.

If the TARGET clause is not specified, the native operating system is assumed.

Fortran Compiler

The optional FORTRAN COMPILER clause specified in the shared memory definition (see "Shared Regions" on page 1-4) takes GNU or CONCURRENT as a parameter.

If a FORTRAN COMPILER clause is specified, it is checked for consistency with any **-G**/**--gnu-f77** command line option (see "Command Syntax" on page 1-1). If the specified compilers do not match, the command line option will override the *compiler* specified in the FORTRAN COMPILER clause and a warning will be issued.

If a FORTRAN COMPILER clause is not specified, the Concurrent Fortran compiler's naming conventions are assumed for any common blocks specified in the shared region.

Initialization

An `INIT USING` clause, if specified, provides a way to modify the behavior of the `shm_init()` function for associated shared regions (see "Shared Regions" on page 1-4).

After attaching the shared memory region, callbacks are made to `INIT USING` subprograms if and only if the number of processes attached to the shared region is 1. This provides a means to perform more complicated initializations of data residing in a shared memory region.

NOTE

The function named in the `INIT USING` clause in the **`shmdefine`** config file must be callable using simple "C" linkage.

If the function is compiled with a C++ compiler, for example, it must be within an '`extern "C" {...}`' storage class specifier.

Three parameters are passed to the `INIT USING` callback subprogram, which is expected to have the following profile:

```
void init_callback (char * region_name,
                    void  * start_address,
                    int     length);
```

The *region_name* is the same name specified in the `SHARED REGION` clause in the **`shmdefine`** input file. The *start_address* and *length* define the space allocated to the shared region.

The shared region will be uninitialized at the time the `INIT USING` subprogram is called, even if initial values were supplied in the source files defining the variables that are associated with the shared region.

The **`-N`** option may be used to suppress the calling of subprograms specified in an `INIT USING` clause (see "Command Syntax" on page 1-1).

Attributes

Attributes that can be specified in the shared memory definition (see “Shared Regions” on page 1-4) are presented in Table 1-1.

Table 1-1. Attributes

Attribute	Purpose
`ADDRESS`	Enables you to specify a starting virtual address for the shared memory segment.
`IPC`	Enables you to set the control flags for the segment.
`SHM_LOCAL`	Enables you to set the NUMA policy for the shared memory segment to the anchored soft-local policy. A soft-local policy allows pages to be allocated from global memory when pages are not available for allocation from the local memory pool. This option has no effect on systems without local memory.
`SHM_HARD`	Enables you to set the NUMA policy for the shared memory segment to the anchored hard-local policy. A hard-local policy causes a process to wait for local memory pages to become available if the pages cannot be allocated from the local memory pool when needed. This option has no effect on systems without local memory.
`KEY`	Enables you to specify a user-chosen identifier for the segment.
`MODE`	Enables you to set the permissions that are associated with the segment.
`SHM_RDONLY`	Enables you to prevent a process from writing to the segment.

Variables

The following types of variables may be associated with the shared memory segment (see “Shared Regions” on page 1-4):

- C external variables

  External variables must be declared with the type qualifier `volatile` in the C source program.

- Ada variables

  Ada variables should be declared volatile via pragma `volatile` and must be exported via pragma `export` in the Ada source program.

- Fortran common blocks

  When using Concurrent Fortran, common blocks should be declared `VOLATILE` in the Fortran source program.

NOTE

GNU Fortran 77 does not support the `VOLATILE` keyword.

- Concurrent Fortran pointer blocks
- Concurrent Fortran datapool dictionaries

The volatile declaration informs the compiler that the values of the variables may be modified in a way that is unknown to the compiler.

Variables are associated with the shared memory segment using the following variable clauses:

- `C EXTERN` *external_name* `[ SIZE` *n* `[ * ` *m* `]]`

  The C external variable *external_name* is included in the current region.

- `Fortran COMMON` *common_block_name* `[ SIZE` *n* `[ * ` *m* `]]`

  The Fortran common block *common_block_name* is included in the current region.

- `Fortran BLANK COMMON [ SIZE` *n* `[ * ` *m* `]]`

  The Fortran unnamed ("blank") common block is included in the current region.

- `Fortran DATAPOOL` *datapool_name*`, "`*filename.o*`"`

  The variables from *datapool_name* defined in the object file *filename.o* are included in the current region. *filename.o* must be compiled from the Fortran datapool dictionary source file (e.g. *my_pool.o* created from **`cf77 -c -g my_pool.dp`**).

NOTE

Fortran datapools are only supported by Concurrent Fortran compilers; use of this clause with other compilers may not allow the program to link.

- Fortran SIZEOFBLOCK *name* [* *count*] [SIZE *n* [* *m*]]

NOTE

Fortran pointer blocks are only supported by Concurrent Fortran compilers; use of this clause with other compilers may not allow the program to link.

Reserves space the size of the Fortran pointer block *name* in the current region. The space is eight-byte aligned, the size is rounded up to an eight-byte multiple, and the start and end addresses of the space are marked with the names sblock__*name* and eblock__*name*. **get_sblock_addr(3F)** and **get_eblock_addr(3F)** return these addresses.

If *count* is specified, space is reserved in the current region for *count* contiguous copies of the Fortran pointer block *name*. Each copy is eight-byte aligned, and its size is rounded up to an eight-byte multiple. **get_block_copy_addr(3F)** returns the start address of a specific copy of the pointer block. **get_block_numcopies(3F)** returns *count*.

NOTE

The optional SIZE clause in the above definitions is required.

It is important to note that space in the shared memory segment is allocated to variables in the same order in which the variables are specified in the input to **shmdefine**.

Example

This example demonstrates how to enable two Fortran programs to cooperate using shared data.

This example utilizes a feature of Fortran called datapools. Concurrent Fortran supports datapools, but the GNU Fortran 77 compiler does not.

NOTE

If you do not have the Concurrent Fortran compiler, you can still follow this example and substitute common blocks for datapools. At each step, the example text will instruct you as to how to make the appropriate substitutions.

The example consists of a **`work`** program which does a simple numeric calculation and then exports the calculation by writing to the variable `calculation`. The second program, **`report`**, prints the value of `calculation`.

Binding shared memory to physical memory

If you wish to bind the shared memory segment to a particular section of physical memory, configure the target system such that the specified shared memory segment already exists and is bound to the appropriate physical address. The command ***`shmconfig`*** *can be used to accomplish this.*

Since our example does not require that the shared memory segment be bound to a section of physical memory, this step is not required.

NOTE

`shmconfig` is only available on RedHawk 2.1 and later.

Creating source programs

Create source programs.

For our example, create the following Fortran source files using a text editor of your choice:

- **`work.f`** (see "work.f" on page 1-10)
- **`report.f`** (see "report.f" on page 1-10)
- **`pool.dp`** (see "pool.dp" on page 1-11)

NOTE

If you do not have the Concurrent Fortran compiler, uncomment the lines currently commented out in the **`work.f`** and **`report.f`** source files (as indicated by "Note 1" below) and comment the line containing the datapool statement (as indicated by "Note 2" below). In this case, skip the creation of **`pool.dp`** as well.

work.f

```
      program work
      real*4 calculation
      real*4 data
C     common /blk/ data, calculation            Note 1
      datapool /pool/ data, calculation         Note 2
      call shm_init
      data = 1.0
10    continue
      calculation = data * 1.3
      data = calculation
      call sleep(1)
      goto 10
      end
```

report.f

```
      program report
      real*4 calculation
C     real*4 data                               Note 1
C     common /blk/ data, calculation            Note 1
      datapool /pool/ calculation
      call shm_init
10    continue
      call sleep(1)
      write (6,*) calculation
      goto 10
      end
```

pool.dp

```
real*4   data
real*4   calculation
datapool /pool/ data, calculation
end
```

Creating shmdefine input files

Create the ***shmdefine*** *input file(s).*

To perform this step, create the **shmdefine** input file using a text editor of your choice (see “Shared Regions” on page 1-4).

Create the input file, **shmdefine.input**

```
shared region share
   key = "/tmp/key"
   # fortran common blk size 8                See NOTE below
   fortran datapool pool, "pool.o"            See NOTE below
end shared region
```

The choice of the filename associated with the KEY attribute is arbitrary. It is used to ensure that a unique identifier for the shared memory segment is obtained and that access to the segment is limited to the cooperating programs (see “Attributes” on page 1-6). The file named in the KEY attribute must exist at the time the program is executed.

The name of the shared region is arbitrarily chosen by the user as well.

NOTE

If you do not have the Concurrent Fortran compiler, uncomment the line:

```
fortran common blk size 8
```

and comment out the line:

```
fortran datapool pool, "pool.o"
```

in **shmdefine.input**.

One advantage of using Fortran datapools is that **shmdefine** can automatically determine the size of the datapool by reading the object file corresponding to the datapool definition file.

For Fortran common blocks, it is the user's responsibility to specify the correct size of the common block in the **shmdefine** input file. In such cases, if you have an object file which describes the entire common block, the script **shmdefine.size** (shipped with the **shmdefine** utility) can help you in obtaining the size.

NOTE

shmdefine.size works only on Linux ELF 32-bit LSB relocatable object files.

Compiling the datapool definition file

NOTE

If you do not have the Concurrent Fortran compiler, skip this step.

Compile the datapool definition file using the following command.

```
% /usr/ccs/bin/cf77 -c -g pool.dp
```

Executing shmdefine

*Execute **shmdefine** with the desired options.*

Execute **shmdefine** using **shmdefine.input** as the input file. Note that the **-b** option designates **shmdef** as the base name for the object files produced by the utility (see "Command Syntax" on page 1-1). Also, the **-t** option instructs the **shmdefine** utility to generate output files.

```
% /usr/bin/shmdefine -b shmdef -t linux shmdefine.input
```

NOTE

If you do not have the Concurrent Fortran compiler, execute **shmdefine** as shown below:

```
% /usr/bin/shmdefine --gnu-f77 -b shmdef -t linux shmdefine.input
```

The **--gnu-f77** option tells **shmdefine** to use GNU gf77 naming rules for symbols. The initialization file (**shmdef.sm.c**) and the linker command file (**shmdef.sm.ld**) are created.

Compiling the initialization output file

*Compile the initialization output file that is produced by **shmdefine**.*

Compile the initialization file by invoking the C compiler. Note that a subsequent listing of your files will include the object file produced by the compiler.

```
% /usr/bin/gcc -c shmdef.sm.c
```

This creates the object file **shmdef.sm.o**.

When linking programs that use **shmdefine**, you must specify the linker configuration file generated by **shmdefine** as well as the compiled form of the C source file generated by **shmdefine**.

Compiling and linking the source programs

*Compile and link the source programs with the **shmdefine** initialization object file and the **shmdefine** link command output file.*

Compile and link the work and report programs by invoking Concurrent Fortran compiler in the following manner:

```
% /usr/ccs/bin/cf77 -g -o work work.f shmdef.sm.ld \
                                shmdef.sm.o pool.o
% /usr/ccs/bin/cf77 -g -o report report.f shmdef.sm.ld \
                                shmdef.sm.o pool.o
```

If you do not have the Concurrent Fortran compiler, use the following invocations of the GNU Fortran 77 compiler:

```
% /usr/bin/g77 -g -o work work.f shmdef.sm.ld shmdef.sm.o
% /usr/bin/g77 -g -o report report.f shmdef.sm.ld shmdef.sm.o
```

Running the programs

Invoke the **work** program in the background. It updates the variable `calculation` in the datapool `pool` once a second.

```
% ./work &
```

Invoke the **report** program. It prints the value of `calculation` once a second.

```
% ./report
2.8561
3.7129
4.8268
6.2749
8.1573
...
```

NOTE

The **report** program only refers to the variable `calculation`; it does not need to have a complete description of the datapool. This is the major advantage of datapools over `COMMON` blocks - the program only needs to reference the variables with which it is concerned.

Terminate the **report** program using the Ctrl-c keyboard sequence.

Terminate the **work** program by typing:

```
% fg
```

and then using the Ctrl-c keyboard sequence.