Documentation


```
1. --
2. --  Uwe R. Zimmer, Australia, September 2016
3. --
4.
5. with Ada.Float_Text_IO;                use Ada.Float_Text_IO;
6. with Ada.Integer_Text_IO;              use Ada.Integer_Text_IO;
7. with Ada.Numerics.Discrete_Random;     use Ada.Numerics;
8. with Ada.Text_IO;                      use Ada.Text_IO;
9. with Generic_Message_StructuresTo API docTo spec;
10.with Generic_RouterTo API docTo specTo body;
11.with Generi
12.with Generi
13.with GNAT.Command_Line;                                  Line;
14.with Routers_Configuration_Structures                    body; use
   Routers_Configuration_StructuresTo API docTo specTo body;
15.with TopologiesTo API docTo specTo body;                 use
   TopologiesTo API docTo specTo body;
16.
17.procedure Test_Routers is
18.
19.   Command_Line_Parameters : Command_Line_OptionsTo API docTo spec;
20.   Options_Ok              : Boolean               := True;
21.
22.   procedure Print_Options is
23.
24.   begin
25.      New_Line; Put ("accepted options:");
```

```
26.      New_Line; Put ("   [-t {Topology           : String   }] -> "); Put
  (Preconfigured_TopologiesTo API docTo spec&apos;Image
  (Command_Line_Parameters.Selected_Topology));

27.      New_Line; Put ("      by Size            : Line, Ring, Star,
  Fully_Connected");

28.      New_Line; Put ("      by Degree, Depths  : Tree");

29.      New_Line; Put ("      by Dimension, Size : Mesh, Torus");

30.      New_Line; Put ("      by Dimension       : Hypercube,
  Cube_Connected_Cycles,");

31.      New_Line; Put ("                           Butterfly,
  Wrap_Around_Butterfly");

32.      New_Line; Put ("   [-s {Size               : Positive }] -> "); Put
  (Command_Line_Parameters.Size, 3);

33.      New_Line; Put ("   [-g {Degree             : Positive }] -> "); Put
  (Command_Line_Parameters.Degree, 3);

34.      New_Line; Put ("   [-p {Depths             : Positive }] -> "); Put
  (Command_Line_Parameters.Depths, 3);

35.      New_Line; Put ("   [-d {Dimension          : Positive }] -> "); Put
  (Command_Line_Parameters.Dimension, 3);

); Put   36.      New_L
           (Boolean&apo

); Put   37.      New_L
           (Boolean&apos;Image (Command_Line_Para                ces));

38.      New_Line; Put ("   [-w {Routers                nds  }] -> "); Put
  (Float (Command_Line_Parameters.Router                2, 0);

39.      New_Line; Put ("   [-o {Comms timeout      : Seconds  }] -> "); Put
  (Float (Command_Line_Parameters.Comms_Timeout), 2, 2, 0);

40.      New_Line; Put ("   [-m {Test mode          : String   }] -> "); Put
  (Test_ModesTo API docTo spec&apos;Image (Command_Line_Parameters.Test_Mode));

41.      New_Line; Put ("      Available modes: One_to_All, All_to_One");

42.      New_Line; Put ("   [-x {Dropouts           : Natural  }] -> "); Put
  (Command_Line_Parameters.Dropouts, 3);

43.      New_Line;

44.      New_Line;

45.   end Print_OptionsTo specTo body;

46.

47.begin

48.   Initialize_Option_Scan;

49.   loop
```

```
50.      declare

51.         Option : constant Character := Getopt ("t: s: g: p: d: c: i: w: o: m:
  x:");

52.      begin

53.         case Option is

54.            when ASCII.NUL => exit;

55.            when &apos;t&apos; =>
  Command_Line_Parameters.Selected_Topology   := Preconfigured_TopologiesTo API
  docTo spec&apos;Value (Parameter);

56.            when &apos;s&apos; =>
  Command_Line_Parameters.Size                := Positive&apos;Value (Parameter);

57.            when &apos;g&apos; =>
  Command_Line_Parameters.Degree              := Positive&apos;Value (Parameter);

58.            when &apos;p&apos; =>
  Command_Line_Parameters.Depths              := Positive&apos;Value (Parameter);

59.            when &apos;d&apos; =>
  Command_Line_Parameters.Dimension           := Positive&apos;Value (Parameter);

60.            when &apos;c&apos; =>
  Command_Line_Parameters.Print_Connections   := Boolean&apos;Value (Parameter);

61.
(Parameter);   Command_Line

62.            when &apos;w&apos; =>
  Command_Line_Parameters.Routers_Settle                pos;Value (Parameter);

63.            when &apos;o&apos; =>
  Command_Line_Parameters.Comms_Timeout       := Duration&apos;Value (Parameter);

64.            when &apos;m&apos; =>
  Command_Line_Parameters.Test_Mode           := Test_ModesTo API docTo
  spec&apos;Value (Parameter);

65.            when &apos;x&apos; =>
  Command_Line_Parameters.Dropouts            := Natural&apos;Value (Parameter);

66.            when others => raise Program_Error;

67.         end case;

68.      exception

69.         when others =>

70.            New_Line; Put ("---> Error in option -"); Put (Option); New_Line;

71.            Options_Ok := False;

72.      end;

73.   end loop;

74.
```

```
75.   Print_OptionsTo specTo body;

76.

77.   if Options_Ok then

78.

79.      New_Line;

80.      Put_Line ("----------------------- Instantiating router tasks
   -----------------------------");

81.

82.      declare

83.

84.         package Routers_Configuration is new Generic_Routers_ConfigurationTo
   API docTo spec (Command_Line_Parameters);

85.         package Message_Structures    is new Generic_Message_StructuresTo API
   docTo spec    (Routers_Configuration);

86.         package Router                is new Generic_RouterTo API docTo
   specTo body   (Message_Structures);

87.         package Routers               is new Generic_RoutersTo API docTo
   specTo body

88.

89.         use Routers_Configuration;

90.         use Message_Structures;

91.         use Routers;

92.

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

94.         use Random_Router;

95.

96.         use Message_StringsTo API docTo spec;

97.

98.         Router_Generator : Generator;

99.

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

101.

102.         procedure Print_Connections is
```

```
103.
104.         begin
105.            New_Line;
106.            Put ("    ");
107.            for i in Router_RangeTo API docTo spec loop
108.               Put (Integer (i), 3);
109.            end loop;
110.            New_Line;
111.            Put ("    +");
112.            for i in Router_RangeTo API docTo spec loop
113.               Put ("---");
114.            end loop;
115.            Put (&apos;+&apos;);
116.            New_Line;
117.                              _____________                oop
118.
119.               Put (" |");
120.               for j in Router_Range                        oop
121.                  if i = j then
122.                     Put (" . ");
123.                  elsif Nodes_ConnectedTo API docTo spec
  (Connection_Topology, Positive (i), Positive (j)) then
124.                     if Router_ActiveTo API docTo spec (i) and then
  Router_ActiveTo API docTo spec (j) then
125.                        if Nodes_ConnectedTo API docTo spec
  (Connection_Topology, Positive (j), Positive (i)) then
126.                           Put ("<->");
127.                        else
128.                           Put (" ->");
129.                        end if;
130.                     else
131.                        Put (" x ");
132.                     end if;
```

```
133.                  else
134.                     Put ("   ");
135.                  end if;
136.               end loop;
137.               Put (&apos;|&apos;);
138.               New_Line;
139.            end loop;
140.            Put ("    +");
141.            for i in Router_RangeTo API docTo spec loop
142.               Put ("---");
143.            end loop;
144.            Put (&apos;+&apos;);
145.            New_Line;
146.         end Print_Connections;
147.
148.      p
149.
150.      begin
151.         New_Line;
152.         Put ("     ");
153.         for i in Router_RangeTo API docTo spec loop
154.            Put (Integer (i), 3);
155.         end loop;
156.         New_Line;
157.         Put ("    +");
158.         for i in Router_RangeTo API docTo spec loop
159.            Put ("---");
160.         end loop;
161.         Put (&apos;+&apos;);
162.         New_Line;
163.         for i in Router_RangeTo API docTo spec loop
```

```
164.               Put (Integer (i), 3);
165.               Put (" |");
166.               for j in Router_RangeTo API docTo spec loop
167.                  if i = j then
168.                     Put ("  .");
169.                  elsif Map (i, j) = 1 then
170.                     Put ("   ");
171.                  elsif Router_ActiveTo API docTo spec (i) and then
  Router_ActiveTo API docTo spec (j) then
172.                     Put (Map (i, j), 3);
173.                  else
174.                     Put ("  x");
175.                  end if;
176.               end loop;
177.               Put (&apos;|&apos;);
178.
179.
180.            Put ("    +");
181.            for i in Router_RangeTo
182.               Put ("---");
183.            end loop;
184.            Put (&apos;+&apos;);
185.            New_Line;
186.         end Print_Distance_Map;
187.
188.      begin
189.         if Routers_ConfiguredTo API docTo spec then
190.
191.            Put_Line ("  => Routers up and running ");
192.            Put_Line ("------------------------------- Waiting
  ---------------------------------------");
193.            Put ("  Time for routers to establish their strategies : "); Put
  (Float (Command_Line_Parameters.Routers_Settle_Time), 2, 2, 0); Put ("
```

```
   second(s)"); New_Line;

194.

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

196.

197.            if Command_Line_Parameters.Dropouts > 0 then

198.               Reset (Router_Generator);

199.               for Id in 1 .. Command_Line_Parameters.Dropouts loop

200.                  loop

201.                     declare

202.                        Candidate : constant Router_RangeTo API docTo spec :=
   Random (Router_Generator);

203.                     begin

204.                        if Router_ActiveTo API docTo spec (Candidate) then

205.                           Router_ShutdownTo API docTo specTo body
   (Candidate);

206.   (Integer (Candidate),
   3); Put_Line

207.                           exit;

208.                        end if;

209.                     end;

210.                  end loop;

211.               end loop;

212.               Put (Command_Line_Parameters.Dropouts); Put_Line (" routers in
   total dropped out.");

213.            end if;

214.

215.            Put_Line ("------------------------------ Measurements
   -----------------------------------");

216.

217.            declare

218.               Sum_Hops                : Natural       := 0;

219.               Min_Hops                : Natural       := Natural&amp;apos;Last;

220.               Max_Hops                : Natural       := Natural&amp;apos;First;
```

```
221.                Distance_Map            : Distances_Map := (others => (others
  => Natural&apos;Last));

222.                Measurements_Successful : Boolean       := True;

223.

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

225.

226.                begin

227.                   select

228.                      Router_TasksTo API docTo spec (Sender).Send_MessageTo
  API docTo spec ((Destination => Receiver,

229.                                                          The_Message =>
  To_Bounded_String (" - The quick brown fox jumps over the lazy dog - ")));

230.                      return True;

231.                   or

232.                      delay Command_Line_Parameters.Comms_Timeout;

233.                      Put_Line ("Error: Unresponsive router found : " &
  Router_Range'Image
  Send_Message

borted");234.

235.                      return False;

236.                   end select;

237.                end Send_Probe;

238.

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

240.

241.                   Mailbox_Message : Messages_MailboxTo API docTo spec;

242.

243.                begin

244.                   select

245.                      Router_TasksTo API docTo spec
  (Receiver).Receive_MessageTo API docTo spec (Mailbox_Message);

246.                      Distance_Map (Mailbox_Message.Sender, Receiver) :=
  Mailbox_Message.Hop_Counter;

247.                      Sum_Hops := Sum_Hops + Mailbox_Message.Hop_Counter;
```

```
248.                     Min_Hops := Natural&apos;Min (Min_Hops,
  Mailbox_Message.Hop_Counter);

249.                     Max_Hops := Natural&apos;Max (Max_Hops,
  Mailbox_Message.Hop_Counter);

250.                     return True;

251.                  or

252.                     delay Command_Line_Parameters.Comms_Timeout;

253.                     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));

254.                     Put_Line ("   -> Measurements aborted");

255.                     return False;

256.                  end select;

257.               end Receive_Probe;

258.

259.               begin

260.               Main_Measurement : for i in Router_RangeTo API docTo spec loop

261.                                                  spec      loop

262.                  if i /= j and then Router_ActiveTo API docTo spec (i)
  and then Router_ActiveTo API docTo spe

263.                                     case Command                st_Mode is

264.                                        when One_To_All => Measurements_Successful :=
  Send_Probe (i, j);

265.                                        when All_to_One => Measurements_Successful :=
  Send_Probe (j, i);

266.                                     end case;

267.                                     if not Measurements_Successful then

268.                                        exit Main_Measurement;

269.                                     end if;

270.                  end if;

271.               end loop;

272.               for j in Router_RangeTo API docTo spec loop

273.                  if i /= j and then Router_ActiveTo API docTo spec (i)
  and then Router_ActiveTo API docTo spec (j) then

274.                                     case Command_Line_Parameters.Test_Mode is
```

```
275.                            when One_To_All => Measurements_Successful :=
  Receive_Probe (i, j);

276.                            when All_to_One => Measurements_Successful :=
  Receive_Probe (j, i);

277.                         end case;

278.                         if not Measurements_Successful then

279.                            exit Main_Measurement;

280.                         end if;

281.                      end if;

282.                   end loop;

283.                end loop Main_Measurement;

284.

285.                if Measurements_Successful then

286.                   declare

287.                      Avg_Hops : constant Float := Float (Sum_Hops) / Float
  (((Router_RangeTo API docTo spec&apos;Last ** 2) - Router_RangeTo API docTo
  spec&apos;Last));

288.

289.                      Put ("Minimal hops : "); Put (Min_Hops, 3); New_Line;

290.                      Put ("Maximal hops : "); Put (Max_Hops, 3); New_Line;

291.                      Put      ("Average hops : "); Put (Avg_Hops, 3, 2, 0);
  New_Line;

292.                      for i in Router_RangeTo API docTo spec loop

293.                         for j in Router_RangeTo API docTo spec&apos;First ..
  i loop

294.                            if Distance_Map (i, j) /= Distance_Map (j, i) then

295.                               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))

296.                                         & " while " & "(" & Router_RangeTo
  API docTo spec&apos;Image (j) & "->" & Router_RangeTo API docTo spec&apos;Image
  (i) & "):" & Natural&apos;Image (Distance_Map (j, i)));

297.                            end if;

298.                         end loop;

299.                      end loop;

300.
```

```
301.                     if Command_Line_Parameters.Print_Distances then

302.                        Print_Distance_Map (Distance_Map);

303.                     end if;

304.                  end;

305.               end if;

306.            end;

307.            New_Line;

308.

309.         else

310.            Put_Line ("  => Routers did not respond to configuration call ->
  no measurements performed");

311.         end if;

312.

313.         Put_Line ("--------------- Information about the selected network
  topology ---------------");

314.         P
  Preconfigure
  (Command_Lin

315.         case Command_Line_Parameters.Selected_Topology is

316.            when Line                  => Put ("  Size
  : "); Put (Command_Line_Parameters.Siz                e);

317.            when Ring                  => Put ("  Size
  : "); Put (Command_Line_Parameters.Size,      4); New_Line;

318.            when Star                  => Put ("  Size
  : "); Put (Command_Line_Parameters.Size,      4); New_Line;

319.            when Fully_Connected       => Put ("  Size
  : "); Put (Command_Line_Parameters.Size,      4); New_Line;

320.            when Tree                  => Put ("  Degree
  : "); Put (Command_Line_Parameters.Degree,    4); New_Line;

321.               Put ("  Depths                        : "); Put
  (Command_Line_Parameters.Depths,    4); New_Line;

322.            when Mesh                  => Put ("  Dimension
  : "); Put (Command_Line_Parameters.Dimension, 4); New_Line;

323.               Put ("  Size                          : "); Put
  (Command_Line_Parameters.Size,      4); New_Line;

324.            when Torus                 => Put ("  Dimension
  : "); Put (Command_Line_Parameters.Dimension, 4); New_Line;

325.               Put ("  Size                          : "); Put
  (Command_Line_Parameters.Size,      4); New_Line;
```

```
326.            when Hypercube             => Put ("  Dimension
  : "); Put (Command_Line_Parameters.Dimension, 4); New_Line;

327.            when Cube_Connected_Cycles => Put ("  Dimension
  : "); Put (Command_Line_Parameters.Dimension, 4); New_Line;

328.            when Butterfly              => Put ("  Dimension
  : "); Put (Command_Line_Parameters.Dimension, 4); New_Line;

329.            when Wrap_Around_Butterfly  => Put ("  Dimension
  : "); Put (Command_Line_Parameters.Dimension, 4); New_Line;

330.         end case;

331.         Put    ("  Number of nodes in topology : "); Put
  (Nodes_in_TopologyTo API docTo spec (Connection_Topology), 4); New_Line;

332.         if Min_DegreeTo API docTo specTo body (Connection_Topology) =
  Max_DegreeTo API docTo specTo body (Connection_Topology) then

333.            Put ("  Constant connection degree  : ");  Put (Min_DegreeTo API
  docTo specTo body (Connection_Topology), 4); New_Line;

334.         else

335.            Put ("  Minimal connection degree   : "); Put (Min_DegreeTo API
  docTo specTo body (Connection_Topology), 4); New_Line;

336.            Put ("  Maximal connection degree   : "); Put (Max_DegreeTo API
  docTo specTo

337.         e

338.         if Command_Line_Parameters.                    hen

339.            Print_Connections;

340.         end if;

341.         New_Line;

342.

343.         Global_ShutdownTo API docTo specTo body;

344.

345.      end;

346.   end if;

347.

348.end Test_RoutersTo specTo body;
```