Gem #139: Освойте командную строку - Часть 2

Автор: Emmanuel Briot — AdaCore

Краткое содержание: В первой части данной серии было описано, как использовать команду `GNAT.Command_Line` для получения параметров и аргументов, передаваемых приложению. Для этого все еще необходимо написание большого объема кода. В данной части будет пояснен высокоуровневый API команды `GNAT.Command_Line` и показано, как значительно упростить процесс работы с командной строкой.

Давайте начнём…

Высокоуровневый API

Как упоминалось в части 1, `GNAT.Command_Line` также предоставляет высокоуровневый API, где большая часть обработки полностью автоматическая. Вот пример его использования, аналогичный примеру, рассмотренному в части 1:

```
with GNAT.Strings;  use GNAT.Strings;

declare
   Config : Command_Line_Configuration;
   A_Enabled   : Boolean := False;
   B_Option    : String_Access;
   Long_Option : Integer := 0;

   procedure Callback (Switch, Param, Section : String) is
   begin
      if Switch = "-a" then
         A_Enabled := True;
      elsif Switch = "-b" then
         B_Option := new String'(Param);
      elsif Switch = "--long" then
         Long_Option := Integer'Value (Param);
      elsif Switch = "--help" then
         Display_Help (Config);    -- 1
      end if;
   end Callback;

begin
   Define_Switch (Config, "-a", Help => "Enable option a");   -- 2
   Define_Switch (Config, "-b:", Help => "Enable option b");
   Define_Switch (Config, Long_Switch => "--long=",
                  Help => "Enable long option. Arg is an integer");
   Define_Switch (Config, Long_Switch => "--help",
                  Help => "Display help");

   Getopt (Config, Callback'Access);   -- 3
end;

Put_Line ("File argument was " & Get_Argument);  --  7
```

Код не намного короче, чем в предыдущем примере, но на самом деле он более реалистичен, поскольку сохраняет значения параметров различных переключателей, поскольку предположительно эти параметры используются для настройки приложения каким-либо образом.

У нас больше нет явного цикла для получения переключателей один за другим. Вместо этого вызов `Getopt` на шаге 3 автоматически вызывает `Callback` для каждого найденного им переключателя (как определено вызовами `Define_Switch` на шаге 2).

На шаге 1 мы видим, что документация для списка допустимых ключей может быть сгенерирована автоматически. Это имеет большое преимущество обеспечения согласованности между тем, что задокументировано, и тем, что доступно. Документация для каждого коммутатора также близка к его определению.

Тем не менее, этот API требует много ручного управления для сохранения параметров. Это также не безопасно с точки зрения типа. Например, нет указания на то, что параметр `--long`, как ожидается, будет целым числом, кроме как в справочном сообщении.

Таким образом, `GNAT.Command_Line` предоставляет еще более удобный API, который можно комбинировать с приведенным выше, как показано здесь:

```
with GNAT.Strings;  use GNAT.Strings;

declare
   Config : Command_Line_Configuration;
   A_Enabled   : aliased Boolean := False;
   B_Option    : aliased String_Access;
   Long_Option : aliased Integer := 0;

begin
   Define_Switch (Config, A_Enabled'Access, "-a",
                  Help => "Enable option a");
   Define_Switch (Config, B_Option'Access, "-b:",
                  Help => "Enable option b");
   Define_Switch (Config, Long_Option'Access,
                  Long_Switch => "--long=",
                  Help => "Enable long option. Arg is an integer");

   Getopt (Config);
end;

Put_Line ("File argument was " & Get_Argument);
```

Теперь мы только объявляем коммутаторы и указываем, где должны храниться их параметры. Нам больше не нужно явно упоминать "`--help`", который автоматически поддерживается.

Этот API высокого уровня также поддерживает использование разделов и автоматически обрабатывает аргументы в каждом разделе. Он также поддерживает чтение переключателей из списка строк (через объект `Opt_Parser`, как и раньше).

При объявлении коммутаторов предоставляются различные подпрограммы, указывающие, как их можно объединить в командной строке. Например:

```
Define_Prefix (Config, "-gnaty");
Define_Switch (Config, "-gnatya");
Define_Switch (Config, "-gnatyb");
```

будет анализировать командную строку «`-gnatyab`» как комбинацию двух ключей «`-gnatya -gnatyb`».

Кроме того, вы можете объявить псевдонимы. Например, в дополнение к вышесказанному вы можете использовать:

```
Define_Alias (Config, "-gnaty", "-gnatyab");
```

что также позволит пользователям использовать "-gnaty" как эквивалент "-gnatya -gnatyb".

Тем не менее, помните, что ваши параметры командной строки должны быть простыми!

Создание командных строк

Наконец, GNAT.Command_Line позволяет вам создавать командную строку. Это редко требуется, но это может принести пользу приложениям, таким как IDE, которые позволяют редактировать переключатели, используемые при порождении приложений.

С указанным выше объектом Config приложение теперь может быть написано довольно просто следующим образом:

```
declare
   Cmd  : Command_Line;
   Args : Argument_List_Access;

begin
   Set_Configuration (Cmd, Config);
   Add_Switch (Cmd, "-gnatya");
   Add_Switch (Cmd, "-a");
   Add_Switch (Cmd, "-gnatyb");

   Build (Cmd, Args);
   --  Args now contains ("-gnaty", "-a"), which is the shortest
   --  representation of the switches we specified.
end;
```

Связанный со статьёй текст программы

Файлы примеров Ada Gems распространяются AdaCore и могут быть использованы или изменены для любых целей без ограничений.

Об авторе

Автор: Emmanuel Briot - AdaCore

Эммануэль Брио работает в AdaCore с 1998 года. Он принимал участие в различных проектах, в частности, ориентированных на графические пользовательские интерфейсы, включая GtkAda, GPS, XML / Ada, GnatTracker и нашу внутреннею CRM. Он получил степень инженера в Национальной школе телекоммуникаций - Брест, Франция (Ecole Nationale des Telecommunications - Brest, France).

Last Updated: 10/13/2017
Posted on: 1/14/2013
Обсуждение…