Gem #15: Таймеры

Автор: Anh Vo, Santa Clara, California

Краткое содержание: Gem Ада #15- Таймеры - весьма важные программные элементы встроенных систем и систем, работающих в реальном времени. Простой и понятный способ создания таймеров облегчает задачу разработки программного обеспечения.

Давайте начнем…

Таймеры являются важными программными элементами встроенных и систем реального времени. Таким образом, создание интуитивно понятного и простого способа создания таймеров помогает в процессе разработки программного обеспечения. Ada предоставляет предопределенный пакет с именем Ada.Real_Time.Timing_Events для этого. С помощью этого пакета можно создать таймер, одноразовый или периодический, с бриз. Кроме того, эти таймеры могут быть обобщены для разных длительностей. На следующем шаге эти общие таймеры могут быть объединены в один общий пакет, как показано ниже, вместе с тестовыми кодами. Обратите внимание, что тестовые коды довольно грубые. Однако выход распечатывается каждую секунду приблизительно. Таким образом, результаты тестирования можно проверить, используя правило подсчета 1000, тысяча одна, одна тысяча два ...

Вот пример создания 250-миллисекундного таймера с одним срабатыванием.

```
with Generic_Timers;
with Ada.Real_Time;
with Ada.Text_Io;

--...
declare
   use Ada;
   Span : constant Time_Span := Real_Time.Milliseconds (250);
   Timer_Id : constant String := "250 Millisecond One Shot timer";

   procedure Timer_Handler is
   begin
      Put_Line ("Do whatever is necessary when the timer expires");
   end Timer_Handler;

   package The_Timer is new Generic_Timers (True, Timer_Id, Span,
Timer_Hander); --timer object
begin
   --...
   The_Timer.Start;
   --...
end;
--...
```

Другие разовые таймеры и периодические таймеры для разных длительностей в тестовом пакете Timers_Test. Обратите внимание, что в моем текущем проекте создания таймера занимает от пяти до десяти раз дольше и гораздо менее понятным. На этой ноте наслаждайтесь им с улыбкой.

Связанный исходный код

```
with Ada.Real_Time.Timing_Events;

generic
   One_Shot : Boolean := True;
   Timer_Name : String := "Generic_Timers";
   For_Duration : in Ada.Real_Time.Time_Span;
   with procedure Action is <>;

package Generic_Timers is

   Timer_Error : exception;

   procedure Start;
   procedure Stop;
   procedure Cancel;

private

   The_Event : Ada.Real_Time.Timing_Events.Timing_Event;

end Generic_Timers;
package Timers_Test is

   procedure Start;
   procedure Shutdown;

end Timers_Test;
with Ada.Exceptions;
with Ada.Text_Io;

with Timers_Test;

procedure All_Timers_Test is

   use Ada;
   use Text_Io;

begin

   Put_Line ("All Timers Test");

   Timers_Test.Start;
   delay 60.0;
   Timers_Test.Shutdown;

exception
   when Error : others =>
      Put_Line ("Testing fails for because of ==> " &
                                    Exceptions.Exception_Information (Error));
end All_Timers_Test;
package body Generic_Timers is

   use Ada;

   protected Events is
      procedure Handler (Event: in out Real_Time.Timing_Events.Timing_Event);
   end Events;

   protected body Events is
    procedure Handler (Event: in out Real_Time.Timing_Events.Timing_Event) is
      begin
```

```
         Action;
         if not One_Shot then
            Start;  -- periodic timer continues
         end if;
      end Handler;
   end Events;

   procedure Start is
      use type Ada.Real_Time.Timing_Events.Timing_Event_Handler;
   begin
      if Real_Time.Timing_Events.Current_Handler (The_Event) = null then
         Real_Time.Timing_Events.Set_Handler (
                              The_Event, For_Duration, Events.Handler'access);
      else
         raise Timer_Error with Timer_Name & " started already";
      end if;
   end Start;

   procedure Stop is
      Success : Boolean := False;
      use type Ada.Real_Time.Timing_Events.Timing_Event_Handler;
   begin
      if Real_Time.Timing_Events.Current_Handler (The_Event) /= null then
         Real_Time.Timing_Events.Cancel_Handler (The_Event, Success);
         if not Success then
            raise Timer_Error with "fails to cancel " & Timer_Name;
         end if;
      end if;
   end Stop;

   procedure Cancel renames Stop;

end Generic_Timers;

with Ada.Real_Time;
with Ada.Text_Io;

with Generic_Timers;

package body Timers_Test is

   use Ada;
   use Real_Time;
   use Text_Io;

-------------------------------------------------------------------------
-- Below are generic one shot Timers being tested                      --
-------------------------------------------------------------------------
   Three_Seconds : constant Time_Span := Real_Time.Milliseconds (3000);
   Three_Second_Timer_Id : constant String := "Three Second One Shot timer";

   procedure Action_Three is
   begin
      Put_Line ("Three (3) second one shot timer, Generic_Timers, expires");
   end Action_Three;

   Package Three_Second_One_Shot_Timer is new Generic_Timers (
                  True, Three_Second_Timer_Id, Three_Seconds, Action_Three);

   ------------------------------------------------------
   Five_Seconds : constant Time_Span := Real_Time.Milliseconds (5000);
   Five_Second_Timer_Id : constant String := "Five Second One Shot timer";

   procedure Action_Five is
```

```
   begin
      Put_Line ("Five (5) second one shot timer, Generic_Timers, expires");
   end Action_Five;

   Package Five_Second_One_Shot_Timer is new Generic_Timers (
                        True, Five_Second_Timer_Id, Five_Seconds,
Action_Five);

-------------------------------------------------------------------------
-- Below are generic Periodic Timers being tested                       --
-------------------------------------------------------------------------
   One_Seconds : constant Time_Span := Real_Time.Milliseconds (1000);

   procedure Action_One is
   begin
      Put_Line ("One (1) second cyclic timer, Generic_Timers, expires");
   end Action_One;

   Package One_Second_Periodic_Timer is new Generic_Timers (
                  False, "One Second Periodic Timer", One_Seconds,
Action_One);

   ------------------------------------------------------
   Two_Seconds : constant Time_Span := Real_Time.Milliseconds (2000);

   procedure Action_Two is
   begin
      Put_Line ("Two (2) second cyclic timer, Generic_Timers, expires");
   end Action_Two;

   Package Two_Second_Periodic_Timer is new Generic_Timers (
                  False, "Two Second Periodic Timer", Two_Seconds,
Action_Two);

   ------------------------------------------------------
   Four_Seconds : constant Time_Span := Real_Time.Milliseconds (4000);

   procedure Action_Four is
   begin
      Put_Line ("Four (4) second cyclic timer, Generic_Timers, expires");
   end Action_Four;

   Package Four_Second_Periodic_Timer is new Generic_Timers (
                False, "Four Second Periodic Timer", Four_Seconds,
Action_Four);

   ------------------------------------------------------
   Eight_Seconds : constant Time_Span := Real_Time.Milliseconds (8000);

   procedure Action_Eight is
   begin
      Put_Line ("Eight (8) second cyclic timer, Generic_Timers, expires");
   end Action_Eight;

   Package Eight_Second_Periodic_Timer is new Generic_Timers (
            False, "Eight Second Periodic Timer", Eight_Seconds,
Action_Eight);

-------------------------------------------------------------------------
-------------------------------------------------------------------------

   procedure Start is
   begin
      Put_Line ("Timers Test begins");
```

```
      One_Second_Periodic_Timer.Start;
      Two_Second_Periodic_Timer.Start;
      Four_Second_Periodic_Timer.Start;
      Eight_Second_Periodic_Timer.Start;

      for Index in 1 .. 2 loop
         Three_Second_One_Shot_Timer.Start;
         Five_Second_One_Shot_Timer.Start;
         delay 6.0;
      end loop;
   end Start;

   procedure Shutdown is
   begin
      One_Second_Periodic_Timer.Cancel;
      Two_Second_Periodic_Timer.Cancel;
      Four_Second_Periodic_Timer.Cancel;
      Eight_Second_Periodic_Timer.Cancel;

      Three_Second_One_Shot_Timer.Cancel;
      Five_Second_One_Shot_Timer.Cancel;
      Put_Line ("Timers testing ends");
   end Shutdown;

end Timers_Test;
```

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

Обсуждение…

3 комментария
Anh Vo
30 октября 2007 г.

После прочтения собственного драгоценного камня я заметил два опечатка. Первый - в процедуре Timer_Handler. Строка должна читать «Делать все, что необходимо, когда таймер истекает (дополнительно). Вторая опечатка начинается с предложения, где отсутствует s. Он должен читать с помощью Generic_Timers.

Томас Кино
2 ноября 2007 г.

Спасибо Anh, мы обновили текст драгоценного камня.

Марк
26 февраля 2010 г.

Я продолжаю получать предупреждение компиляции с приведенным выше кодом (другой пример работает как ветер): «не локальный указатель не может указывать на локальный объект».

И тогда есть опечатка в разработке пакета The_Timer, третьим аргументом должен быть Time_Handler, а не Time_Hander.

Иначе настоящая жемчужина. Я бы никогда не узнал. Благодарю!