Documentation


```
1. --
2. --  Uwe R. Zimmer, Australia, September 2019
3. --
4.
5. package body TopologiesTo API docTo spec is
6.
7.    --  Basic topology parameters
8.
9.    type Topology_by_Size is abstract new Topology_KindTo API docTo spec with
   record
10.      Size : Positive;
11.   end record;
12.
13.   type Topology_by_Dimension is abstract new Topology_KindTo API docTo spec
   with record
14.      Dimension : Positive;
15.   end record;
16.
17.   type Topology_by_Dimension_and_Size is abstract new Topology_by_Dimension
   with record
18.      Size : Positive;
19.   end record;
20.
21.   type Topology_by_Degree is abstract new Topology_KindTo API docTo spec with
   record
22.      Degree : Positive;
23.   end record;
24.
```

```
25.   type Topology_by_Degree_and_Depths is abstract new Topology_by_Degree with
      record
26.      Depths : Positive;
27.   end record;
28.
29.   --  Cube_Connected_Cycles
30.
31.   type Topology_Cube_Connected_Cycles is new Topology_by_Dimension with null
      record;
32.
33.   overriding function Nodes_in_Topology (Configuration :
      Topology_Cube_Connected_Cycles) return Positive;
34.   overriding function Nodes_Connected   (Configuration :
      Topology_Cube_Connected_Cycles;
35.                                          Node_A, Node_B : Positive) return
      Boolean;
36.
37.   --  Trees
38.
39.   type Topology_Trees is new Topology_by_Degree_and_Depths with null record;
40.
41.   overriding function Nodes_in_Topology (Configuration : Topology_Trees)
      return Positive;
42.   overriding function Nodes_Connected   (Configuration : Topology_Trees;
43.                                          Node_A, Node_B : Positive) return
      Boolean;
44.
45.   --  Mesh
46.
47.   type Topology_Mesh is new Topology_by_Dimension_and_Size with null record;
48.
49.   overriding function Nodes_in_Topology (Configuration : Topology_Mesh)
      return Positive;
50.   overriding function Nodes_Connected   (Configuration : Topology_Mesh;
51.                                          Node_A, Node_B : Positive) return
      Boolean;
```

```
52.
53.    --  Torus
54.
55.    type Topology_Torus is new Topology_by_Dimension_and_Size with null record;
56.
57.    overriding function Nodes_in_Topology (Configuration : Topology_Torus)
   return Positive;
58.    overriding function Nodes_Connected   (Configuration : Topology_Torus;
59.                                           Node_A, Node_B : Positive) return
   Boolean;
60.
61.    --  Butterfly
62.
63.    type Topology_Butterfly is new Topology_by_Dimension with null record;
64.
65.    overriding function Nodes_in_Topology (Configuration : Topology_Butterfly)
   return Positive;
66.    overriding function Nodes_Connected   (Configuration : Topology_Butterfly;
67.                                           Node_A, Node_B : Positive) return
   Boolean;
68.
69.    --  Wrap_Around_Butterfly
70.
71.    type Topology_Wrap_Around_Butterfly is new Topology_by_Dimension with null
   record;
72.
73.    overriding function Nodes_in_Topology (Configuration :
   Topology_Wrap_Around_Butterfly) return Positive;
74.    overriding function Nodes_Connected   (Configuration :
   Topology_Wrap_Around_Butterfly;
75.                                           Node_A, Node_B : Positive) return
   Boolean;
76.
77.    --  Star
78.
```

```
79.   type Topology_Star is new Topology_by_Size with null record;
80.
81.   overriding function Nodes_in_Topology (Configuration : Topology_Star)
  return Positive;
82.   overriding function Nodes_Connected   (Configuration : Topology_Star;
83.                                          Node_A, Node_B : Positive) return
  Boolean;
84.
85.   --  Fully_Connected
86.
87.   type Topology_Fully_Connected is new Topology_by_Size with null record;
88.
89.   overriding function Nodes_in_Topology (Configuration :
  Topology_Fully_Connected) return Positive;
90.   overriding function Nodes_Connected   (Configuration :
  Topology_Fully_Connected;
91.                                          Node_A, Node_B : Positive) return
  Boolean;
92.
93.   --  Cube_Connected_Cycles
94.
95.   overriding function Nodes_in_Topology (Configuration :
  Topology_Cube_Connected_Cycles) return Positive is
96.
97.     (Configuration.Dimension * (2 ** (Configuration.Dimension)));
98.
99.   overriding function Nodes_Connected (Configuration  :
  Topology_Cube_Connected_Cycles;
100.                                        Node_A, Node_B : Positive) return
  Boolean is
101.
102.      subtype Corners is Natural range 0 .. (2 ** (Configuration.Dimension))
  - 1;
103.      subtype Cycles  is Natural range 0 .. Configuration.Dimension - 1;
104.
105.      type CCC_Coordinates is record
```

```
106.         Corner_Nr   : Corners;
107.         Cycle_Nr    : Cycles;
108.      end record;
109.
110.      function To_CCC_Coordinates (Node : Positive) return CCC_Coordinates is
111.
112.         Coordinate : constant CCC_Coordinates := (Corner_Nr => (Node - 1) /
  Configuration.Dimension,
113.                                                  Cycle_Nr  => (Node - 1)
  mod Configuration.Dimension);
114.
115.      begin
116.         return Coordinate;
117.      end To_CCC_Coordinates;
118.
119.      CCC_Node_A : constant CCC_Coordinates := To_CCC_Coordinates (Node_A);
120.      CCC_Node_B : constant CCC_Coordinates := To_CCC_Coordinates (Node_B);
121.
122.      type Bit_Arrays is array (Cycles) of Boolean;
123.
124.      function Bit_Array (Corner_Nr : Corners) return Bit_Arrays is
125.
126.         Bits : Bit_Arrays;
127.
128.      begin
129.         for Bit in Bits&apos;Range loop
130.            Bits (Bit) := (Corner_Nr / (2 ** Bit)) mod 2 > 0;
131.         end loop;
132.         return Bits;
133.      end Bit_Array;
134.
135.      function Invert_Bit (Bit_Nr : Cycles; Bits : Bit_Arrays) return
  Bit_Arrays is
```

```
136.
137.         Return_Bits : Bit_Arrays := Bits;
138.
139.      begin
140.         Return_Bits (Bit_Nr) := not Return_Bits (Bit_Nr);
141.         return Return_Bits;
142.      end Invert_Bit;
143.
144.   begin
145.      return (CCC_Node_A.Corner_Nr = CCC_Node_B.Corner_Nr
146.        and then (CCC_Node_A.Cycle_Nr = (CCC_Node_B.Cycle_Nr + 1) mod
  Configuration.Dimension
147.          or else CCC_Node_A.Cycle_Nr = (CCC_Node_B.Cycle_Nr - 1) mod
  Configuration.Dimension))
148.        or else (CCC_Node_A.Cycle_Nr = CCC_Node_B.Cycle_Nr
149.                 and then Bit_Array (CCC_Node_A.Corner_Nr) = Invert_Bit
  (CCC_Node_A.Cycle_Nr, Bit_Array (CCC_Node_B.Corner_Nr)));
150.   end Nodes_ConnectedTo specTo body;
151.
152.   --  Trees
153.
154.   overriding function Nodes_in_Topology (Configuration : Topology_Trees)
  return Positive is
155.
156.      Nodes : Positive := 1;
157.
158.   begin
159.      for Level in 1 .. Configuration.Depths - 1 loop
160.         Nodes := Nodes + (Configuration.Degree ** Level);
161.      end loop;
162.      return Nodes;
163.   end Nodes_in_TopologyTo specTo body;
164.
```

```
165.   overriding function Nodes_Connected (Configuration  : Topology_Trees;
166.                                        Node_A, Node_B : Positive) return
  Boolean is
167.
168.      Node_Nr : Positive := 1;
169.
170.      function Construct_Tree (Parent_Nr, Depth : Positive) return Boolean is
171.
172.      begin
173.         if Depth <= Configuration.Depths then
174.            for i in 1 .. Configuration.Degree loop
175.               Node_Nr := Node_Nr + 1;
176.               if (Parent_Nr = Node_A and then Node_Nr = Node_B)
177.                 or else (Parent_Nr = Node_B and then Node_Nr = Node_A)
178.               then
179.                  return True;
180.               else
181.                  if Construct_Tree (Node_Nr, Depth + 1) then
182.                     return True;
183.                  end if;
184.               end if;
185.            end loop;
186.            return False;
187.         else
188.            return False;
189.         end if;
190.      end Construct_Tree;
191.
192.   begin
193.      return Construct_Tree (Node_Nr, 2);
194.   end Nodes_ConnectedTo specTo body;
```

```
195.
196.    --  Mesh
197.
198.    overriding function Nodes_in_Topology (Configuration : Topology_Mesh)
   return Positive is
199.
200.       (Configuration.Size ** Configuration.Dimension);
201.
202.    overriding function Nodes_Connected (Configuration   : Topology_Mesh;
203.                                         Node_A, Node_B : Positive) return
   Boolean is
204.
205.       subtype Nodes_in_Line is Natural range 0 .. Configuration.Size - 1;
206.       type Coordinates is array (0 .. Configuration.Dimension - 1) of
   Nodes_in_Line;
207.
208.       function To_Coordinates (Node_Nr : Positive) return Coordinates is
209.
210.          Coordinate : Coordinates;
211.
212.       begin
213.          for Dim in 0 .. Coordinate&apos;Last loop
214.             Coordinate (Dim) := (Node_Nr - 1) / Configuration.Size ** Dim mod
   Configuration.Size;
215.          end loop;
216.          return Coordinate;
217.       end To_Coordinates;
218.
219.       Coordinate_A : constant Coordinates := To_Coordinates (Node_A);
220.       Coordinate_B : constant Coordinates := To_Coordinates (Node_B);
221.
222.       Matching_Coordinates : Natural := 0;
223.
```

```
224.   begin
225.      for Dim in Coordinates&apos;Range loop
226.         if Coordinate_A (Dim) = Coordinate_B (Dim) then
227.            Matching_Coordinates := Matching_Coordinates + 1;
228.         end if;
229.      end loop;
230.      if Matching_Coordinates = Configuration.Dimension - 1 then
231.         for Dim in Coordinates&apos;Range loop
232.            if      (Coordinate_A (Dim) < Nodes_in_Line&apos;Last and then
  Coordinate_A (Dim) + 1 = Coordinate_B (Dim))
233.              or else (Coordinate_B (Dim) < Nodes_in_Line&apos;Last and then
  Coordinate_B (Dim) + 1 = Coordinate_A (Dim))
234.            then
235.               return True;
236.            end if;
237.         end loop;
238.         return False;
239.      else
240.         return False;
241.      end if;
242.   end Nodes_ConnectedTo specTo body;
243.
244.   --  Torus
245.
246.   overriding function Nodes_in_Topology (Configuration : Topology_Torus)
  return Positive is
247.
248.     (Configuration.Size ** Configuration.Dimension);
249.
250.   overriding function Nodes_Connected (Configuration   : Topology_Torus;
251.                                        Node_A, Node_B : Positive) return
  Boolean is
252.
```

```
253.      subtype Nodes_in_Line is Natural range 0 .. Configuration.Size - 1;

254.      type Coordinates is array (0 .. Configuration.Dimension - 1) of
  Nodes_in_Line;

255.

256.      function To_Coordinates (Node_Nr : Positive) return Coordinates is

257.

258.         Coordinate : Coordinates;

259.

260.      begin

261.         for Dim in 0 .. Coordinate&apos;Last loop

262.            Coordinate (Dim) := (Node_Nr - 1) / Configuration.Size ** Dim mod
  Configuration.Size;

263.         end loop;

264.         return Coordinate;

265.      end To_Coordinates;

266.

267.      Coordinate_A : constant Coordinates := To_Coordinates (Node_A);

268.      Coordinate_B : constant Coordinates := To_Coordinates (Node_B);

269.

270.      Matching_Coordinates : Natural := 0;

271.

272.   begin

273.      for Dim in Coordinates&apos;Range loop

274.         if Coordinate_A (Dim) = Coordinate_B (Dim) then

275.            Matching_Coordinates := Matching_Coordinates + 1;

276.         end if;

277.      end loop;

278.      if Matching_Coordinates = Configuration.Dimension - 1 then

279.         for Dim in Coordinates&apos;Range loop

280.            if      (Coordinate_A (Dim) + 1) mod Configuration.Size =
  Coordinate_B (Dim)

281.              or else (Coordinate_B (Dim) + 1) mod Configuration.Size =
  Coordinate_A (Dim)
```

```
282.            then
283.               return True;
284.            end if;
285.         end loop;
286.         return False;
287.      else
288.         return False;
289.      end if;
290.   end Nodes_ConnectedTo specTo body;
291.
292.   --  Butterfly
293.
294.   overriding function Nodes_in_Topology (Configuration : Topology_Butterfly)
  return Positive is
295.
296.     ((Configuration.Dimension + 1) * (2 ** Configuration.Dimension));
297.
298.   overriding function Nodes_Connected (Configuration  : Topology_Butterfly;
299.                                        Node_A, Node_B : Positive) return
  Boolean is
300.
301.      subtype Lines  is Natural range 0 .. (2 ** (Configuration.Dimension)) -
  1;
302.      subtype Layers is Natural range 0 .. Configuration.Dimension;
303.      subtype Bits   is Natural range 0 .. Configuration.Dimension - 1;
304.
305.      type Butterfly_Coordinates is record
306.         Line  : Lines;
307.         Layer : Layers;
308.      end record;
309.
310.      function To_Butterfly_Coordinates (Node : Positive) return
  Butterfly_Coordinates is
```

```
311.
312.         Coordinate : constant Butterfly_Coordinates := (Line  => (Node -
  1) /   (Configuration.Dimension + 1),
313.                                                         Layer => (Node - 1)
  mod (Configuration.Dimension + 1));
314.
315.      begin
316.         return Coordinate;
317.      end To_Butterfly_Coordinates;
318.
319.      Butterfly_A : constant Butterfly_Coordinates :=
  To_Butterfly_Coordinates (Node_A);
320.      Butterfly_B : constant Butterfly_Coordinates :=
  To_Butterfly_Coordinates (Node_B);
321.
322.      type Bit_Arrays is array (Bits) of Boolean;
323.
324.      function To_Bit_Arrays (Line_Nr : Lines) return Bit_Arrays is
325.
326.         Bit_Array : Bit_Arrays;
327.
328.      begin
329.         for Bit in Bits&apos;Range loop
330.            Bit_Array (Bit) := (Line_Nr / (2 ** Bit)) mod 2 > 0;
331.         end loop;
332.         return Bit_Array;
333.      end To_Bit_Arrays;
334.
335.      function Invert_Bit (Bit_Nr : Bits; Bit_Array : Bit_Arrays) return
  Bit_Arrays is
336.
337.         Return_Bits : Bit_Arrays := Bit_Array;
338.
339.      begin
```

```
340.         Return_Bits (Bit_Nr) := not Return_Bits (Bit_Nr);

341.         return Return_Bits;

342.      end Invert_Bit;

343.

344.   begin

345.      return         ((Butterfly_A.Layer < Layers&apos;Last and then
  Butterfly_A.Layer + 1 = Butterfly_B.Layer)

346.                       or else (Butterfly_B.Layer < Layers&apos;Last and then
  Butterfly_B.Layer + 1 = Butterfly_A.Layer))

347.        and then           (Butterfly_A.Line = Butterfly_B.Line

348.                            or else ((Butterfly_A.Layer < Butterfly_B.Layer)

349.                                     and then To_Bit_Arrays
  (Butterfly_A.Line) = Invert_Bit (Butterfly_A.Layer, To_Bit_Arrays
  (Butterfly_B.Line)))

350.                            or else ((Butterfly_B.Layer < Butterfly_A.Layer)

351.                                     and then To_Bit_Arrays
  (Butterfly_B.Line) = Invert_Bit (Butterfly_B.Layer, To_Bit_Arrays
  (Butterfly_A.Line))));

352.   end Nodes_Connected_To_specTo_body;

353.

354.   --  Wrap_Around_Butterfly

355.

356.   overriding function Nodes_in_Topology (Configuration :
  Topology_Wrap_Around_Butterfly) return Positive is

357.

358.     (Configuration.Dimension * (2 ** Configuration.Dimension));

359.

360.   overriding function Nodes_Connected (Configuration   :
  Topology_Wrap_Around_Butterfly;

361.                                        Node_A, Node_B : Positive) return
  Boolean is

362.

363.      subtype Lines  is Natural range 0 .. (2 ** (Configuration.Dimension)) -
  1;

364.      subtype Layers is Natural range 0 .. Configuration.Dimension - 1;

365.      subtype Bits   is Natural range 0 .. Configuration.Dimension - 1;
```

```
366.
367.      type Butterfly_Coordinates is record
368.         Line  : Lines;
369.         Layer : Layers;
370.      end record;
371.
372.      function To_Butterfly_Coordinates (Node : Positive) return
  Butterfly_Coordinates is
373.
374.         Coordinate : constant Butterfly_Coordinates := (Line  => (Node -
  1) /   Configuration.Dimension,
375.                                                         Layer => (Node - 1)
  mod Configuration.Dimension);
376.
377.      begin
378.         return Coordinate;
379.      end To_Butterfly_Coordinates;
380.
381.      Butterfly_A : constant Butterfly_Coordinates :=
  To_Butterfly_Coordinates (Node_A);
382.      Butterfly_B : constant Butterfly_Coordinates :=
  To_Butterfly_Coordinates (Node_B);
383.
384.      type Bit_Arrays is array (Bits) of Boolean;
385.
386.      function To_Bit_Arrays (Line_Nr : Lines) return Bit_Arrays is
387.
388.         Bit_Array : Bit_Arrays;
389.
390.      begin
391.         for Bit in Bits&apos;Range loop
392.            Bit_Array (Bit) := (Line_Nr / (2 ** Bit)) mod 2 > 0;
393.         end loop;
394.         return Bit_Array;
```

```
395.      end To_Bit_Arrays;

396.

397.      function Invert_Bit (Bit_Nr : Bits; Bit_Array : Bit_Arrays) return
  Bit_Arrays is

398.

399.         Return_Bits : Bit_Arrays := Bit_Array;

400.

401.      begin

402.         Return_Bits (Bit_Nr) := not Return_Bits (Bit_Nr);

403.         return Return_Bits;

404.      end Invert_Bit;

405.

406.   begin

407.      return         ((Butterfly_A.Layer + 1) mod Configuration.Dimension =
  Butterfly_B.Layer

408.                      or else (Butterfly_B.Layer + 1) mod
  Configuration.Dimension = Butterfly_A.Layer)

409.        and then          (Butterfly_A.Line = Butterfly_B.Line

410.                         or else ((Butterfly_A.Layer + 1) mod
  Configuration.Dimension = Butterfly_B.Layer

411.                                  and then To_Bit_Arrays
  (Butterfly_A.Line) = Invert_Bit (Butterfly_A.Layer, To_Bit_Arrays
  (Butterfly_B.Line)))

412.                            or else ((Butterfly_B.Layer + 1) mod
  Configuration.Dimension = Butterfly_A.Layer

413.                                  and then To_Bit_Arrays
  (Butterfly_B.Line) = Invert_Bit (Butterfly_B.Layer, To_Bit_Arrays
  (Butterfly_A.Line))));

414.   end Nodes_ConnectedTo specTo body;

415.

416.   --  Star

417.

418.   overriding function Nodes_in_Topology (Configuration : Topology_Star)
  return Positive is

419.

420.     (Configuration.Size);
```

```
421.
422.   overriding function Nodes_Connected (Configuration  : Topology_Star;
423.                                        Node_A, Node_B : Positive) return
  Boolean is
424.
425.     (Node_A = 1 or else Node_B = 1);
426.
427.   --  Fully connected
428.
429.   overriding function Nodes_in_Topology (Configuration :
  Topology_Fully_Connected) return Positive is
430.
431.     (Configuration.Size);
432.
433.   overriding function Nodes_Connected (Configuration :
  Topology_Fully_Connected;
434.                                        Node_A, Node_B : Positive) return
  Boolean is
435.
436.     (True);
437.
438.   --
439.   --  Degrees
440.   --
441.
442.   function Min_DegreeTo API docTo spec (ConfigurationTo API docTo spec :
  Topology_KindTo API docTo spec&apos;Class) return Natural is
443.
444.      subtype Nodes_Range is Positive range 1 .. Nodes_in_TopologyTo API
  docTo spec (ConfigurationTo API docTo spec);
445.
446.      Min : Natural := Nodes_Range&apos;Last;
447.
448.   begin
```

```
449.      for i in Nodes_Range loop
450.         declare
451.            Degree : Natural := 0;
452.         begin
453.            for j in Nodes_Range loop
454.               if Nodes_ConnectedTo API docTo spec (ConfigurationTo API docTo
  spec, i, j) then
455.                  Degree := Degree + 1;
456.               end if;
457.            end loop;
458.            Min := Natural&apos;Min (Min, Degree);
459.         end;
460.      end loop;
461.      return Min;
462.   end Min_DegreeTo API docTo specTo body;
463.
464.   --
465.
466.   function Max_DegreeTo API docTo spec (ConfigurationTo API docTo spec :
  Topology_KindTo API docTo spec&apos;Class) return Natural is
467.
468.      subtype Nodes_Range is Positive range 1 .. Nodes_in_TopologyTo API
  docTo spec (ConfigurationTo API docTo spec);
469.
470.      Max : Natural := 0;
471.
472.   begin
473.      for i in Nodes_Range loop
474.         declare
475.            Degree : Natural := 0;
476.         begin
477.            for j in Nodes_Range loop
478.               if Nodes_ConnectedTo API docTo spec (ConfigurationTo API docTo
```

```
   spec, i, j) then
479.                   Degree := Degree + 1;
480.                end if;
481.             end loop;
482.             Max := Natural&apos;Max (Max, Degree);
483.          end;
484.       end loop;
485.       return Max;
486.    end Max_DegreeTo API docTo specTo body;
487.
488.    --
489.    --  Constructors
490.    --
491.
492.    function LineTo API docTo spec (Size : Positive)
   return Topology_KindTo API docTo spec&apos;Class is (Topology_Mesh&apos;
   (Dimension => 1, Size => SizeTo API docTo spec));
493.    function RingTo API docTo spec (Size : Positive)
   return Topology_KindTo API docTo spec&apos;Class is (Topology_Torus&apos;
   (Dimension => 1, Size => SizeTo API docTo spec));
494.    function StarTo API docTo spec (Size : Positive)
   return Topology_KindTo API docTo spec&apos;Class is (Topology_Star&apos;(Size
   => SizeTo API docTo spec));
495.    function Fully_ConnectedTo API docTo spec (Size : Positive)
   return Topology_KindTo API docTo spec&apos;Class is
   (Topology_Fully_Connected&apos;(Size => SizeTo API docTo spec));
496.    function TreesTo API docTo spec (Degree, Depths : Positive)
   return Topology_KindTo API docTo spec&apos;Class is (Topology_Trees&apos;
   (Degree => DegreeTo API docTo spec, Depths => DepthsTo API docTo spec));
497.    function MeshTo API docTo spec (Dimension, Size : Positive)
   return Topology_KindTo API docTo spec&apos;Class is (Topology_Mesh&apos;
   (Dimension => DimensionTo API docTo spec, Size => SizeTo API docTo spec));
498.    function TorusTo API docTo spec (Dimension, Size : Positive)
   return Topology_KindTo API docTo spec&apos;Class is (Topology_Torus&apos;
   (Dimension => DimensionTo API docTo spec, Size => SizeTo API docTo spec));
499.    function HypercubeTo API docTo spec (Dimension : Positive)
   return Topology_KindTo API docTo spec&apos;Class is (Topology_Torus&apos;
   (Dimension => DimensionTo API docTo spec, Size => 2));
500.    function Cube_Connected_CyclesTo API docTo spec (Dimension : Positive)
   return Topology_KindTo API docTo spec&apos;Class is
   (Topology_Cube_Connected_Cycles&apos;(Dimension => DimensionTo API docTo
```

```
   spec));

501.   function ButterflyTo API docTo spec (Dimension : Positive)
   return Topology_KindTo API docTo spec&apos;Class is (Topology_Butterfly&apos;
   (Dimension => DimensionTo API docTo spec));

502.   function Wrap_Around_ButterflyTo API docTo spec (Dimension : Positive)
   return Topology_KindTo API docTo spec&apos;Class is
   (Topology_Wrap_Around_Butterfly&apos;(Dimension => DimensionTo API docTo
   spec));

503.

504.end TopologiesTo API docTo specTo body;
```