Startup-gen User's Guide

AdaCore

Apr 28, 2026

CONTENTS

CHAPTER

ONE

ABOUT STARTUP-GEN

`startup-gen` generates startup files (crt0 and linker script) based on properties of the target device such as: architecture, memory layout, number of interrupts.

One of the goals of this tool is to make it easier to start developing on micro-controllers by providing the basic elements of a Board Support Package (BSP).

The target device properties are provided inside a GNAT Project file (see *Input format*) and the output files are generated from templates in the template-parser format (see *Custom templates*).

CHAPTER

TWO

USAGE

To use `startup-gen` you need a GNAT Project file that contains the target device properties (see *Input format*).

Here is an example of project file:

```
project Spec is

  package Device_Configuration is

    for CPU_Name use "cortex-m4f";
    for Number_Of_Interrupts use "25";

    for Main_Stack_Size use ("4K");

    for Memories use ("flash", "sram");

    for Boot_Memory use "flash";

    for Main_Stack_Memory use "sram";
    for Main_Stack_Size use "8K";

    --  flash
    for Mem_Kind ("flash") use "ROM";
    for Address ("flash")  use "16#08000000#";
    for Size ("flash")     use "1024K";

    --  sram
    for Mem_Kind ("sram") use "RAM";
    for Address ("sram")  use "16#20000000#";
    for Size ("sram")     use "128K";

  end Device_Configuration;

end Spec;
```

The following command line switches are available:

- -l ARG Name of the generated linker script.
- -s ARG Name of the generated startup code.
- -X ARG Specify an external reference for Project Files.
- -P ARG Name of the project file with the device configuration.

- –print-tags Print the tags available in templates.

Here is an example of `startup-gen` command:

```
$ startup-gen -P spec.gpr -l link.ld -s crt0.S
```

This command will generate a linker script in the file `link.ld` and a crt0 in the file `crt0.S` based on the target properties provided in `spec.gpr`.

The `-P` switch is required. The linker script and/or crt0 will be generated only if respectively the `-l` and/or `-s` switches are provided.

CHAPTER

THREE

INPUT FORMAT

The properties of the target device must be provided in a `Device_Configuration` package inside a GNAT Project file.

Here are the attributes supported by `startup-gen`:

3.1 Architecture and CPU

- `for CPU_Name use ("<STRING>");` This attribute (optional if a supported run-time is defined) specifies the name of the target CPU. It is used, for instance, to determine the default templates that will be used for output generation.
- `for Float_Handling use "<soft|hard>";` This attribute is optional and specifies whether the floating point support is provided by the CPU/hardware (`hard`) or by the compiler/software (`soft`). It is used, for instance, to determine whether the Floating Point Unit (FPU) is initialized in the `crt0`. Note that you can decide to not use hardware floating point even if the target device has an FPU.

3.2 Memory

- `for Memories use (<LIST_OF_STRINGS>);` This attribute specifies the list of memory banks of the device, e.g. `("flash", "sram", "dtcm")`. For each bank of this list, the `Kind`, `Address` and `Size` attributes are **required**.
- `for Boot_Memory use "<BANK_NAME>";` This attribute specifies the memory bank from which the program will start. It can be a ROM bank if the program is written to a non-volatile memory such as flash, or a RAM bank if the program is loaded by a debugger or a bootloader.
- `for Mem_Kind (<BANK_NAME>) use "<ROM|RAM>";` This attribute specifies the kind of memory `<BANK_NAME>`, either `ROM` (read-only) or `RAM`. This attribute is used to determine what linker sections will be allocated to the bank. For instance a `.data` or `.bss` cannot be allocated in read-only memory.
- `for Address (<BANK_NAME>) use "<VALUE>";` This attribute specifies the base address of the memory bank. `<VALUE>` is a string containing an Ada or C numeric literal, e.g. `16#08000000#` or `0x08000000`.
- `for Size (<BANK_NAME>) use "<VALUE>";` This attribute specifies the size of the memory bank. `<VALUE>` is a string containing a numeric literal. Multiplier suffix `K` and `M` are supported, e.g. `16K`.
- `for Main_Stack_Size use "<VALUE>";` This attribute specifies the size of the main program stack statically allocated in the linker script. The value is a string containing a numeric literal. Multiplier suffix `K` and `M` are supported, e.g. `16K`
- `for Main_Stack_Memory use "<BANK_NAME>";` This attribute specifies the memory bank where the the main program stack is statically allocated in the linker script. The bank must be a RAM bank.

3.3 Interrupts

- `for Number_Of_Interrupts use "<VALUE>";` This attribute specifies the number of interrupt lines on the device. The value is a string containing a numeric literal. Depending on the architecture, this value can be used to create a trap vector in the `crt0`.
- `for Interrupt("<ID>") use "<NAME>";` This attribute specifies a name for an interrupt. `<ID>` is a numeric literal representing the interrupt id (starts at zero). `<NAME>` is a string that will be used to declare assembly symbols for instance. Therefore the name must be a valid assembly symbol name.

3.4 Templates

- `for Startup_Template use "<PATH>";` This attribute specifies a path to a custom template file used to control startup (`crt0`) generation (see *Custom Templates*).
- `for Linker_Template use "<PATH>";` This attribute specifies a path to a custom template file used to control linker script generation (see *Custom Templates*).
- `for User_Tag ("<TAG_NAME>") use "<TAG_VALUE>";` This attribute specifies a user defined tag to be used in templates. It can be used control optional features in templates or provide extra values that are not generated by startup-gen.

CHAPTER

FOUR

ADVANCED TOPICS

4.1 Scenario Variables

`startup-gen` supports the use of scenario variables in the input project file.

These can be used in multiple ways, here are some examples:

4.1.1 Select boot memory

```
project Prj is

   type Boot_Mem is ("flash", "sram");
   Boot : Boot_Mem := external ("BOOT_MEM", "flash");

   package Device_Configuration is

      for Memories use ("flash", "sram");

      for Boot_Memory use Boot;

      --  [...]
   end Device_Configuration;
end Prj;
```

```
$ startup-gen -P prj.gpr -l link.ld -s crt0.S -XBOOT_MEM=flash
$ startup-gen -P prj.gpr -l link.ld -s crt0.S -XBOOT_MEM=sram
```

4.1.2 Select boards with different device configuration

```
project Prj is

   type Board_Kind is ("dev_board", "production_board");
   Board : Board_Kind := external ("BOARD", "dev_board");

   package Device_Configuration is

      for Memories use ("flash", "sram");
```

(continues on next page)

(continued from previous page)

```
      case Board is
         when "dev_board" =>
            for Size ("sram")     use "256K";
         when "production_board" =>
            for Size ("sram")     use "128K";
      end case;

      --  [...]
   end Device_Configuration;
end Prj;
```

```
$ startup-gen -P prj.gpr -l link.ld -s crt0.S -XBOARD=dev_board
$ startup-gen -P prj.gpr -l link.ld -s crt0.S -XBOARD=production_board
```

4.2 Custom Templates

You can provide your own templates for startup and linker script files using the `Startup_Template` and `Linker_Template` attributes (see *Templates*).

The output files are generated using the template-parser library, you can therefore refer to this documentation to write your templates.

To see the list of predefined tags available for your templates, use the `--print-tags` command line switch.

For instance:

```
$ startup-gen -P spec.gpr --print-tags
--- Template tags ---
BOOT_FROM_ROM => TRUE
BOOT_MEM => ("flash")
BOOT_MEM_ADDR => ("0x8000000")
BOOT_MEM_SIZE => ("0x100000")
MAIN_RAM => ("sram")
MAIN_RAM_ADDR => ("0x20000000")
MAIN_RAM_SIZE => ("0x20000")
RAM_REGION => ("ccm")
RAM_ADDR => ("0x10000000")
RAM_SIZE => ("0x10000")
ROM_REGION => ()
ROM_ADDR => ()
ROM_SIZE => ()
MAIN_STACK_SIZE => ("0x1000")
MAIN_STACK_REGION => ("ccm")
INTERRUPT_NAME => ()
INTERRUPT_ID => ()
---------------------
[...]
```

CHAPTER

FIVE

EXAMPLE OF PROJECT USING A GENERIC LIGHT ADA RUN-TIME AND STARTUP-GEN

For this example we will use STM32F4-Discovery development board from STmicroelectronics. The board is equipped with an ARM Cortex-M4F microcontroller.

The sources of this example can be found in: *<install_directory>/share/examples/startup-gen/startup-gen-stm32f4*

5.1 Board specifications

To begin with we need to know the specification of the board and microcontroller. We will need:

- The name of the CPU core architecture (ARM Cortex-M4F in our case)
- Base address and size of memory banks (flash, RAM, etc.)
- The number of interrupts

You can get the information from the vendor documentation or product page.

5.2 The project file

The project file will require some specific fields:

- The list of languages must contain *ASM_CPP*, because we will compile the startup code (crt0) written in assembly language.
- The run-time should be set to *light-cortex-m4f* because we are using an ARM Cortex-M4F microcontroller
- The linker script must be specified as a linker option
- The board specifications in a *Device_Configuration*

Here is what the project file looks like:

```
project Hello is

   for Languages use ("Ada", "ASM_CPP"); -- ASM_CPP to compile the startup code
   for Source_Dirs use ("src");
   for Object_Dir use "obj";
   for Main use ("hello.adb");

   for Target use "arm-eabi";

```

(continues on next page)

(continued from previous page)

```
   --  generic Light run-time compatible with our MCU
   for Runtime ("Ada") use "light-cortex-m4f";

   package Linker is
      --  Linker script generated by startup-gen
      for Switches ("Ada") use ("-T", Project'Project_Dir & "/src/link.ld");
   end Linker;

     package Device_Configuration is

      --  Name of the CPU core on the STM32F407
      for CPU_Name use "ARM Cortex-M4F";

      for Float_Handling use "hard";

      --  Number of interrupt lines on the STM32F407
      for Number_Of_Interrupts use "82";

      --  List of memory banks on the STM32F407
      for Memories use ("SRAM", "FLASH", "CCM");

      --  Specify from which memory bank the program will load
      for Boot_Memory use "FLASH";

      --  Allocate the main stack in CCM with 8K size
      for Main_Stack_Memory use "CCM";
      for Main_Stack_Size use "8K";

      --  Specification of the SRAM
      for Mem_Kind ("SRAM") use "ram";
      for Address ("SRAM") use "0x20000000";
      for Size ("SRAM") use "128K";

      --  Specification of the FLASH
      for Mem_Kind ("FLASH") use "rom";
      for Address ("FLASH") use "0x08000000";
      for Size ("FLASH") use "1024K";

      --  Specification of the CCM RAM
      for Mem_Kind ("CCM") use "ram";
      for Address ("CCM") use "0x10000000";
      for Size ("CCM") use "64K";

   end Device_Configuration;
end Hello;
```

5.3 Generate the linker script and startup-code with startup-gen

Once the project file is ready we can use startup-gen to generate the linker script and startup code.

To do this use the following command line:

```
$ startup-gen -P hello.gpr -l src/link.ld -s src/crt0.S
```

This means that startup-gen will create a linker script in src/link.ld and a startup code in src/crt0.S

5.4 Build

We can now build our project:

```
$ gprbuild -P hello.gpr
```

You can also open this project in GNATstudio and build it from there.

5.5 Running on the GNATemulator

You can now run the example on GNATemulator:

```
$ arm-eabi-gnatemu --board=STM32F4 obj/hello
```