Documentation

Table of Contents
Class Inheritance Tree

```
--
--  Uwe R. Zimmer, Australia, September 2016
--

with Ada.Float_Text_IO;                use Ada.Float_Text_IO;
with Ada.Integer_Text_IO;              use Ada.Integer_Text_IO;
with Ada.Numerics.Discrete_Random;     use Ada.Numerics;
with Ada.Text_IO;                      use Ada.Text_IO;
with Generic_Message_StructuresTo API docTo spec;
with Generic_RouterTo API docTo specTo body;
with Generic_RoutersTo API docTo specTo body;
with Generic_Routers_ConfigurationTo API docTo spec;
with GNAT.Command_Line;                 use GNAT.Command_Line;
with Routers_Configuration_StructuresTo API docTo specTo body; use
Routers_Configuration_StructuresTo API docTo specTo body;
with TopologiesTo API docTo specTo body;                      use
TopologiesTo API docTo specTo body;

procedure Test_Routers is

   Command_Line_Parameters : Command_Line_OptionsTo API docTo spec;
   Options_Ok              : Boolean              := True;

   procedure Print_Options is

   begin
      New_Line; Put ("accepted options:");
      New_Line; Put ("   [-t {Topology            : String   }] -> "); Put (Preconfigured_TopologiesTo API docTo spec&apos;Image (Command_Line_Parameters.Selected_Topology));
      New_Line; Put ("      by Size            : Line, Ring, Star, Fully_Connected");
      New_Line; Put ("      by Degree, Depths  : Tree");
      New_Line; Put ("      by Dimension, Size : Mesh, Torus");
      New_Line; Put ("      by Dimension       : Hypercube, Cube_Connected_Cycles,");
      New_Line; Put ("                           Butterfly, Wrap_Around_Butterfly");
      New_Line; Put ("   [-s {Size                : Positive }] -> "); Put (Command_Line_Parameters.Size, 3);
      New_Line; Put ("   [-g {Degree              : Positive }] -> "); Put (Command_Line_Parameters.Degree, 3);
      New_Line; Put ("   [-p {Depths              : Positive }] -> "); Put (Command_Line_Parameters.Depths, 3);
      New_Line; Put ("   [-d {Dimension           : Positive }] -> "); Put (Command_Line_Parameters.Dimension, 3);
      New_Line; Put ("   [-c {Print connections   : Boolean  }] -> "); Put (Boolean&apos;Image (Command_Line_Parameters.Print_Connections));
      New_Line; Put ("   [-i {Print distances     : Boolean  }] -> "); Put (Boolean&apos;Image (Command_Line_Parameters.Print_Distances));
      New_Line; Put ("   [-w {Routers settle time : Seconds  }] -> "); Put (Float (Command_Line_Parameters.Routers_Settle_Time), 2, 2, 0);
      New_Line; Put ("   [-o {Comms timeout       : Seconds  }] -> "); Put (Float (Command_Line_Parameters.Comms_Timeout), 2, 2, 0);
      New_Line; Put ("   [-m {Test mode           : String   }] -> "); Put (Test_ModesTo API docTo spec&apos;Image (Command_Line_Parameters.Test_Mode));
      New_Line; Put ("      Available modes: One_to_All, All_to_One");
      New_Line; Put ("   [-x {Dropouts            : Natural  }] -> "); Put (Command_Line_Parameters.Dropouts, 3);
      New_Line;
      New_Line;
   end Print_OptionsTo specTo body;

begin
   Initialize_Option_Scan;
   loop
```

```
      declare
         Option : constant Character := Getopt ("t: s: g: p: d: c: i: w: o: m: x:");
      begin
         case Option is
            when ASCII.NUL => exit;
            when &apos;t&apos; => Command_Line_Parameters.Selected_Topology   := Preconfigured_TopologiesTo API docTo spec&apos;Value (Parameter);
            when &apos;s&apos; => Command_Line_Parameters.Size                := Positive&apos;Value (Parameter);
            when &apos;g&apos; => Command_Line_Parameters.Degree              := Positive&apos;Value (Parameter);
            when &apos;p&apos; => Command_Line_Parameters.Depths              := Positive&apos;Value (Parameter);
            when &apos;d&apos; => Command_Line_Parameters.Dimension           := Positive&apos;Value (Parameter);
            when &apos;c&apos; => Command_Line_Parameters.Print_Connections   := Boolean&apos;Value (Parameter);
            when &apos;i&apos; => Command_Line_Parameters.Print_Distances     := Boolean&apos;Value (Parameter);
            when &apos;w&apos; => Command_Line_Parameters.Routers_Settle_Time := Duration&apos;Value (Parameter);
            when &apos;o&apos; => Command_Line_Parameters.Comms_Timeout       := Duration&apos;Value (Parameter);
            when &apos;m&apos; => Command_Line_Parameters.Test_Mode           := Test_ModesTo API docTo spec&apos;Value (Parameter);
            when &apos;x&apos; => Command_Line_Parameters.Dropouts            := Natural&apos;Value (Parameter);
            when others => raise Program_Error;
         end case;
      exception
         when others =>
            New_Line; Put ("---> Error in option -"); Put (Option); New_Line;
            Options_Ok := False;
      end;
   end loop;

   Print_OptionsTo specTo body;

   if Options_Ok then

      New_Line;
      Put_Line ("---------------------- Instantiating router tasks ----------------------------");

      declare

         package Routers_Configuration is new Generic_Routers_ConfigurationTo API docTo spec (Command_Line_Parameters);
         package Message_Structures    is new Generic_Message_StructuresTo API docTo spec    (Routers_Configuration);
         package Router                is new Generic_RouterTo API docTo specTo body               (Message_Structures);
         package Routers               is new Generic_RoutersTo API docTo specTo body               (Router);

         use Routers_Configuration;
         use Message_Structures;
         use Routers;

         package Random_Router         is new Discrete_Random (Router_RangeTo API docTo spec);
         use Random_Router;

         use Message_StringsTo API docTo spec;

         Router_Generator : Generator;

         type Distances_Map is array (Router_RangeTo API docTo spec, Router_RangeTo API docTo spec) of Natural;

         procedure Print_Connections is
```

```

begin
   New_Line;
   Put ("    ");
   for i in Router_RangeTo API docTo spec loop
      Put (Integer (i), 3);
   end loop;
   New_Line;
   Put ("    +");
   for i in Router_RangeTo API docTo spec loop
      Put ("---");
   end loop;
   Put (&apos;+&apos;);
   New_Line;
   for i in Router_RangeTo API docTo spec loop
      Put (Integer (i), 3);
      Put (" |");
      for j in Router_RangeTo API docTo spec loop
         if i = j then
            Put (" . ");
         elsif Nodes_ConnectedTo API docTo spec (Connection_Topology, Positive (i), Positive (j)) then
            if Router_ActiveTo API docTo spec (i) and then Router_ActiveTo API docTo spec (j) then
               if Nodes_ConnectedTo API docTo spec (Connection_Topology, Positive (j), Positive (i)) then
                  Put ("<->");
               else
                  Put (" ->");
               end if;
            else
               Put (" x ");
            end if;
         else
            Put ("   ");
         end if;
      end loop;
      Put (&apos;|&apos;);
      New_Line;
   end loop;
   Put ("    +");
   for i in Router_RangeTo API docTo spec loop
      Put ("---");
   end loop;
   Put (&apos;+&apos;);
   New_Line;
end Print_Connections;

procedure Print_Distance_Map (Map : Distances_Map) is

begin
   New_Line;
   Put ("     ");
   for i in Router_RangeTo API docTo spec loop
      Put (Integer (i), 3);
   end loop;
   New_Line;
   Put ("    +");
   for i in Router_RangeTo API docTo spec loop
      Put ("---");
   end loop;
   Put (&apos;+&apos;);
   New_Line;
   for i in Router_RangeTo API docTo spec loop
```

```
            Put (Integer (i), 3);
            Put (" |");
            for j in Router_RangeTo API docTo spec loop
               if i = j then
                  Put ("  .");
               elsif Map (i, j) = 1 then
                  Put ("   ");
               elsif Router_ActiveTo API docTo spec (i) and then Router_ActiveTo API docTo spec (j) then
                  Put (Map (i, j), 3);
               else
                  Put ("  x");
               end if;
            end loop;
            Put (&apos;|&apos;);
            New_Line;
         end loop;
         Put ("    +");
         for i in Router_RangeTo API docTo spec loop
            Put ("---");
         end loop;
         Put (&apos;+&apos;);
         New_Line;
      end Print_Distance_Map;

   begin
      if Routers_ConfiguredTo API docTo spec then

         Put_Line ("  => Routers up and running ");
         Put_Line ("-------------------------------- Waiting ---------------------------------------");
         Put ("  Time for routers to establish their strategies : "); Put (Float (Command_Line_Parameters.Routers_Settle_Time), 2, 2, 0); Put ("
second(s)"); New_Line;

         delay Command_Line_Parameters.Routers_Settle_Time; -- let the routers establish their strategies first

         if Command_Line_Parameters.Dropouts > 0 then
            Reset (Router_Generator);
            for Id in 1 .. Command_Line_Parameters.Dropouts loop
               loop
                  declare
                     Candidate : constant Router_RangeTo API docTo spec := Random (Router_Generator);
                  begin
                     if Router_ActiveTo API docTo spec (Candidate) then
                        Router_ShutdownTo API docTo specTo body (Candidate);
                        Put ("   -> Router"); Put (Integer (Candidate), 3); Put_Line (" dropped service");
                        exit;
                     end if;
                  end;
               end loop;
            end loop;
            Put (Command_Line_Parameters.Dropouts); Put_Line (" routers in total dropped out.");
         end if;

         Put_Line ("------------------------------ Measurements ------------------------------------");

         declare
            Sum_Hops                : Natural       := 0;
            Min_Hops                : Natural       := Natural&apos;Last;
            Max_Hops                : Natural       := Natural&apos;First;
```

```
   Distance_Map            : Distances_Map := (others => (others => Natural&apos;Last));
   Measurements_Successful : Boolean       := True;

   function Send_Probe (Sender, Receiver : Router_RangeTo API docTo spec) return Boolean is

   begin
      select
         Router_TasksTo API docTo spec (Sender).Send_MessageTo API docTo spec ((Destination => Receiver,
                                                                              The_Message => To_Bounded_String (" - The quick brown fox jumps over the lazy dog - ")));
         return True;
      or
         delay Command_Line_Parameters.Comms_Timeout;
         Put_Line ("Error: Unresponsive router found : " & Router_RangeTo API docTo spec&apos;Image (Sender) & " (does not respond to Send_Message)");
         Put_Line ("   -> Measurements aborted");
         return False;
      end select;
   end Send_Probe;

   function Receive_Probe (Sender, Receiver : Router_RangeTo API docTo spec) return Boolean is

      Mailbox_Message : Messages_MailboxTo API docTo spec;

   begin
      select
         Router_TasksTo API docTo spec (Receiver).Receive_MessageTo API docTo spec (Mailbox_Message);
         Distance_Map (Mailbox_Message.Sender, Receiver) := Mailbox_Message.Hop_Counter;
         Sum_Hops := Sum_Hops + Mailbox_Message.Hop_Counter;
```

```
            Min_Hops := Natural&apos;Min (Min_Hops, Mailbox_Message.Hop_Counter);
            Max_Hops := Natural&apos;Max (Max_Hops, Mailbox_Message.Hop_Counter);
            return True;
         or
            delay Command_Line_Parameters.Comms_Timeout;
            Put_Line ("Error: Message not received in time : from router" & Router_RangeTo API docTo spec&apos;Image (Sender) & " to router" & Router_RangeTo API docTo spec&apos;Image (Receiver));
            Put_Line ("   -> Measurements aborted");
            return False;
         end select;
      end Receive_Probe;

   begin
      Main_Measurement : for i in Router_RangeTo API docTo spec loop
         for j in Router_RangeTo API docTo spec loop
            if i /= j and then Router_ActiveTo API docTo spec (i) and then Router_ActiveTo API docTo spec (j) then
               case Command_Line_Parameters.Test_Mode is
                  when One_To_All => Measurements_Successful := Send_Probe (i, j);
                  when All_to_One => Measurements_Successful := Send_Probe (j, i);
               end case;
               if not Measurements_Successful then
                  exit Main_Measurement;
               end if;
            end if;
         end loop;
         for j in Router_RangeTo API docTo spec loop
            if i /= j and then Router_ActiveTo API docTo spec (i) and then Router_ActiveTo API docTo spec (j) then
               case Command_Line_Parameters.Test_Mode is
```

```
                  when One_To_All => Measurements_Successful := Receive_Probe (i, j);
                  when All_to_One => Measurements_Successful := Receive_Probe (j, i);
               end case;
               if not Measurements_Successful then
                  exit Main_Measurement;
               end if;
            end if;
         end loop;
      end loop Main_Measurement;

      if Measurements_Successful then
         declare
            Avg_Hops : constant Float := Float (Sum_Hops) / Float (((Router_RangeTo API docTo spec&apos;Last ** 2) - Router_RangeTo API docTo spec&apos;Last));
         begin
            Put ("Minimal hops : "); Put (Min_Hops, 3); New_Line;
            Put ("Maximal hops : "); Put (Max_Hops, 3); New_Line;
            Put      ("Average hops : "); Put (Avg_Hops, 3, 2, 0); New_Line;
            for i in Router_RangeTo API docTo spec loop
               for j in Router_RangeTo API docTo spec&apos;First .. i loop
                  if Distance_Map (i, j) /= Distance_Map (j, i) then
                     Put_Line ("Warning: unsymmetrical distances " & "(" & Router_RangeTo API docTo spec&apos;Image (i) & "->" & Router_RangeTo API docTo spec&apos;Image (j) & "):" & Natural&apos;Image (Distance_Map (i, j))
                               & " while " & "(" & Router_RangeTo API docTo spec&apos;Image (j) & "->" & Router_RangeTo API docTo spec&apos;Image (i) & "):" & Natural&apos;Image (Distance_Map (j, i)));
                  end if;
               end loop;
            end loop;

```

```
                  if Command_Line_Parameters.Print_Distances then
                     Print_Distance_Map (Distance_Map);
                  end if;
               end;
            end if;
         end;
         New_Line;

      else
         Put_Line ("  => Routers did not respond to configuration call -> no measurements performed");
      end if;

      Put_Line ("--------------- Information about the selected network topology ---------------");
      Put_Line ("  Topology                    : " & Preconfigured_TopologiesTo API docTo spec&apos;Image (Command_Line_Parameters.Selected_Topology));
      case Command_Line_Parameters.Selected_Topology is
         when Line                  => Put ("  Size                     : "); Put (Command_Line_Parameters.Size,      4); New_Line;
         when Ring                  => Put ("  Size                     : "); Put (Command_Line_Parameters.Size,      4); New_Line;
         when Star                  => Put ("  Size                     : "); Put (Command_Line_Parameters.Size,      4); New_Line;
         when Fully_Connected       => Put ("  Size                     : "); Put (Command_Line_Parameters.Size,      4); New_Line;
         when Tree                  => Put ("  Degree                   : "); Put (Command_Line_Parameters.Degree,    4); New_Line;
            Put ("  Depths                      : "); Put (Command_Line_Parameters.Depths,    4); New_Line;
         when Mesh                  => Put ("  Dimension                : "); Put (Command_Line_Parameters.Dimension, 4); New_Line;
            Put ("  Size                        : "); Put (Command_Line_Parameters.Size,      4); New_Line;
         when Torus                 => Put ("  Dimension                : "); Put (Command_Line_Parameters.Dimension, 4); New_Line;
            Put ("  Size                        : "); Put (Command_Line_Parameters.Size,      4); New_Line;
            when Hypercube             => Put ("  Dimension : "); Put (Command_Line_Parameters.Dimension, 4); New_Line;
            when Cube_Connected_Cycles => Put ("  Dimension : "); Put (Command_Line_Parameters.Dimension, 4); New_Line;
            when Butterfly              => Put ("  Dimension : "); Put (Command_Line_Parameters.Dimension, 4); New_Line;
            when Wrap_Around_Butterfly  => Put ("  Dimension : "); Put (Command_Line_Parameters.Dimension, 4); New_Line;
         end case;
         Put    ("  Number of nodes in topology : "); Put (Nodes_in_TopologyTo API docTo spec (Connection_Topology), 4); New_Line;
         if Min_DegreeTo API docTo specTo body (Connection_Topology) = Max_DegreeTo API docTo specTo body (Connection_Topology) then
            Put ("  Constant connection degree  : ");  Put (Min_DegreeTo API docTo specTo body (Connection_Topology), 4); New_Line;
         else
            Put ("  Minimal connection degree   : "); Put (Min_DegreeTo API docTo specTo body (Connection_Topology), 4); New_Line;
            Put ("  Maximal connection degree   : "); Put (Max_DegreeTo API docTo specTo body (Connection_Topology), 4); New_Line;
         end if;
         if Command_Line_Parameters.Print_Connections then
            Print_Connections;
         end if;
         New_Line;

         Global_ShutdownTo API docTo specTo body;

      end;
   end if;

end Test_RoutersTo specTo body;
```