Documentation

Table of Contents
Class Inheritance Tree

```
--
--  Uwe R. Zimmer, Australia, September 2019
--

package body TopologiesTo API docTo spec is

   --  Basic topology parameters

   type Topology_by_Size is abstract new Topology_KindTo API docTo spec with record
      Size
   end record;

   type Topology_by_Dimension is abstract new Topology_KindTo API docTo spec with record
      Dimension : Positive;
   end record;

   type Topology_by_Dimension_and_Size is abstract new Topology_by_Dimension with record
      Size : Positive;
   end record;

   type Topology_by_Degree is abstract new Topology_KindTo API docTo spec with record
      Degree : Positive;
   end record;

   type Topology_by_Degree_and_Depths is abstract new Topology_by_Degree with record
      Depths : Positive;
   end record;

   --  Cube_Connected_Cycles

   type Topology_Cube_Connected_Cycles is new Topology_by_Dimension with null record;

   overriding function Nodes_in_Topology (Configuration : Topology_Cube_Connected_Cycles) return Positive;
   overriding function Nodes_Connected   (Configuration : Topology_Cube_Connected_Cycles;
                                          Node_A, Node_B : Positive) return Boolean;

   --  Tree

   type Topology_Trees is new Topolog[illegible]ths with null record;

   overriding function Nodes_in_Topology (Configuration : Topology_Trees) return Positive;
   overriding function Nodes_Connected   (Configuration : Topology_Trees;
                                          Node_A, Node_B : Positive) return Boolean;

   --  Mesh

   type Topology_Mesh is new Topology_by_Dimension_and_Size with null record;

   overriding function Nodes_in_Topology (Configuration : Topology_Mesh) return Positive;
   overriding function Nodes_Connected   (Configuration : Topology_Mesh;
                                          Node_A, Node_B : Positive) return Boolean;
   --  Torus

   type Topology_Torus is new Topology_by_Dimension_and_Size with null record;

   overriding function Nodes_in_Topology (Configuration : Topology_Torus)
return Positive;
   overriding function Nodes_Connected   (Configuration : Topology_Torus;
                                          Node_A, Node_B : Positive) return
Boolean;

   --  Butterfly

   type Topology_Butterfly is new Topology_by_Dimension with null record;

   overridi                on :                            Topology_Butterfly)
return Posit
   overridi                  on                          : Topology_Butterfly;
                                                         : Positive) return
Boolean;

   --  Wrap_Around_Butterfly

   type Topology_Wrap_Around_Butterfly is new Topology_by_Dimension with null
record;

   overriding function Nodes_in_Topology (Configuration :
Topology_Wrap_Around_Butterfly) return Positive;
   overriding function Nodes_Connected   (Configuration :
Topology_Wrap_Around_Butterfly;
                                          Node_A, Node_B : Positive) return
Boolean;

   --  Star

```

```
type Topology_Star is new Topology_by_Size with null record;

overriding function Nodes_in_Topology (Configuration : Topology_Star) return Positive;
overriding function Nodes_Connected   (Configuration : Topology_Star;
                                       Node_A, Node_B : Positive) return Boolean;

--  Fully_Connected

type Topology_Fully_Connected is new Topology_by_Size with null record;

overriding function Nodes_in_Topology (Configuration : Topology_Fully_Connected) return Positive;
overriding function Nodes_Connected   (Configuration : Topology_Fully_Connected;
                                       Node_A, Node_B : Positive) return Boolean;

--  Cube_Connected_Cycles

overriding function Nodes_in_Topology (Configuration : Topology_Cube_Connected_Cycles) return Positive is

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

overriding function Nodes_Connected (Configuration  : Topology_Cube_Connected_Cycles;
                                      Node_A, Node_B : Positive) return Boolean is

    subtype Corners is Natural range 0 .. (2 ** (Configuration.Dimension)) - 1;
    subtype Cycles  is Natural range 0 .. Configuration.Dimension - 1;

    type CCC_Coordinates is record
```

```
      Corner_Nr   : Corners;
      Cycle_Nr    : Cycles;
   end record;

   function To_CCC_Coordinates (Node : Positive) return CCC_Coordinates is

      Coordinate : constant CCC_Coordinates := (Corner_Nr => (Node - 1) / Configuration.Dimension,
                                                 Cycle_Nr  => (Node - 1) mod Configuration.Dimension);

   begin
      return Coordinate;
   end To_CCC_Coordinates;
```

```
   CCC_                                                   Node_A);
   CCC_                                                   Node_B);
```

```
   type Bit_Arrays is array (Cycl

   function Bit_Array (Corner_Nr : Corners) return Bit_Arrays is

      Bits : Bit_Arrays;

   begin
      for Bit in Bits&apos;Range loop
         Bits (Bit) := (Corner_Nr / (2 ** Bit)) mod 2 > 0;
      end loop;
      return Bits;
   end Bit_Array;

   function Invert_Bit (Bit_Nr : Cycles; Bits : Bit_Arrays) return Bit_Arrays is
```

```

        Return_Bits : Bit_Arrays := Bits;

     begin
        Return_Bits (Bit_Nr) := not Return_Bits (Bit_Nr);
        return Return_Bits;
     end Invert_Bit;

  begin
     return (CCC_Node_A.Corner_Nr = CCC_Node_B.Corner_Nr
       and then (CCC_Node_A.Cycle_Nr = (CCC_Node_B.Cycle_Nr + 1) mod
Configuration.Dimension
         or else CCC_Node_A.Cycle_Nr = (CCC_Node_B.Cycle_Nr - 1) mod
Configuration.Dimension))
       or else (CCC_Node_A.Cycle_Nr = CCC_Node_B.Cycle_Nr
r_Nr) = Invert_Bit
(CCC_Node_A.
  end Nod

  --  Trees

  overriding function Nodes_in_Topology (Configuration : Topology_Trees)
return Positive is

     Nodes : Positive := 1;

  begin
     for Level in 1 .. Configuration.Depths - 1 loop
        Nodes := Nodes + (Configuration.Degree ** Level);
     end loop;
     return Nodes;
  end Nodes_in_TopologyTo specTo body;

overriding function Nodes_Connected (Configuration  : Topology_Trees;
                                     Node_A, Node_B : Positive) return
Boolean is

   Node_Nr : Positive := 1;

   function Construct_Tree (Parent_Nr, Depth : Positive) return Boolean is

   begin
      if Depth <= Configuration.Depths then
         for i in 1 .. Configuration.Degree loop
            Node_Nr := Node_Nr + 1;
            if (Parent_Nr = Node_A and then Node_Nr = Node_B)
              or else (Parent_Nr = Node_B and then Node_Nr = Node_A)
            then


               if Construct_Tree                  ) then
                  return True;
               end if;
            end if;
         end loop;
         return False;
      else
         return False;
      end if;
   end Construct_Tree;

begin
   return Construct_Tree (Node_Nr, 2);
end Nodes_ConnectedTo specTo body;
```

```

   --  Mesh

   overriding function Nodes_in_Topology (Configuration : Topology_Mesh) return Positive is

     (Configuration.Size ** Configuration.Dimension);

   overriding function Nodes_Connected (Configuration   : Topology_Mesh;
                                        Node_A, Node_B : Positive) return Boolean is

      subtype Nodes_in_Line is Natural range 0 .. Configuration.Size - 1;
      type Coordinates is array (0 .. Configuration.Dimension - 1) of Nodes_in_Line;

es is func


         Coordinate : Coordinates;


      begin
         for Dim in 0 .. Coordinate&apos;Last loop
            Coordinate (Dim) := (Node_Nr - 1) / Configuration.Size ** Dim mod Configuration.Size;
         end loop;
         return Coordinate;
      end To_Coordinates;

      Coordinate_A : constant Coordinates := To_Coordinates (Node_A);
      Coordinate_B : constant Coordinates := To_Coordinates (Node_B);

      Matching_Coordinates : Natural := 0;

```

```
begin
   for Dim in Coordinates&apos;Range loop
      if Coordinate_A (Dim) = Coordinate_B (Dim) then
         Matching_Coordinates := Matching_Coordinates + 1;
      end if;
   end loop;
   if Matching_Coordinates = Configuration.Dimension - 1 then
      for Dim in Coordinates&apos;Range loop
         if      (Coordinate_A (Dim) < Nodes_in_Line&apos;Last and then Coordinate_A (Dim) + 1 = Coordinate_B (Dim))
           or else (Coordinate_B (Dim) < Nodes_in_Line&apos;Last and then Coordinate_B (Dim) + 1 = Coordinate_A (Dim))
         then
            return True;
         end if;
      e
      r
   else
      return False;
   end if;
end Nodes_ConnectedTo specTo body;

--  Torus

overriding function Nodes_in_Topology (Configuration : Topology_Torus) return Positive is

  (Configuration.Size ** Configuration.Dimension);

overriding function Nodes_Connected (Configuration   : Topology_Torus;
                                     Node_A, Node_B : Positive) return Boolean is

   subtype Nodes_in_Line is Natural range 0 .. Configuration.Size - 1;
   type Coordinates is array (0 .. Configuration.Dimension - 1) of Nodes_in_Line;

   function To_Coordinates (Node_Nr : Positive) return Coordinates is

      Coordinate : Coordinates;

   begin
      for Dim in 0 .. Coordinate&apos;Last loop
         Coordinate (Dim) := (Node_Nr - 1) / Configuration.Size ** Dim mod Configuration.Size;
      end loop;
      return Coordinate;
   end To_Coordinates;

   Coor                                                  Node_A);
   Coordinate_B : constant Coordi                   ates (Node_B);

   Matching_Coordinates : Natural := 0;

begin
   for Dim in Coordinates&apos;Range loop
      if Coordinate_A (Dim) = Coordinate_B (Dim) then
         Matching_Coordinates := Matching_Coordinates + 1;
      end if;
   end loop;
   if Matching_Coordinates = Configuration.Dimension - 1 then
      for Dim in Coordinates&apos;Range loop
         if      (Coordinate_A (Dim) + 1) mod Configuration.Size = Coordinate_B (Dim)
           or else (Coordinate_B (Dim) + 1) mod Configuration.Size = Coordinate_A (Dim)
         then
            return True;
         end if;
      end loop;
      return False;
   else
      return False;
   end if;
end Nodes_ConnectedTo specTo body;

--  Butterfly

overriding function Nodes_in_Topology (Configuration : Topology_Butterfly) return Positive is

));   ((Con

overriding function Nodes_Connect                           : Topology_Butterfly;
                                                    : Positive) return Boolean is

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

   type Butterfly_Coordinates is record
      Line  : Lines;
      Layer : Layers;
   end record;

   function To_Butterfly_Coordinates (Node : Positive) return Butterfly_Coordinates is

      Coordinate : constant Butterfly_Coordinates := (Line  => (Node - 1) /   (Configuration.Dimension + 1),
                                                      Layer => (Node - 1) mod (Configuration.Dimension + 1));

   begin
      return Coordinate;
   end To_Butterfly_Coordinates;

   Butterfly_A : constant Butterfly_Coordinates := To_Butterfly_Coordinates (Node_A);
   Butterfly_B : constant Butterfly_Coordinates := To_Butterfly_Coordinates (Node_B);

   type Bit_Arrays is array (Bits) of Boolean;

   func

      Bit_Array : Bit_Arrays;

   begin
      for Bit in Bits&apos;Range loop
         Bit_Array (Bit) := (Line_Nr / (2 ** Bit)) mod 2 > 0;
      end loop;
      return Bit_Array;
   end To_Bit_Arrays;

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

      Return_Bits : Bit_Arrays := Bit_Array;

   begin
```

```
         Return_Bits (Bit_Nr) := not Return_Bits (Bit_Nr);
         return Return_Bits;
      end Invert_Bit;

   begin
      return         ((Butterfly_A.Layer < Layers&apos;Last and then Butterfly_A.Layer + 1 = Butterfly_B.Layer)
                      or else (Butterfly_B.Layer < Layers&apos;Last and then Butterfly_B.Layer + 1 = Butterfly_A.Layer))
        and then           (Butterfly_A.Line = Butterfly_B.Line
                            or else ((Butterfly_A.Layer < Butterfly_B.Layer)
                                     and then To_Bit_Arrays (Butterfly_A.Line) = Invert_Bit (Butterfly_A.Layer, To_Bit_Arrays (Butterfly_B.Line)))
                            or else ((Butterfly_B.Layer < Butterfly_A.Layer)
                                     and then To_Bit_Arrays (Butterfly_B.Line) = Invert_Bit (Butterfly_B.Layer, To_Bit_Arrays (Butterfly_A.Line))));
   end Nod[illegible]

   --  Wrap_Around_Butterfly

   overriding function Nodes_in_Topology (Configuration : Topology_Wrap_Around_Butterfly) return Positive is

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

   overriding function Nodes_Connected (Configuration   : Topology_Wrap_Around_Butterfly;
                                        Node_A, Node_B : Positive) return Boolean is

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

   type Butterfly_Coordinates is record
      Line  : Lines;
      Layer : Layers;
   end record;

   function To_Butterfly_Coordinates (Node : Positive) return Butterfly_Coordinates is

      Coordinate : constant Butterfly_Coordinates := (Line  => (Node - 1) /   Configuration.Dimension,
                                                      Layer => (Node - 1) mod Configuration.Dimension);

   begin
      return Coordinate;
   end

   Butterfly_A : constant Butterfly_Coordinates := To_Butterfly_Coordinates (Node_A);
   Butterfly_B : constant Butterfly_Coordinates := To_Butterfly_Coordinates (Node_B);

   type Bit_Arrays is array (Bits) of Boolean;

   function To_Bit_Arrays (Line_Nr : Lines) return Bit_Arrays is

      Bit_Array : Bit_Arrays;

   begin
      for Bit in Bits&apos;Range loop
         Bit_Array (Bit) := (Line_Nr / (2 ** Bit)) mod 2 > 0;
      end loop;
      return Bit_Array;
      end To_Bit_Arrays;

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

         Return_Bits : Bit_Arrays := Bit_Array;

      begin
         Return_Bits (Bit_Nr) := not Return_Bits (Bit_Nr);
         return Return_Bits;
      end Invert_Bit;

   begin
      return         ((Butterfly_A.Layer + 1) mod Configuration.Dimension =
   Butterfly_B.Layer
) mod
   Configuratio
        an
                              or else                     + 1) mod
   Configuration.Dimension = Butterfly_B.
                                       and then To_Bit_Arrays
   (Butterfly_A.Line) = Invert_Bit (Butterfly_A.Layer, To_Bit_Arrays
   (Butterfly_B.Line))
                              or else ((Butterfly_B.Layer + 1) mod
   Configuration.Dimension = Butterfly_A.Layer
                                       and then To_Bit_Arrays
   (Butterfly_B.Line) = Invert_Bit (Butterfly_B.Layer, To_Bit_Arrays
   (Butterfly_A.Line))));
   end Nodes_ConnectedTo specTo body;

   --  Star

   overriding function Nodes_in_Topology (Configuration : Topology_Star)
   return Positive is

     (Configuration.Size);
```

```

overriding function Nodes_Connected (Configuration  : Topology_Star;
                                     Node_A, Node_B : Positive) return Boolean is

  (Node_A = 1 or else Node_B = 1);

--  Fully connected

overriding function Nodes_in_Topology (Configuration : Topology_Fully_Connected) return Positive is

  (Configuration.Size);

overriding function Nodes_Connected (Configuration : Topology_Fully_Connected;
                                         _B         : Positive) return Boolean is

  (True);

--
--  Degrees
--

function Min_DegreeTo API docTo spec (ConfigurationTo API docTo spec : Topology_KindTo API docTo spec&apos;Class) return Natural is

   subtype Nodes_Range is Positive range 1 .. Nodes_in_TopologyTo API docTo spec (ConfigurationTo API docTo spec);

   Min : Natural := Nodes_Range&apos;Last;

begin
```

```
   for i in Nodes_Range loop
      declare
         Degree : Natural := 0;
      begin
         for j in Nodes_Range loop
            if Nodes_ConnectedTo API docTo spec (ConfigurationTo API docTo spec, i, j) then
               Degree := Degree + 1;
            end if;
         end loop;
         Min := Natural&apos;Min (Min, Degree);
      end;
   end loop;
   return Min;
```

```
end Min_DegreeTo API docTo specTo body;

--

function Max_DegreeTo API docTo s [illegible] o API docTo spec : Topology_KindTo API docTo spec&apos;Class) return Natural is

   subtype Nodes_Range is Positive range 1 .. Nodes_in_TopologyTo API docTo spec (ConfigurationTo API docTo spec);

   Max : Natural := 0;

begin
   for i in Nodes_Range loop
      declare
         Degree : Natural := 0;
      begin
         for j in Nodes_Range loop
            if Nodes_ConnectedTo API docTo spec (ConfigurationTo API docTo
spec, i, j) then
                  Degree := Degree + 1;
               end if;
            end loop;
            Max := Natural&apos;Max (Max, Degree);
         end;
      end loop;
      return Max;
   end Max_DegreeTo API docTo specTo body;

   --
   --  Constructors
   --

   functio
return Topol                                              Topology_Mesh&apos;
(Dimension =

   function RingTo API docTo spec (S
return Topology_KindTo API docTo spec&                        logy_Torus&apos;
(Dimension => 1, Size => SizeTo API do

   function StarTo API docTo spec (Size : Positive)
return Topology_KindTo API docTo spec&apos;Class is (Topology_Star&apos;(Size
=> SizeTo API docTo spec));

   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));

   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));

   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));

   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));

   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));

   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;
```