A Command-Line Driver Generator or What I did when I got tired of writing command-line interfaces myself...

Jacob Sparre Andersen

JSA Research & Innovation

January 2016

A bit of history

The declarations of subprograms in the public part of an Ada package specification can be seen as a declaration of how you are supposed to call the features implemented in the package.

So why should I have to write command-line interfaces myself? Why not let a tool translate between the Ada package specification and the command-line interface?

Last year I was sufficiently tired of writing command-line interfaces, to actually start writing such a tool, and in August 2015 it was released to the unsuspecting public under an Open Source license.

```
with Ada.Strings;
package Trim is
   use Ada.Strings;
   subtype File_Name is String;

   procedure Filter (Sides : in     Trim_End := Both);
   --  Reads lines from standard input,
   --  removes leading/trailing spaces, and
   --  writes the trimmed lines to standard output.

   procedure Copy (Source : in     File_Name;
                   Target : in     File_Name;
                   Sides  : in     Trim_End := Both);
   --  Reads lines from the file Source,
   --  removes leading/trailing spaces, and
   --  writes the trimmed lines to the file Target.
end Trim;
```

```
# Filter:
trim-driver
trim-driver --sides=both
trim-driver --sides=left
trim-driver --sides=right

# File to file:
trim-driver --Source=trim --target=trimmed
trim-driver --Sides=both --Source=trim --target=trimmed
trim-driver --Source=trim --Target=trimmed --Sides=LEFT
trim-driver --sides=right --target=trimmed --Source=trim
```

Small interactive demo.[1]

[1] I have copied "`examples/generated/_trim-driver`" to "`~/.zsh/`" and put "`examples/bin/trim-driver`" in my path.

- Note where source files the package **specification** depends on. List these directories in the environment variable "`ADA_INCLUDE_PATH`".
- Call the tool, "`command_line_parser_generator-run`"[2] with the package **name**[3] as the only argument.
- Find the generated source files (and a Zsh command-completion specification) in the directory "`generated/`" (relatively to where you ran the tool).

[2] Unless you've renamed it to something more sensible.

[3] Specifically, **not** the name of the file containing the package specification.

Input

```
% ls
ada_2012.gpr
default.gpr
trim.adb
trim.ads
```

We have earlier seen that the specification of "`Trim`" only withs "`Ada.Strings`", so we don't need to include other directories in "`ADA_INCLUDE_PATH`".

Running

```
% export ADA_INCLUDE_PATH=.
% command_line_parser_generator-run Trim
Warning: "   use Ada.Strings;" is ignored.
Warning: "   subtype File_Name is String;" is ignored.
package Trim is

   procedure Filter (Sides : in     Ada.Strings.Trim_End
        := Both);
   procedure Copy (Source : in     Trim.File_Name;
       Target : in     Trim.File_Name; Sides : in     Ada
       .Strings.Trim_End := Both);

end Trim;
```

We see how the tool interprets the package specification.

Output

```
% ls generated
_trim-driver
trim-command_line_parser-argument.adb
trim-command_line_parser-argument.ads
trim-command_line_parser-argument_list.adb
trim-command_line_parser-argument_list.ads
trim-command_line_parser-key_list.adb
trim-command_line_parser-key_list.ads
trim-command_line_parser-profiles.adb
trim-command_line_parser-profiles.ads
trim-command_line_parser.adb
trim-command_line_parser.ads
trim-driver.adb
trim-driver.ads
trim-put_help.adb
trim-put_help.ads
trim-show_help.adb
```

The tool ...

1. parses an Ada package specification (using the Ada Semantic Interface Specification, ASIS),
2. checks that the public part of the package specification doesn't declare procedures with `out` or `aliased` parameters – or functions,
3. decides how to map command-line arguments (of type `String`) to the types of the declared formal parameters[4],
4. generates a command-line driver for the package,
5. generates help texts for the command-line driver (if they aren't provided by the package), and finally
6. generates Zsh command-completion patterns (because I'm very lazy :-).

[4] Selecting mapping functions is an area where the tool needs more work.

From the beginning of the main procedure:

```
with Ada.Characters.Handling,
     Ada.Command_Line,
     Ada.Text_IO,
     Ada.Wide_Text_IO;

with Asis,
     Asis.Ada_Environments,
     Asis.Compilation_Units,
     Asis.Declarations,
     Asis.Elements,
     Asis.Implementation,
     Asis.Text;

with Command_Line_Parser_Generator.Formal_Parameter,
     Command_Line_Parser_Generator.Help,
     Command_Line_Parser_Generator.Identifier_Set,
```

Getting your hands on a specific compilation unit:

```
   use Asis.Compilation_Units;
   use all type Asis.Unit_Kinds;
begin
   Compilation_Unit := Library_Unit_Declaration
                         (Name         => To_Wide_String
                             (Argument (1)),
                          The_Context => Context);

   case Unit_Kind (Compilation_Unit) is
      when A_Package =>
```

Checking the mode of a formal parameter:

```
case Mode_Kind (Parameter) is
   when A_Default_In_Mode | An_In_Mode =>
      null; --  Fine, we continue.
   when An_In_Out_Mode | An_Out_Mode =>
      Put_Line (File => Standard_Error,
                Item => "Out parameters not allowed.");
      Set_Exit_Status (Failure);
      return;
   when Not_A_Mode =>
      Put_Line (File => Standard_Error,
                Item => "ASIS error.");
      Set_Exit_Status (Failure);
      return;
end case;
```

Some numbers

GNAT standard library omissions	52 lines
Templates	1’050 lines
Data structures and logic	1’579 lines

Counting raw source text lines.

- Refactoring the main procedure in the tool. It has been growing quite “organic”, and needs more structure.
- Adding missing/skipped features.
- Deciding which Ada type is the best match for a “flag” on the command-line.
- Improving the help and error messages.
- Creating a variation of the tool for producing a web interface to a package[5].

[5] As an alternative to the AWS tools “`ada2wsdl`” and “`wsdl2ada`”.

Contact

Jacob Sparre Andersen
JSA Research & Innovation
jacob@jacob-sparre.dk
http://www.jacob-sparre.dk/

Examples from this presentation:
http://www.jacob-sparre.dk/ada/command-line_driver_generator/fosdem-2016-examples.zip

You can find my Open Source software repositories at:
http://repositories.jacob-sparre.dk/