# *Gem #54: Возможности создания скриптов в GNAT (Часть 2)*

**Автор:** Emmanuel Briot, AdaCore

Краткое содержание: Gem Ада #54 - В Gem # 52 мы показали несколько возможностей GNAT, которые превращают язык Ада в язык скриптования (в какой-то степени). Мы продемонстрировали функции `GNAT.Command_Line`, `GNATCOLL.Mmap`, `GNAT.Regpat` и `GNAT.AWK`. В данном Gem продолжается обсуждение возможностей описания сценариев, на этот раз - в контексте работы с внешними процессами.

**Давайте начнем…**

У многих приложений есть потребность выполнить команды на машине. Существует несколько вариантов использования: программа может пожелать дождаться завершения команды (например, команда создает файл, который приложение затем должно проанализировать); она может порождать команду в фоновом режиме и продолжать выполняться в то же время; или ей может потребоваться взаимодействие с внешним приложением (отправка данных в его входной поток и чтение его выходных данных).

Порождение процесса, с которым приложению не нужно взаимодействовать, легко реализовать. В `GNAT run-time` предоставляется пакет `GNAT.OS_Lib` содержащий ряд подпрограмм, которые обеспечивают низкоуровневый интерфейс к системе. В частности, пакет предоставляет несколько подпрограмм с именами `Spawn` и `Non_Blocking_Spawn`, которые, как указывает их имя, будут порождать внешнее приложение и ждать или нет его завершения.

```
with GNAT.OS_Lib;  use GNAT.OS_Lib;

procedure Main is
   Command : constant String := "myapp -f 'a string with spaces'";
   --  We assume the slightly more complex case where the arguments of the
   --  command are part of the same string (this is generally the case when
   --  the command is provided interactively by the user).

   Args        : Argument_List_Access;
   Exit_Status : Integer;
   Pid         : Process_Id;
   Success     : Boolean;

begin
   --  Prepare the arguments. Splitting properly takes quotes into account.

   Args := Argument_String_To_List (Command);

   --  Spawn the command and wait for its possible completion

   if Background_Mode then
      Pid := Non_Blocking_Spawn
         (Program_Name => Args (Args'First).all,
          Args         => Args (Args'First + 1 .. Args'Last));

      --  We could also wait for completion:
      --  Wait_Process (Pid, Success);

   else
      Exit_Status := Spawn
         (Program_Name => Args (Args'First).all,
          Args         => Args (Args'First + 1 .. Args'Last));
   end if;

   --  Free memory

   Free (Args);
```

```
end Main;
```

Во время обсуждения GNAT.OS_Lib, стоит исследовать оставшиеся подпрограммы. Они обеспечивают независимый от системы манипулирования файлами (проверка, существует ли файл, будь то каталог или символьная ссылка, доступ к Дата последнего изменения файла, копирование, удаление и переименование файлов и т. д.). Одной из очень интересных подпрограмм является Normalize_Pathname, которая способна разрешать символьные ссылки и преобразовывать регистр имен файлов в зависимости от того, доступны ли они с помощью уникального имени или без учета регистра. Это обеспечивает удобный способ проверить, совпадают ли два имени файлов (обработка "..", символические ссылки, и разрешение дела довольно сложно получить правильно и системно-независимым, и GNAT.OS_Lib заботится об этом для вас).

Среди трех вариантов использования, о которых мы упоминали в начале, наиболее сложным является то, где приложение должно взаимодействовать с процессом. Такое взаимодействие обеспечивается через GNAT.Expect пакет. Здесь вы можете создать процесс и во время его выполнения отправить его Входные данные и прочитать Выходные данные. Сообщение между двумя процессами сделано через pipes (каналы/трубы). Другая реализация, использующая подход, основанный на ttys (псевдотерминалах), планируется сделать доступной как часть многократно используемых компонентов GNAT Reusable Components и предназначена для обеспечения более тесной эмуляции того, что происходит, когда вы порождаете процесс из терминала.

Вот пример использования GNAT.Expect:

```
with GNAT.Expect;  use GNAT.Expect;

procedure Main is
   Command : constant String := "gdb myapp"; --  Let's spawn a debugger session.
   Pd      : Process_Descriptor;
   Args    : Argument_List_Access;
   Result  : Expect_Match;

begin
   --  First we spawn the process. Splitting the program name and the
   --  arguments is done as in the previous example, using
   --  Argument_String_To_List. Remember to free the memory at the end.

   Args := Argument_String_To_List (Command);

   Non_Blocking_Spawn
      (Pd,
       Command     => Args (Args'First).all,
       Args        => Args (Args'First + 1 .. Args'Last),
       Buffer_Size => 0);

   --  The debugger is now running. Let's send a command.

   Send (Pd, "break 10");

   --  Then let's read the output of the debugger.
   --  Here, we are expecting any possible non-empty output (hence the
   --  ".+" regexp). We might in fact be expecting a file name that
   --  gdb uses to confirm a breakpoint. The regexp would be something like:
   --   "file (.*), line (\d+)"

   Expect (Pd, Result, Regexp => ".+", Timeout => 1_000);

   case Result is
      when Expect_Timeout => ...;  --  gdb never replied
      when 1  => ...;              --  the regexp matched
          --  We then have access to all the output of gdb since the
          --  last call to Expect, through the Expect_Out subprogram.
```

```
   end case;

   --  When we are done, terminate gdb:

   Close (Pd);
end Main;
```

Этот пример действительно лишь кратко коснулся возможностей `GNAT.Expect`. В качестве дополнительной заметки, весь GPS, наша интегрированная среда разработки использует этот пакет для взаимодействия с внешними процессами.

**Связанный со статьёй текст программы**

**Attached Files отсутствуют**

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

**Об авторе**

Сведения отсутствуют.

*Last Updated: 10/13/2017*
*Posted on: 11/24/2008*

**Обсуждение…**