DesignBuilder XML User Guide

DesignBuilder v2025.1

August 2025

Contents

DesignBuilder XML User Guide

This document provides an overview of the DesignBuilder data set and how it may be exported and imported as XML data. It begins with an explanation on how to export and import models followed by an introduction to the organisation of data within a DesignBuilder model, together with a description of object geometry. Subsequent sections refer to elements and sub-elements within the DesignBuilder XML Schema.

DesignBuilder XML is also known as dsbXML.

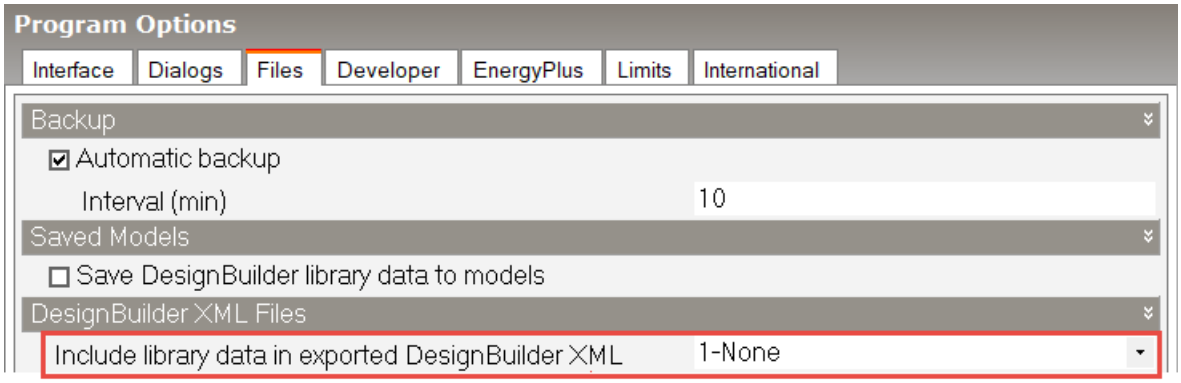
Using DesignBuilder XML

DesignBuilder XML includes the full model description and is an XML equivalent of the SKH binary format that is compressed within standard DSB model files. File format aside, the only difference between dsbXML files and SKH files is that dsbXML files do not include calculation results.

Exporting DesignBuilder XML

DesignBuilder XML files have extension .xml and can be exported using the **File > Export > Export model as XML** menu command.

The **Include library data in exported DesignBuilder XML** program option setting can be used to control whether library data is included in the exported dsbXML file. See screenshot below.



When exporting for subsequent importing to DesignBuilder v2025.1 and later, you should generally use the default **1-None** option to minimise the file size. In this case, only user-defined components and templates are stored in the exported file.

If, on the other hand, the file is intended for loading into a 3rd party tool or into DesignBuilder v7.3 or earlier, you should generally use the **2-All** program option to include all library data in the tables. This ensures that data for all components and templates referenced within the model is included in the XML, making the data set self-contained. Bear in mind that the dsbXML file size can be very large when including all library data.

Note: Library data is identified as any component or template in a data table with an Id value of less than 10,000. User-defined custom data, on the other hand, has an Id of 10,000 or more. Custom data is always exported regardless of the program option setting.

Importing DesignBuilder XML

DesignBuilder XML files can be imported using the **File > Import > Import model from XML** menu command.

After loading the file, DesignBuilder automatically saves the imported model in DSB format to the default model folder (usually Documents\DesignBuilder Data). It uses the name of the dsbXML file as a base and appends a number to ensure uniqueness. For example, when importing the file model.xml, the imported mode will be automatically saved as model1.dsb.

Note: When importing dsbXML files into DesignBuilder, any library component and template data stored in the file is replaced with the latest library data. Imported user-defined data is always retained.

DesignBuilder XML Schema

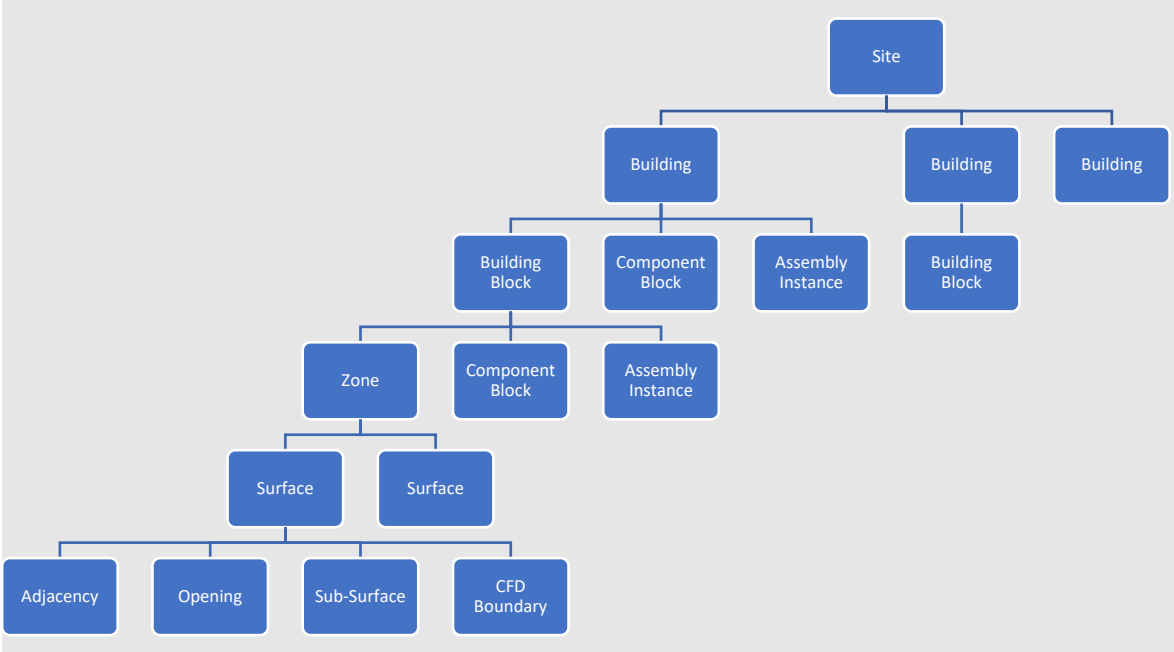
The DesignBuilder XML schema defines the data structure in the DesignBuilder model tree. It can be downloaded from our GitHub repository:

https://github.com/DesignBuilderSoftware/db-dsbxml-schema

The **DesignBuilder.xsd** schema file can be visualised in Visual Studio XML Schema Explorer.

DesignBuilder Model Data Overview

The DesignBuilder building model comprises a hierarchy of objects that collectively represent a building or collection of buildings within a site. The hierarchy is a tree-like structure starting with a single Site object at the top.



The basic model geometric component is a 'block'. There are several types of blocks including Building Blocks, Component Blocks and Outline Blocks. **Building Blocks** are the main model component.

The **Site** and **Building** objects have no inherent geometry themselves but act as containers for other objects. The Site is a container object for **Building** objects, and the **Building** is principally a container object for **Building Block** objects.

A **Building** object, at its simplest, is composed of a single **Building Block** object or a list of **Building Block** objects but may also contain **Component Blocks** and/or Assembly Instances.

A **Building Block**, in turn, at its simplest, comprises a Zone or number of **Zones** but may also contain **Component Blocks** and/or **Assembly Instances**.

Zones are composed of a number of **Surfaces** where each surface represents one of the bounding elements of the zone (walls, floors, ceilings/roofs).

Each Surface has at least one Adjacency and may have a number of Openings, **Sub-Surfaces** and/or **CFD Boundaries**.

An **Adjacency** is a segment of the surface, the geometry of which is essentially a polygon or list of polygons that represent the geometric portion of the surface that is adjacent to the outside of the building or another zone in the building. An **Adjacency** also contains additional information such as construction, etc.

Opening objects represent any opening within the surface, including windows, doors and holes.

Sub-surfaces are similar to Openings but are used to represent segments of a surface that have a different construction to the parent surface.

CFD Boundaries are also similar to openings but are used to model various CFD boundary conditions including temperature and heat flux patches as well as ventilation boundaries.

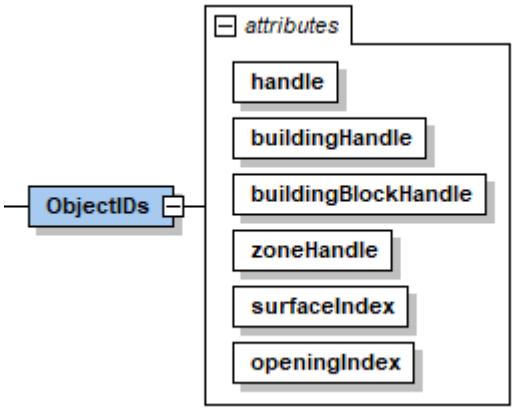
Adjacencies, **Openings**, **Sub-Surfaces** and **CFD Boundaries** are the lowest level objects in the DesignBuilder model hierarchy.

In a similar fashion to **Adjacencies**, the geometry of **Openings**, **Sub-Surfaces** and **CFD Boundaries** are essentially simple polygons and the objects also contain additional information concerning construction, etc.

Each level in the hierarchy is known as a **Level of Decomposition** (**Site**/**Building**/**Building Block**/**Zone**/**Surface**/**Opening**). At each **Level of Decomposition**, a number of different types of model object may be present (**Building Blocks**, **Component Blocks** and **Assembly Instances** at **Building level** and **Zones**, **Component Blocks** and **Assembly Instances** at **Building Block** level). These objects are known as **Decomposition Level Objects**.

DesignBuilder Object IDs and Handles

The DesignBuilder XML elements that represent the main model components (buildings, blocks, zones, etc.) incorporate an **ObjectIDs** element. The **ObjectIDs** are the handles and indices that are used by the software to identify and locate objects. The **handle** is the unique identifier which is assigned to a new object on creation. It's essential that all objects in the model hierarchy are assigned a unique identifier (handle). The current index of the model handle (after incrementation through object assignation) is located in the **handle** attribute of the **Site** element. Building objects are assigned the parent building handle (**buildingHandle**), building block objects are assigned the parent building block handle (**buildingBlockHandle**), etc.



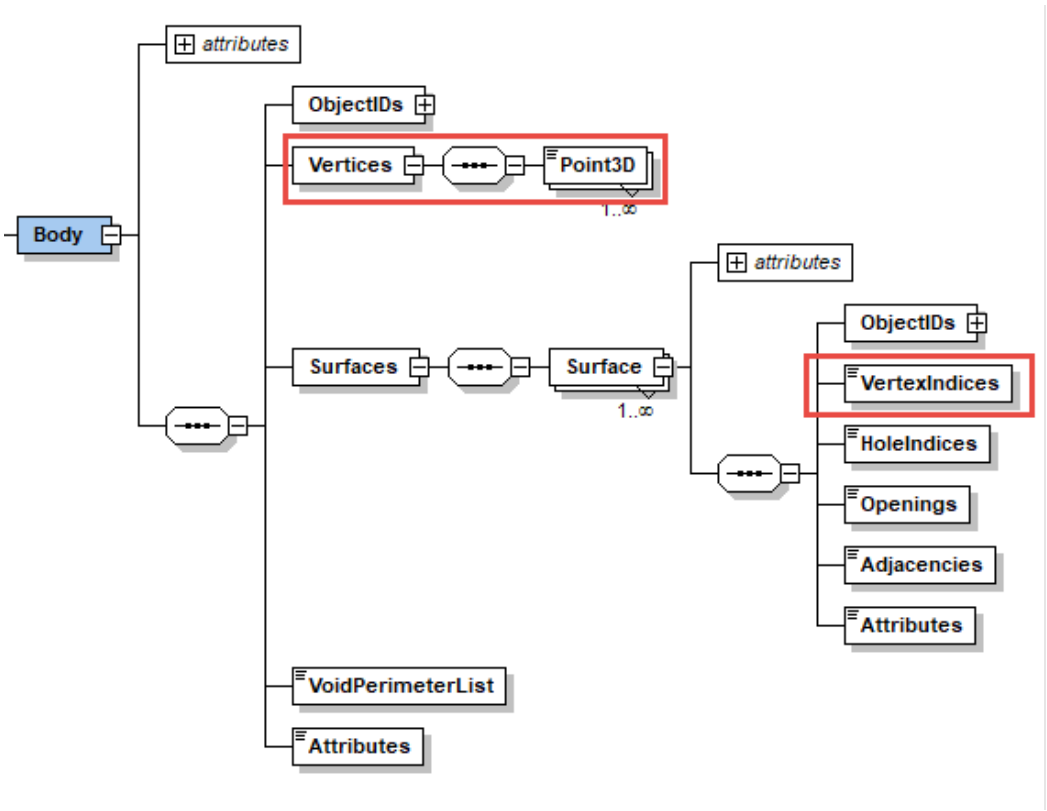
The **zoneHandle**, **surfaceIndex** and **openingIndex** can be used to associate other objects (e.g. polygons) with a parent object.

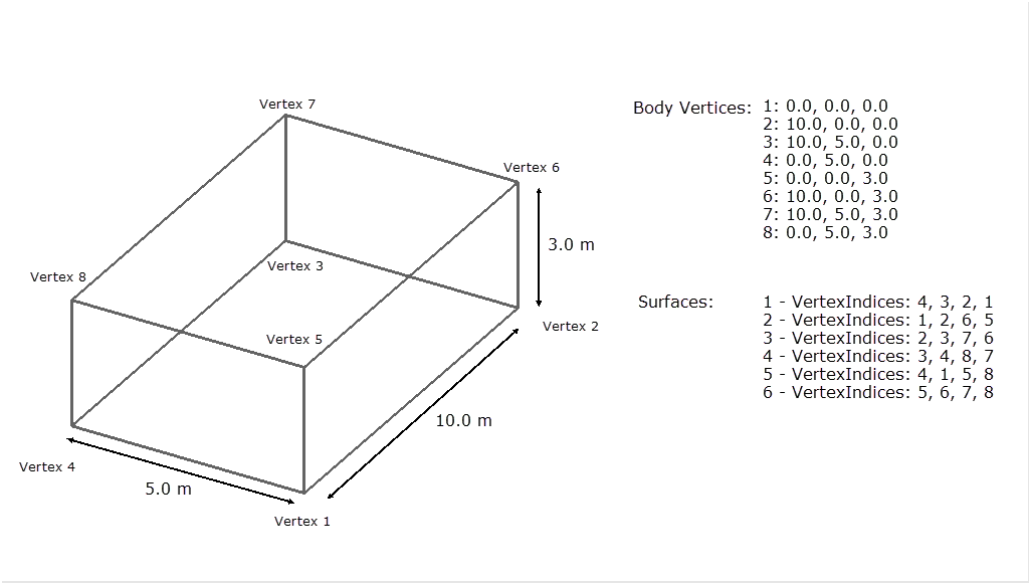
DesignBuilder Geometry

The basic model geometric component from which all blocks are derived is known as a body, represented by the XML **Body** element.

Body

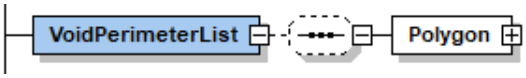
A **Body** is a polyhedron defined using a list of vertices together with a list of surfaces where each surface contains a list of indices that point to vertices in the body vertex list.





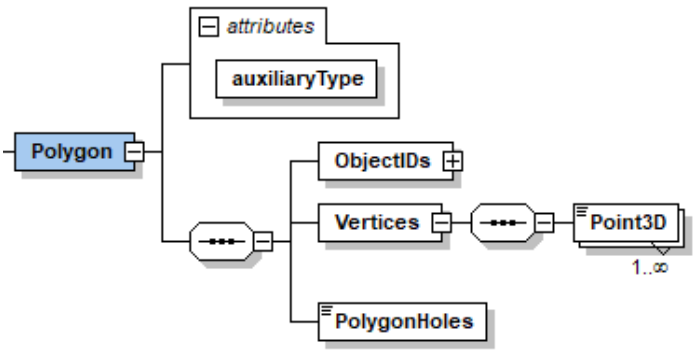
Surface vertices are defined in an anticlockwise order when viewing the body from the outside. The **Vertices** object is a list of 3D points (**Point3D**. All DesignBuilder geometry vertices are written to the XML file as Point3D elements which comprise the X, Y and Z cartesian coordinates of the vertex. The X, Y and Z coordinates are written using a precision of 16 decimal places.

Alongside a list of **Surfaces**, a **Body** object also incorporates a **VoidPerimeterList** object. The **VoidPerimeterList** only applies to plan extrusion bodies and comprises a list of plan void perimeters within the body. Voids created from the **VoidPerimeterList** are known as 'soft' voids because they can easily be removed from the parent body. Voids that are created in **Body** objects using Boolean operations are known as 'hard' voids because they cannot be removed once formed. Void perimeters are defined using the **Polygon** element:



Polygon

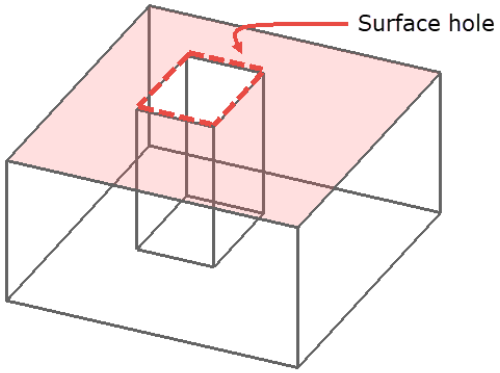
The **Polygon** element is made up of a list of vertices (**Point3D** elements) which are ordered in an anticlockwise sequence. Polygons can incorporate holes which are themselves **Polygon** objects, but where the vertices are ordered in a clockwise sequence. A list of polygon holes is defined using the **PolygonHoles** element.

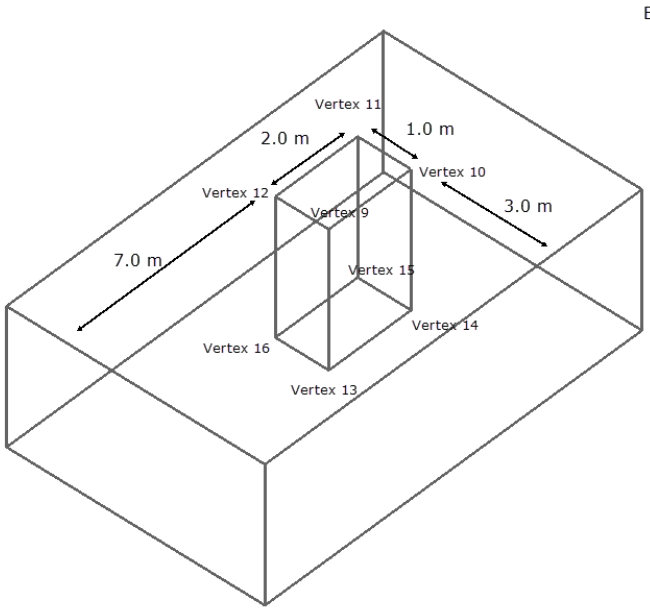


Surface

As well as incorporating the **VertexIndices** object, which contains a list of indices that point to vertices in the **Body Vertices** object, the **Surface** object also incorporates a **HoleIndices** object. The **HoleIndices** object contains a list of surface indices where each index points to another **Surface** object in the **Surfaces** list that represents a hole in the **Surface**.

For example, for a body containing a void:



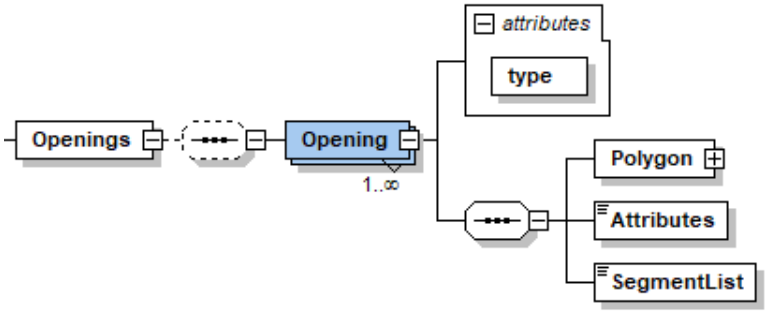


Body Vertices:
1: 0.0, 0.0, 0.0
2: 10.0, 0.0, 0.0
:
:
9:

Surfaces:
1 - VertexIndices: 4, 3, 2, 1
HoleIndices: 7
2 - VertexIndices: 1, 2, 6, 5
HoleIndices: 8
:
:
7 - VertexIndices: 13, 14, 15, 16
8 - VertexIndices: 9, 10, 11, 12
9 - VertexIndices: 13, 14, 10, 9
10 - VertexIndices: 14, 15, 11, 10
11 - VertexIndices: 16, 15, 11, 12
12 - VertexIndices: 16, 13, 9, 12

Opening

The **Openings** element is of particular significance for body objects representing building zones (see **Zone** element), where each **Opening** may represent a window, door, vent, hole or CFD boundary element.

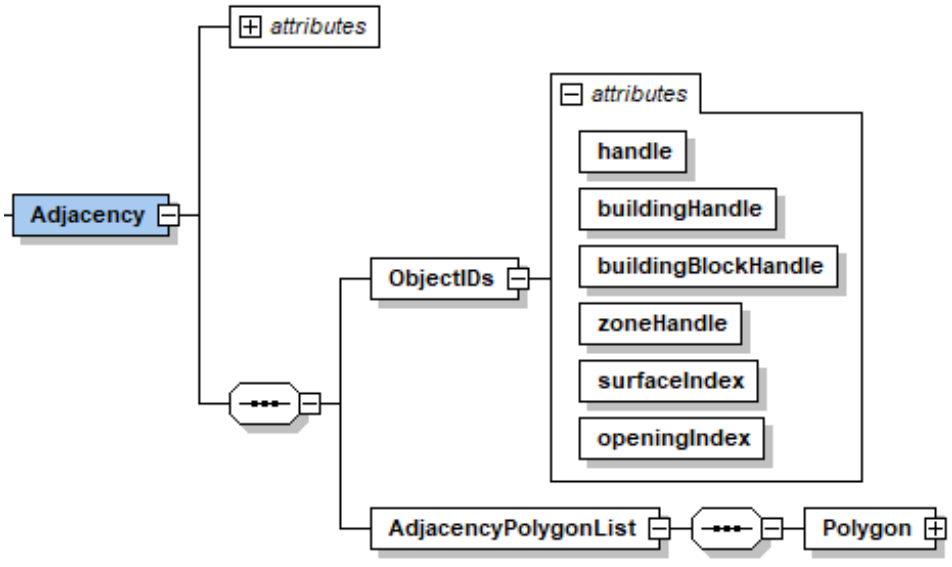


The geometry of an Opening is represented using a **Polygon** element. The attribute **type** contains an enumerated surface type which can be one of the types listed in Appendix I.

Adjacency

The Adjacency element is used to specify what is adjacent to a particular section of a surface. In the case of a building zone surface, the surface may be adjacent to a zone, several other zones and/or the exterior of the building.

The geometry of the portion of the surface occupied by an adjacency is represented using a list of **Polygon** elements. The type of adjacency is defined using the adjacency **ObjecIDs** element whereby the **zoneHandle** element attribute if non-negative contains the handle of the zone (or body) to which the surface is adjacent. If the **zoneHandle** is non-negative, then the **surfaceIndex** attribute contains the index of the adjacent surface in the adjacent body. If the **zoneHandle** is negative (-1), this indicates that the adjacency is to the exterior of the building.



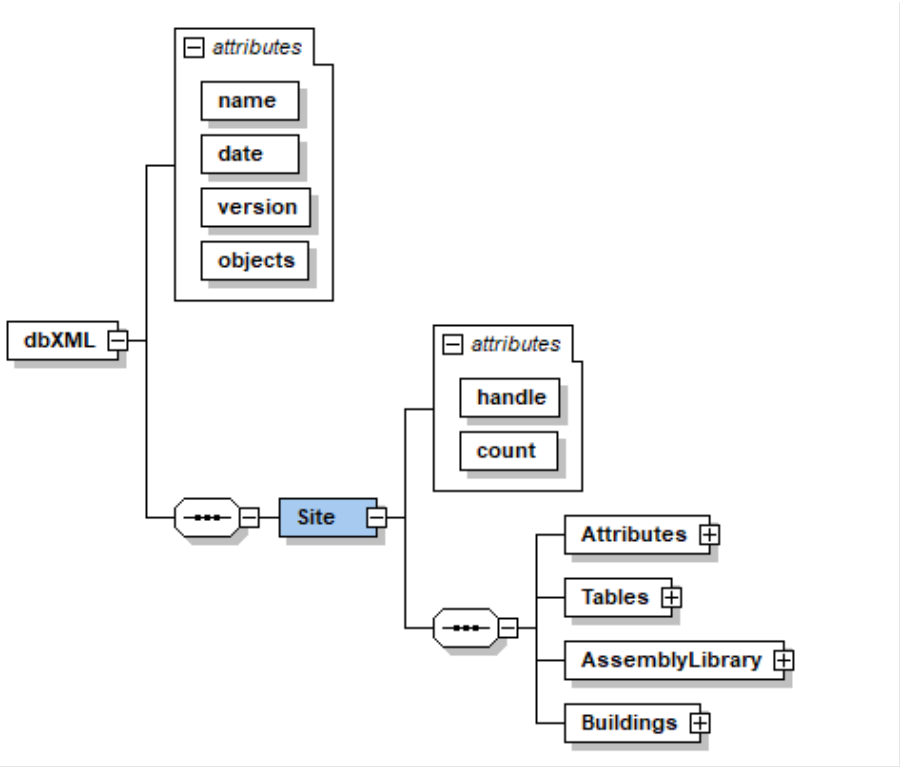
Element Hierarchy within the DesignBuilder XML Schema

The following sections describe the function of the main elements within DesignBuilder XML.

Site

The Site object is the highest-level object in the model hierarchy. The Site object incorporates a list of Building objects, Tables, Attributes and an Assembly library.

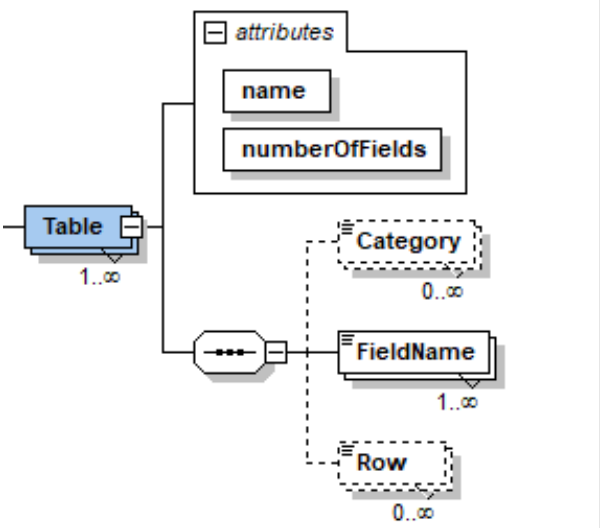
The following diagram shows the XML schema for the Site object:



Tables

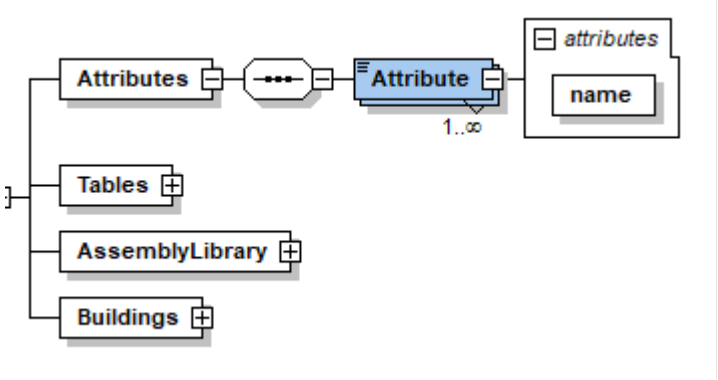
Tables are used to store and manipulate data that is best suited to a tabular format, such as building materials, constructions, glazing types, etc. Tables are also used by DesignBuilder for storage of results and intermediate data. Tables comprise a specified number of fields (columns) and any number of records (rows). Tables used to house data such as constructions, etc. require the data to use a specific format whereby the field of each record is delimited using a '#' character, the first record contains field categories and the second record contains the field names. In contrast, tables used for results and intermediate storage don't have the same requirement for the first two records and data is accessed exclusively by index.

A list of category IDs are located at the top of DesignBuilder Component and Template tables. Category IDs are referenced using the 'CategoryId' field. Category IDs point to externalised text in LocalisedCategories_n.txt files.



Attributes

All non-geometric data is stored and manipulated using **Attribute** objects. An Attribute is a simple object comprising a pair of strings. One string houses the name of the attribute, and the other string houses the value. DesignBuilder employs an inheritance mechanism whereby attributes are automatically inherited from an object at a higher level in the object hierarchy (see DesignBuilder Model section below) to a lower level unless the attribute is 'hard-set' at the lower level. When an attribute is 'hard-set' at a particular level, it is stored explicitly at that level. Using this approach, the software can obtain a specific setting value by walking up the hierarchy until it finds an attribute with the setting name at which point it can retrieve the value. Consequently, each object in the hierarchy will only store attributes that have been hard-set at that level in the hierarchy.

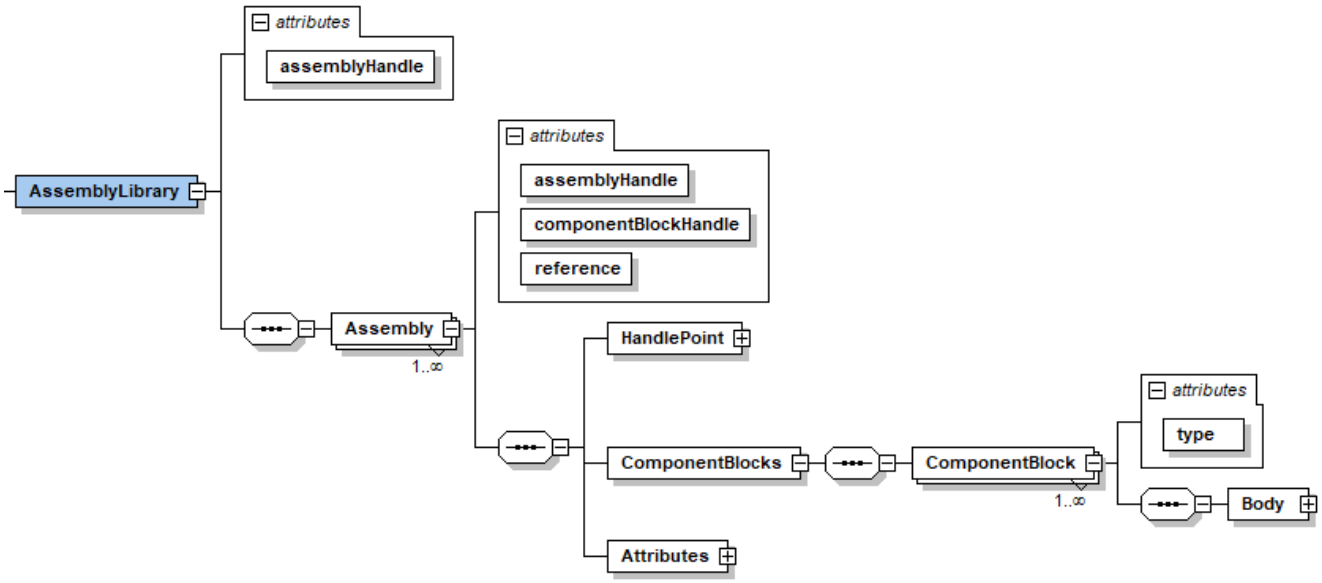


Assembly Library

The **AssemblyLibrary** is the DB XML element that contains the model assembly library. The model assembly library is a container object for compound objects known as assemblies. These compound objects are assemblies of component blocks (**Body** objects). Assemblies are used to model reusable components that may be common to all building models, e.g. occupants, furniture, planters, trees, etc.

Assemblies may be defined through the UI by selecting a number of component blocks and invoking a tool which will automatically process the blocks to form an assembly and add it to the model assembly library. The **HandlePoint** is a point on the assembly which is defined by the user via the UI when adding an assembly to the assembly library.

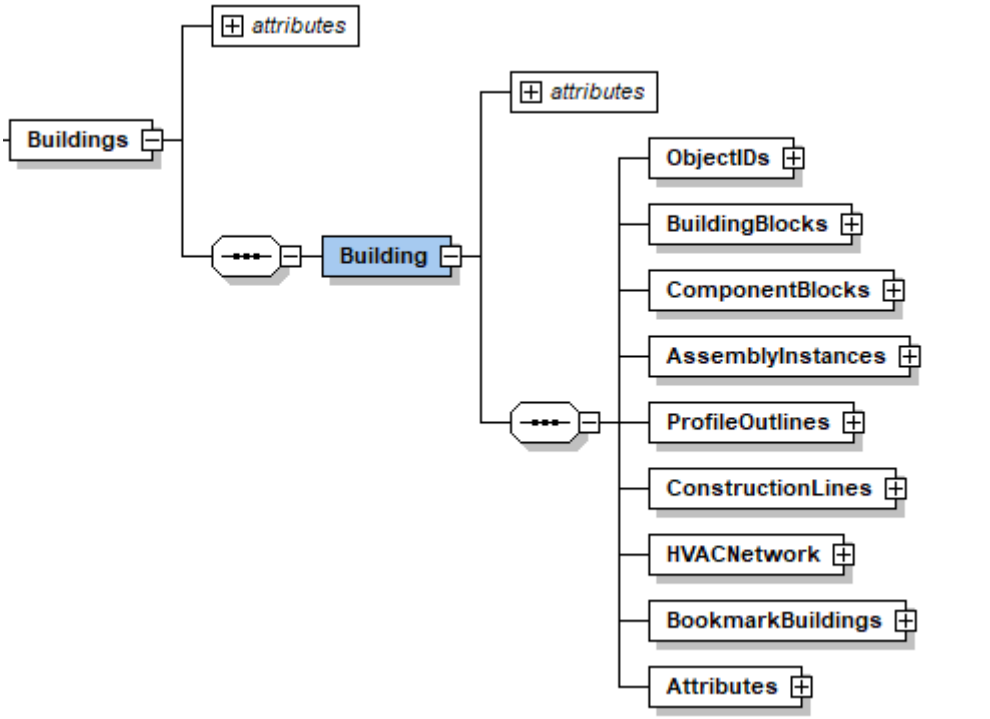
Each library assembly contains a unique handle (distinct from object handles used throughout the model). When an assembly is added to the library, it is assigned its own handle (**assemblyHandle**) using the library handle (**assemblyHandle**) which is subsequently incremented. Similarly, each component block in the assembly component block list is assigned a unique handle (an **ObjectIds handle** unique to the assembly) , which is allocated using the assembly component block handle, which is incremented each time a component block is added to the assembly.



Assemblies are not added directly to the model but rather as 'instances' of the assembly. An assembly instance is a memory/storage efficient mechanism that simply stores a pointer to the parent assembly together with a transformation matrix that allows the software to determine any rotation, scaling or translation that needs to be applied to the assembly.

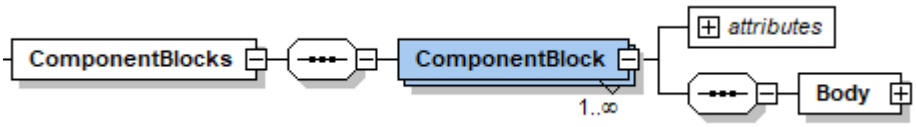
Building

The Building is the second highest object in the model hierarchy and contains a number of objects that collectively represent the building model together with various components that are used to help in graphically developing the model through the UI.



Component Block

A building component block is a simple **Body** object that can be used to model objects such as shading devices for thermal simulation and airflow obstructions for CFD simulations. Component blocks are also the main geometric component of assemblies (see Assembly Library).



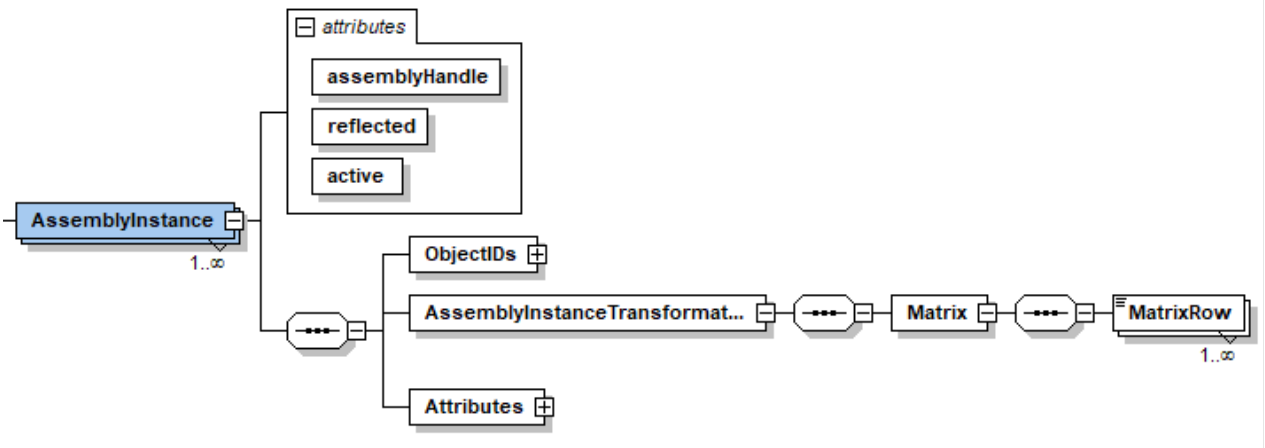
Profile Outline (Outline Block)

A profile outline is similar to a component block in that it is a simple **Body** object. However, profile outlines, unlike component blocks, aren't part of the model as such but instead can be used to assist in model creation. Profile outlines can intersect each other and consequently can be used in the creation of complicated blocks where cutting and Boolean operations are required. After arriving at the required geometry, profile outlines can be converted to building blocks or component blocks.

Profile outlines are known as 'outline blocks' within the DesignBuilder UI for the sake of consistency with other blocks (building blocks and component blocks).

Assembly Instance

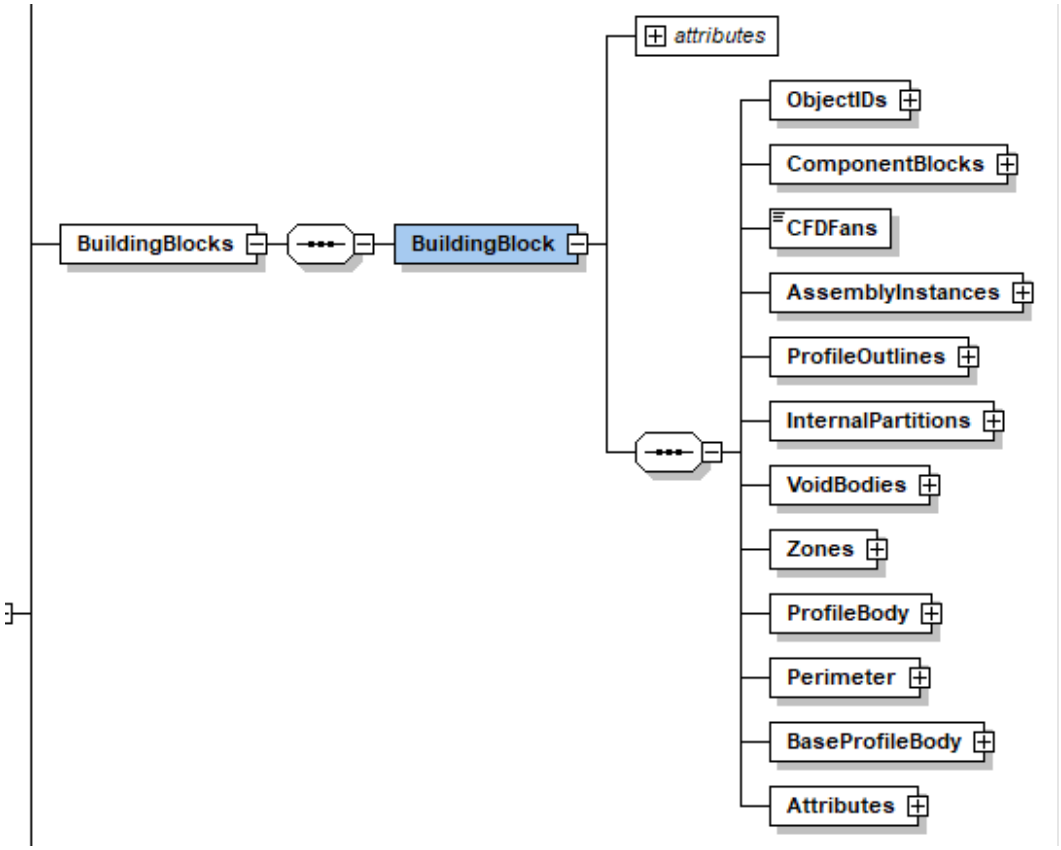
DesignBuilder has a feature that enables collections of component blocks to be combined to form a single composite element known as an assembly. This feature allows complex building features such as furniture, occupants, external shading devices, etc. that incorporate several component blocks to be defined once and then placed as a single item repeatedly throughout the model. The assembly is not added directly to the model but instead added to an assembly library, and then an 'instance' of that assembly is placed in the model. The assembly instance is merely a handle that points to the parent assembly together with a transformation matrix that incorporates all scaling, translation and rotation information required to transform the parent assembly into the required instance.



Building Block

One of the most significant elements in the **Building** object is the **BuildingBlock** which is listed under the **BuildingBlocks** element. A **BuildingBlock** is the main component of a DesignBuilder model.

BuildingBlock objects are essentially polyhedral objects and can take on virtually any form. Building blocks can be divided into a number of zones, and this process is described in detail under the **InternalPartition** section.

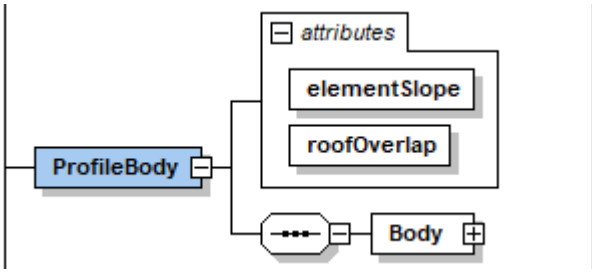


Component Blocks, Assembly Instances and Profile Outlines

The component block, assembly instance and profile outline (outline block) elements are identical to the elements contained within the **Building** element. Refer to the **Building** element entries for further information.

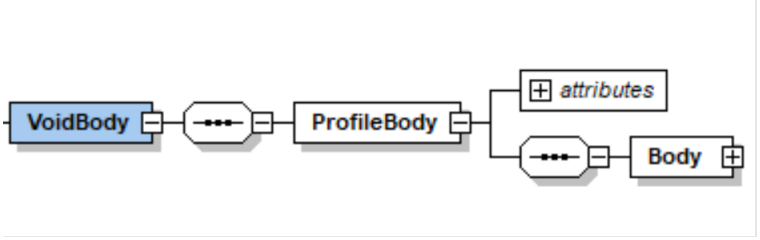
Profile Body and Base Profile Body

The 3D geometry of a **BuildingBlock** element is defined using two **Body** elements , the **ProfileBody** and the **BaseProfileBody**. The **BaseProfileBody** is the body from which the building block was created (i.e. the block polyhedron drawn using the DB UI). The **ProfileBody** is then created simply by copying the **BaseProfileBody**. The software uses this mechanism to allow the building block profile topology to be changed completely (say to a roof topology) and then restored to the original topology (using the **BaseProfileBody** object. The **ProfileBody** also contains two XML attributes which are used to define sloping wall and roof type blocks, the **elementSlope** and **roofOverlap**.



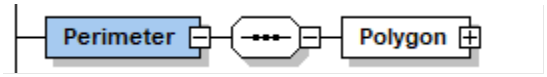
Void Bodies

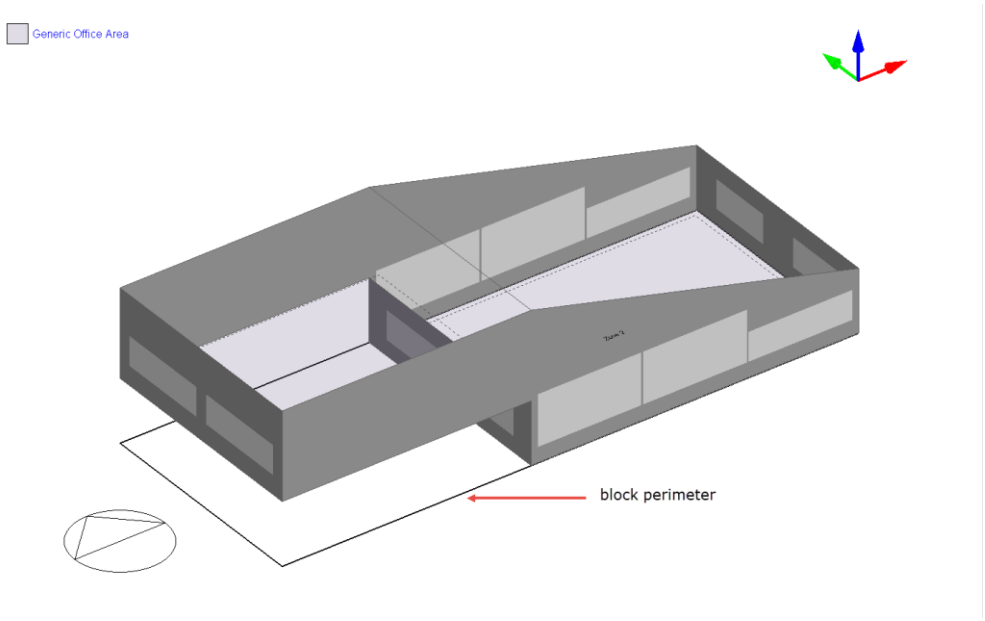
Void bodies are essentially profile bodies that are added to a building block using the 'Add void' UI tool. Voids are objects used to model courtyards or any other external space contained within the perimeter of a building block. This type of void is known as a 'soft void' in DesignBuilder in that it can be moved, copied or deleted and doesn't affect the geometry of the parent building block in the same way as a 'hard void' that can be added to a building block using the UI Boolean tool.



Perimeter

The **Perimeter** element is a polygon defining the floor plate perimeter of the building block which is used when partitioning the block.

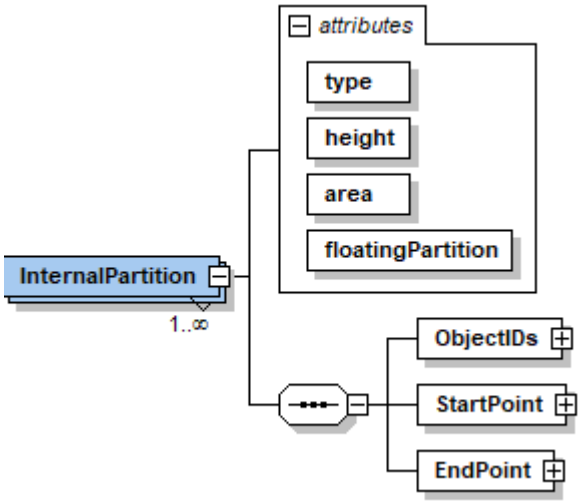




By default, a building block contains a single zone having the same topology as the building block. Building blocks can then be partitioned into any number of zones using **InternalPartition** objects.

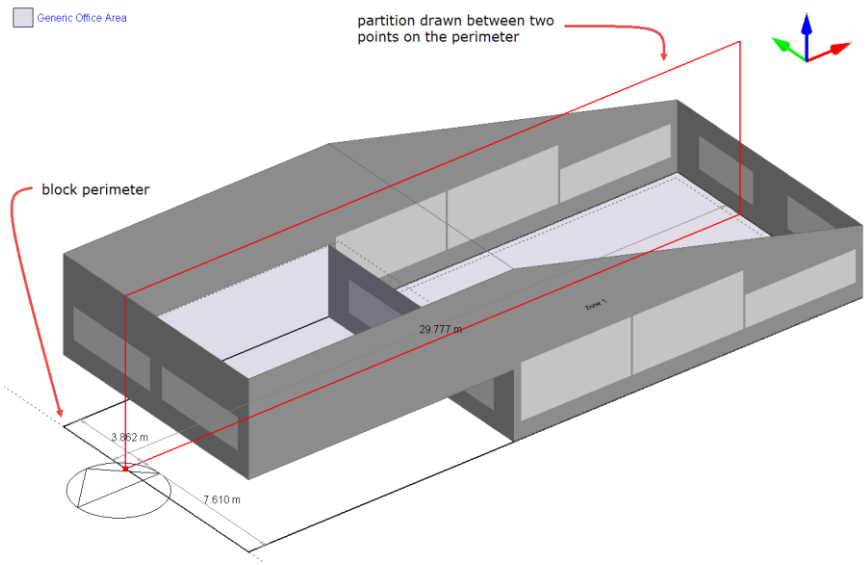
Internal Partition

An **InternalPartition** is simply a line segment that extends from a point on the block base perimeter to another point on the perimeter or to a point on another partition.

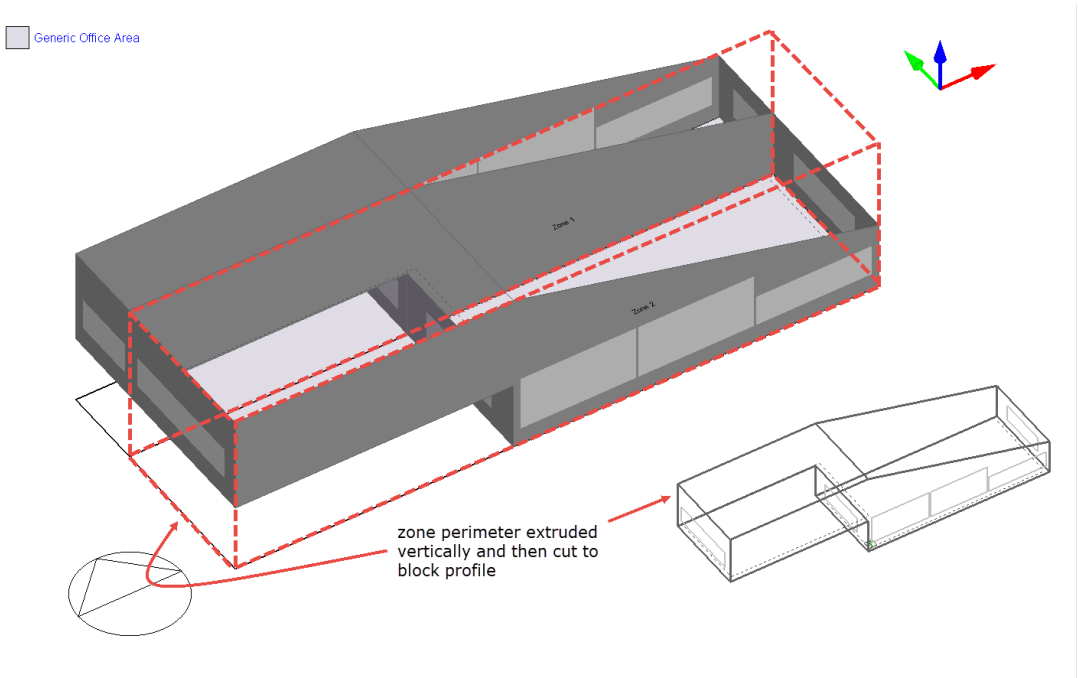


The **StartPoint** and **EndPoint** elements are the **Point3D** elements that define the starting and finishing points of the line segment that represents the partition.

Partitions can also be located between two points on different partitions.



The software extrudes the perimeter of each zone formed by these partition/perimeter intersections to span the height of the block and then cuts the extruded zone to fit the block profile using a Boolean intersection process.

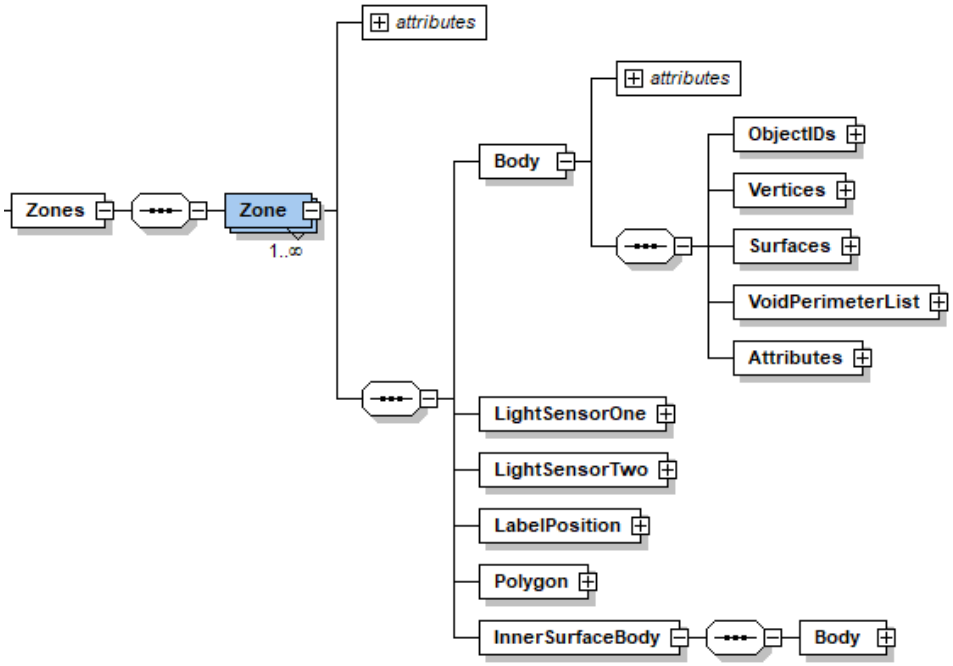


The main limitation when partitioning blocks in DB is that these partitions can only be vertical (blocks can't be partitioned horizontally), and consequently, Separate blocks have to be drawn, one on top of another to represent a horizontal partitioning.

Building blocks can also be partitioned using a special type of internal partition known as a 'virtual' partition. Virtual partitions are essentially the same as standard internal partitions but incorporate a hole that occupies the entire area of the partition. Virtual partitions are used within DesignBuilder to allow zones to be separated into sub-zones without using a solid partition. Quoting from the DesignBuilder Help documentation: "*Virtual partitions are commonly used for separating perimeter zones from core zones when there is to be different HVAC provision or when carrying out daylighting or solar overheating studies in situations where a large open plan space is subject to a high level of solar gain around the perimeter*". The only difference between virtual partition XML elements and standard partition elements is that the **Type** attribute is set to "***Virtual***" rather than "***Solid***".

Zone

A zone is the main entity used to house data for energy simulation. The geometry of a zone is defined using a **Body** object:



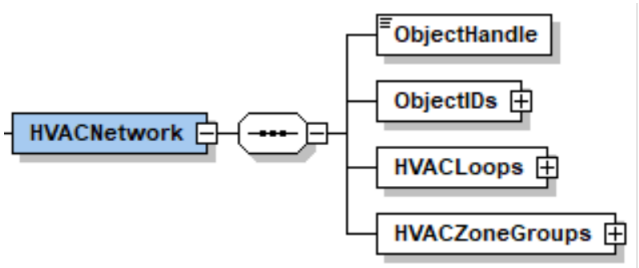
All non-geometric zone data such as setpoint temperatures, heat gains, etc. are stored in the **Attributes** element of the **Body**.

Another body, described by the **InnerSurfaceBody** XML element, is used to define the geometry of the polyhedron that represents the inner surfaces of the zone. This mechanism is used to account for the construction thickness of the various zone elements (walls, ceiling, etc,) in the derivation of zone volumes, surface areas, etc.

Zones are volatile entities in that they are continuously destroyed and recreated during building block editing (i.e. addition/removal of partitions, etc.). Whenever a building block is edited, the software stores all current zones prior to applying the edits and then reassigns all stored data (custom openings, constructions, etc.) to the regenerated zones using a geometric matching mechanism.

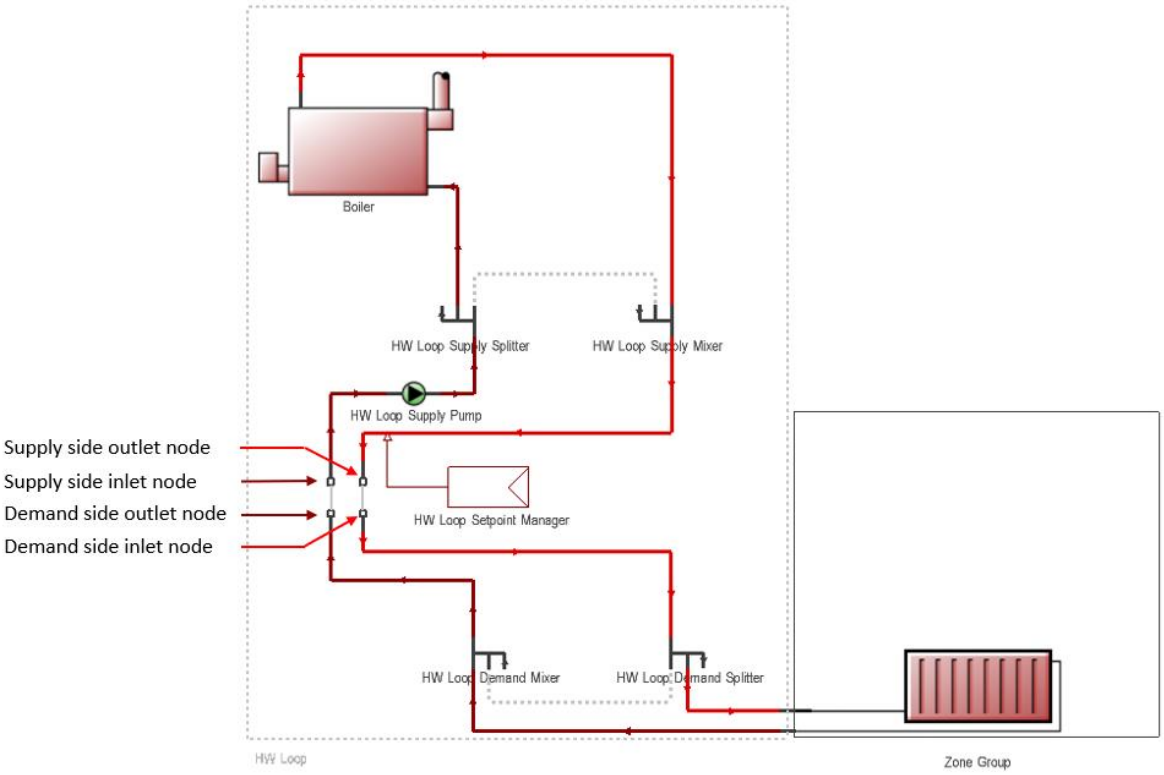
HVAC Network

The HVAC Network element houses all of the elements associated with the DesignBuilder detailed HVAC system. DesignBuilder detailed HVAC is the user interface to the EnergyPlus HVAC simulation facility. The structure of the DesignBuilder HVAC network closely follows the structure of the EnergyPlus HVAC network and further information concerning EnergyPlus HVAC can be obtained from the EnergyPlus "***InputOutputReference***" document.



An EnergyPlus HVAC network consists of various components that are connected physically in the actual system by ducts, piping, etc. Every component in the network has an inlet and outlet "node". In the actual system, a node might be a point in the system at which fluid properties can be measured. In an EnergyPlus simulation, the nodes are points at which fluid properties are evaluated

and passed on to subsequent equipment. Components are linked together to form various loops within the network. Thus, the output node from one component also serves as the inlet node to the next component. Loops are constructed from closed chains of these components. Although loop nodes are a key defining feature in EnergyPlus, they do not appear explicitly within the DesignBuilder UI where connectivity between components is defined by connecting components using graphical representations of pipes and ducts.



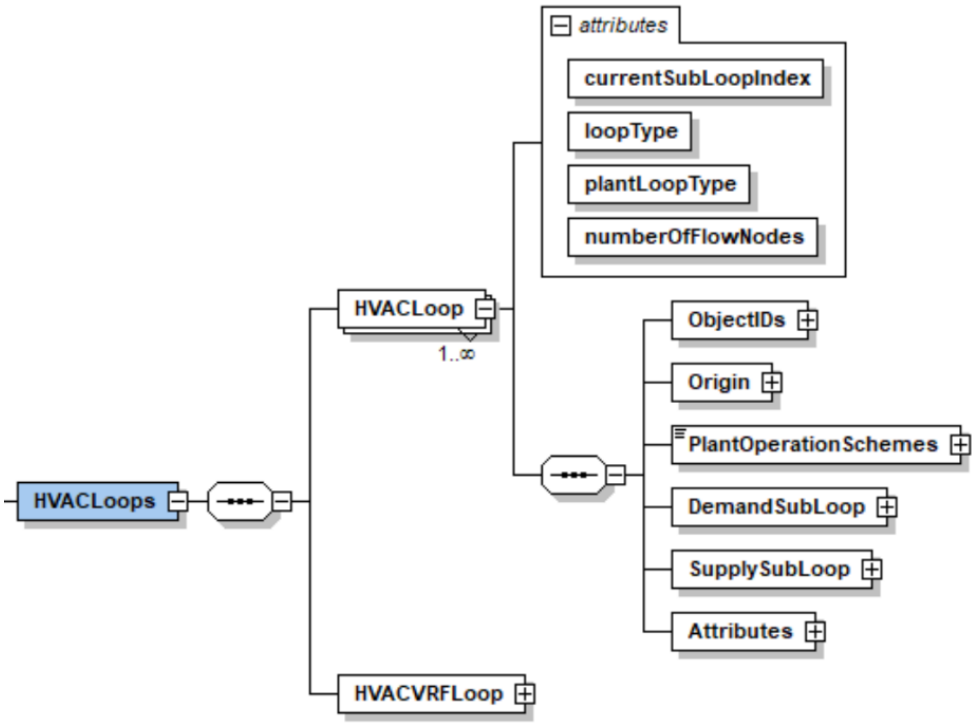
The EnergyPlus nodal data is generated automatically within DesignBuilder from graphical component connectivity data and stored in the form of HVAC component attributes which are then used to write the EnergyPlus nodal data explicitly in the form of IDF data for subsequent EnergyPlus simulation.

In the case of the simple heating system shown above, the heating loop sub-loop nodes are depicted graphically using a sub-loop connection component with pipework connections to the pump/boiler on the supply side and to a zone convector on the demand side and these translate to nodes when exported to IDF:

```
PlantLoop,
   HW Loop,                                                    ! - Name of loop
   Water,                                                      ! - Fluid type
   ,                                                           ! - User-defined fluid type
   HW Loop Operation,                                          ! - Plant equipment operation scheme
   HW Loop Supply Side Outlet,                                 ! - Loop temperature setpoint node
   100.00,                                                     ! - Maximum loop temperature (C)
   10.00,                                                      ! - Minimum loop temperature (C)
   autosize,                                                   ! - Maximum loop flow rate (m3/s)
   0.000000,                                                   ! - Minimum loop flow rate (m3/s)
   autocalculate,                                              ! - Volume of the plant loop (m3)
   HW Loop Supply Side Inlet,                                  ! - Plant side inlet node
   HW Loop Supply Side Outlet,                                 ! - Plant side outlet node
   HW Loop Supply Side Branches,                               ! - Plant side branch list name
   HW Loop Supply Side Connectors,                             ! - Plant side connector list name
   HW Loop Demand Side Inlet,                                  ! - Demand side inlet node
   HW Loop Demand Side Outlet,                                 ! - Demand side outlet node
   HW Loop Demand Side Branches,                               ! - Demand side branch list name
   HW Loop Demand Side Connectors,                             ! - Demand side connector list name
   SequentialLoad,                                             ! - Load distribution scheme
   HW Loop AvailabilityManager List,                           ! - Availability manager list name
   SingleSetpoint,                                             ! - Plant loop demand calculation scheme
   None,                                                       ! - Common pipe simulation
   None;                                                       ! - Pressure simulation type
```

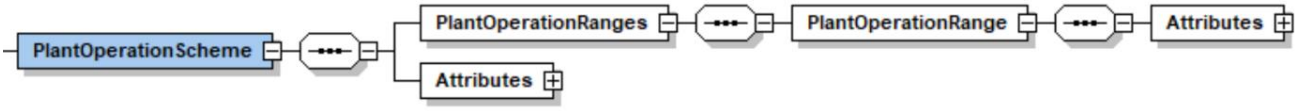
HVAC Loop

The HVACLoop element is used to represent all EnergyPlus HVAC network loops including **AirLoopHVAC**, **CondenserLoop** and **PlantLoop** objects. The **HVACLoop** is comprised of two sub-loops, a supply sub-loop and a demand sub-loop. The supply sub-loop contains components that supply energy to the loop and demand sub-loops contain components that demand energy from the loop. The plant and condenser loops are virtually identical.



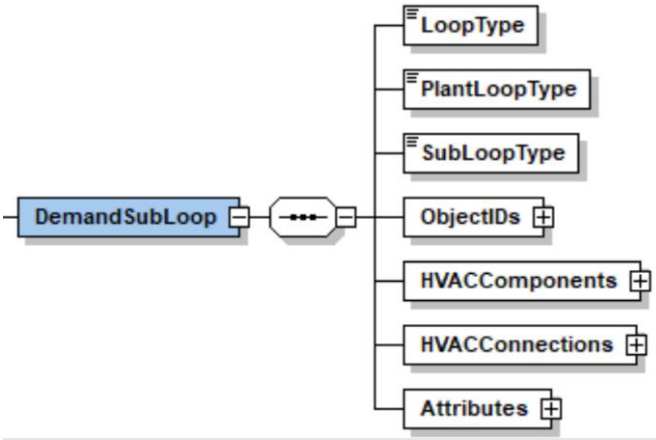
Plant Operation Scheme element

The Plant Operation Schemes element houses the data that defines equipment and operational equipment ranges for the associated loop. The data is stored in the form of attribute data.



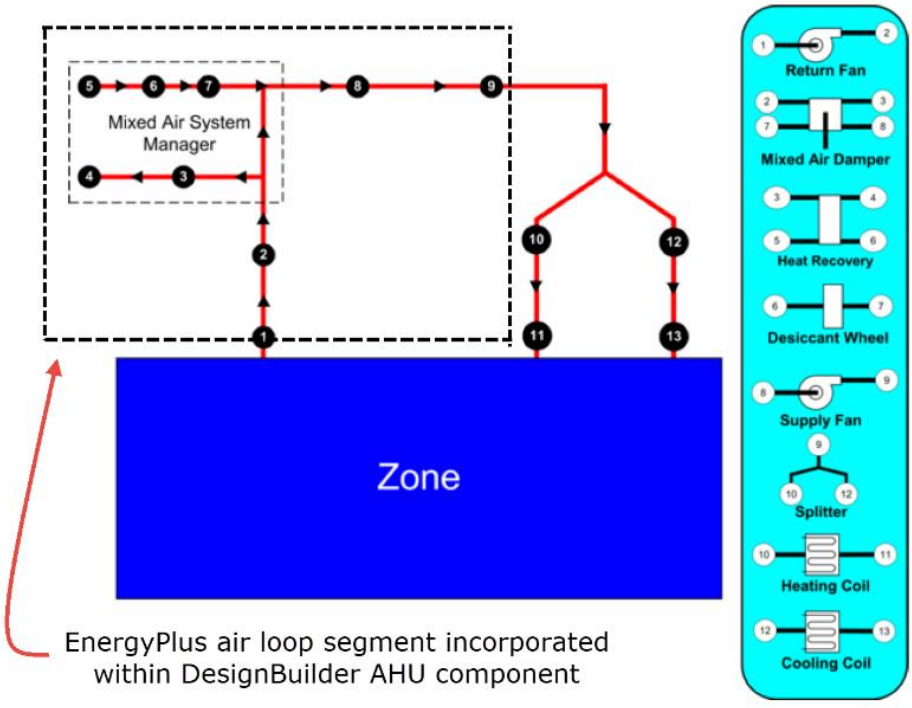
Supply and Demand Sub-Loop Elements

The supply and demand sub-loop elements incorporate the loop type (air, plant, condenser), sub-loop type (supply, demand) together with lists of sub-loop components and the connections (pipework and ductwork) that connect the components.



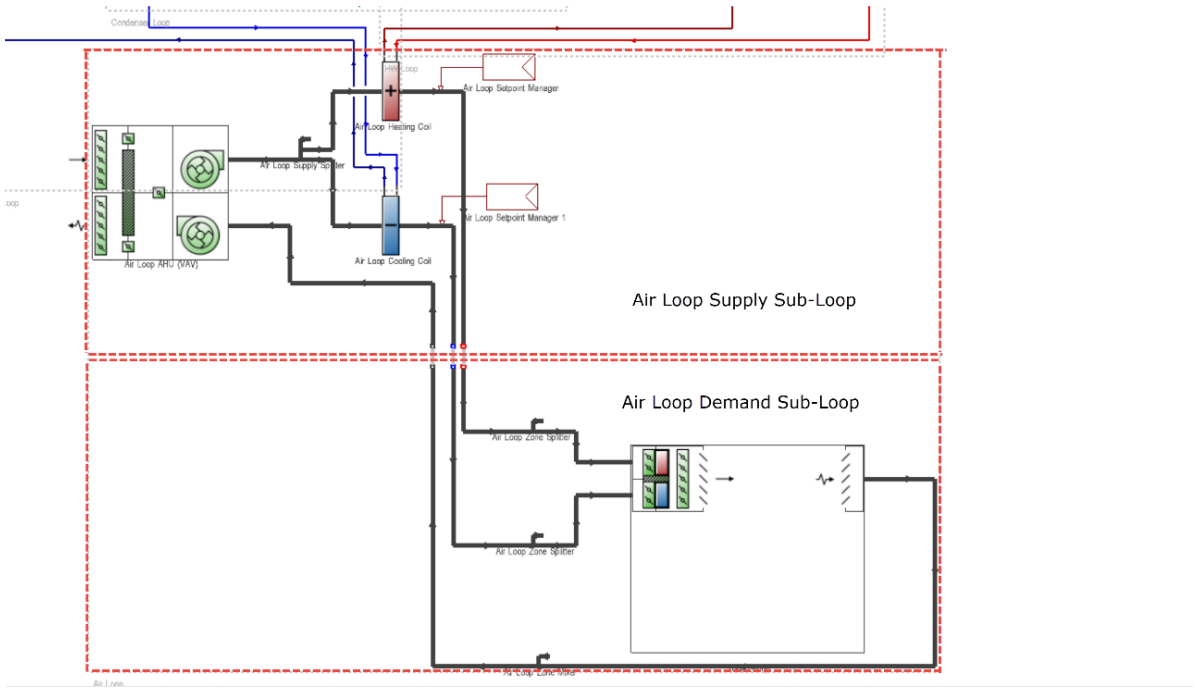
Air Loop Demand Side Sub-Loop (Zone Equipment):
The demand side sub-loop incorporates all zone equipment. Zone equipment can include dampers and reheat coils as well as zone-specific conditioning systems such as thermostatic baseboard/convectors, fan-coil units or a window air conditioner.

Air Loop Supply Side Sub-Loop:
The EnergyPlus air loop is defined by the section of the zone/air loop that starts after the zone return streams are combined and continues on until just before any air stream(s) are branched off to individual zones. The starting point of the air loop is fairly straightforward. The ending point is slightly more complex but can be understood with some examples. For instance, in a terminal reheat system, the end of the air loop would typically be considered the node following the cooling coil. In a dual duct system, the air loop would have two ending points that would correspond to the nodes after the cooling coil and after the heating coil/humidifier. In most cases, the air loop has a single starting point and up to two ending points (for a dual duct system). An outdoor air subsystem can be included in the supply side for ventilation and relief air.



For clarity and to provide a more familiar layout to designers, the various EnergyPlus air loop supply components which constitute the air handling section of the loop are incorporated within a bespoke

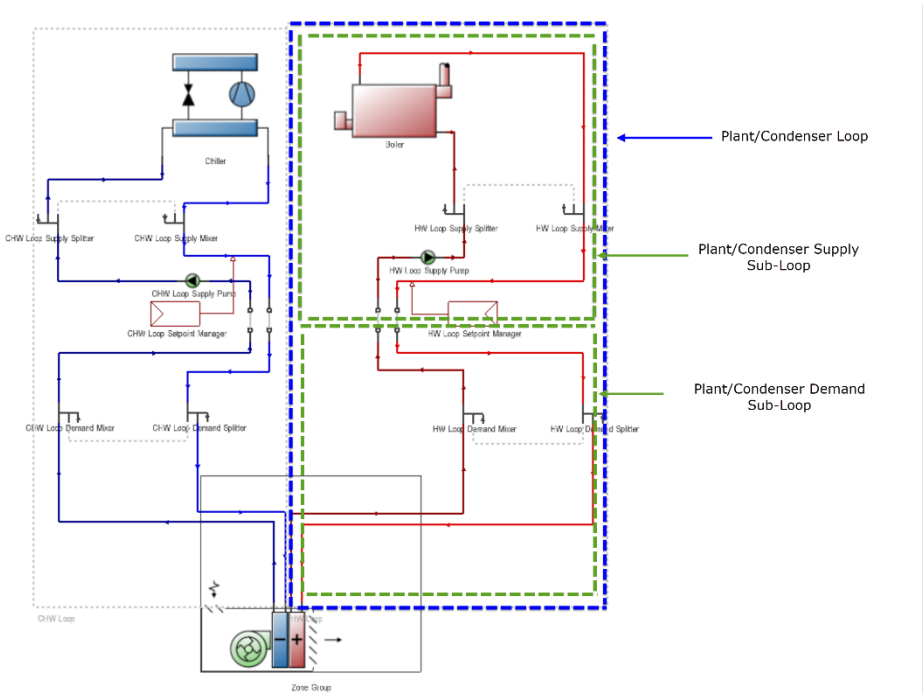
DesignBuilder air handling unit (AHU) component. So, the EnergyPlus dual duct layout illustrated above is represented graphically within DesignBuilder as:



Notice that the hot/cold ducts (together with coils) are represented more or less as direct equivalents, whereas all the other loop components (including the mixed air system manager) are combined within a single AHU component.

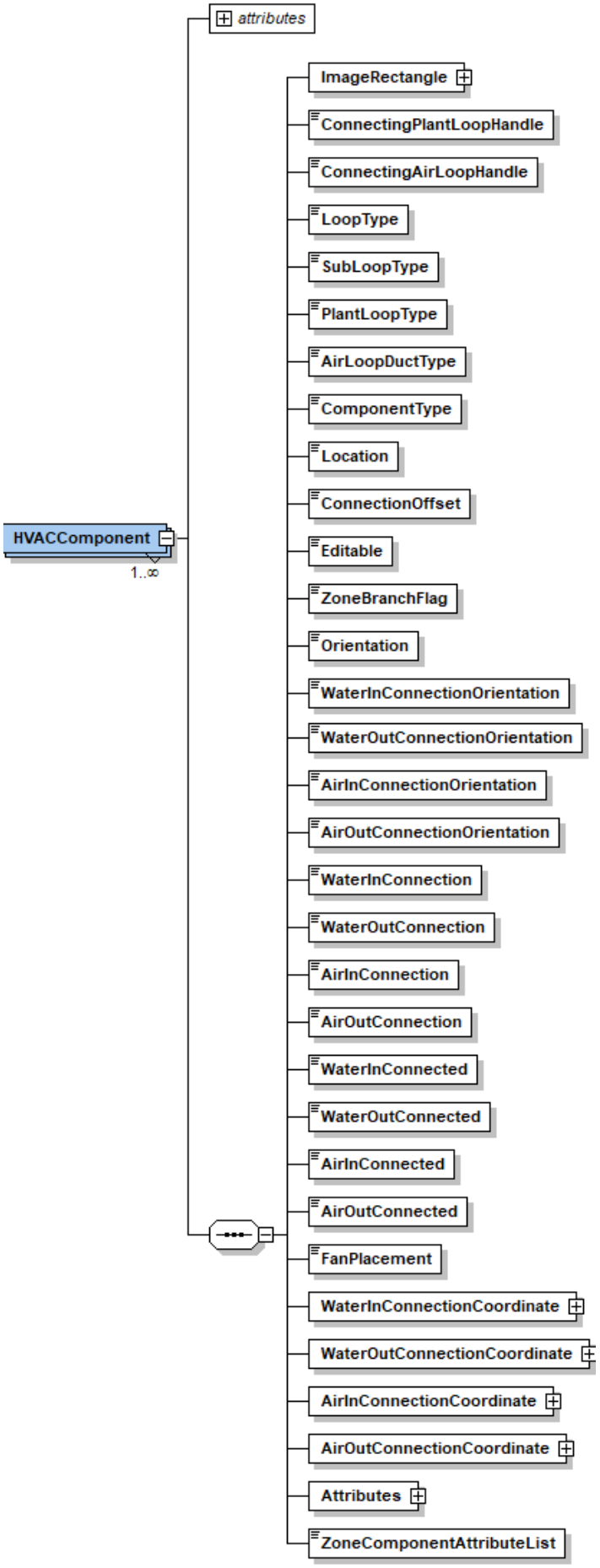
Plant/Condenser Loop Demand Side Sub-Loop:
This side of the plant loop is where energy is “demanded” by various components that make up the air loop or zone equipment. Typically, this is the water side of equipment such as coils, baseboard, radiant heating and cooling, etc. In the case of a condenser loop, energy is typically “demanded” by a chiller condenser or other water source heat pump. The demand side of this loop can also include mixers, flow splitters, and a bypass.

Plant/Condenser Loop Supply Side Sub-Loop:
The other side of the plant/condenser loop is where energy is “supplied” by various components. The components typically found on the supply side include pumps, boilers, chillers, purchased heating and cooling, ice storage, etc. In the case of a condenser, the components would be cooling tower, fluid cooler, or ground source heat exchanger, etc. As with the demand side, this loop can also include mixers, flow splitters, and a bypass.



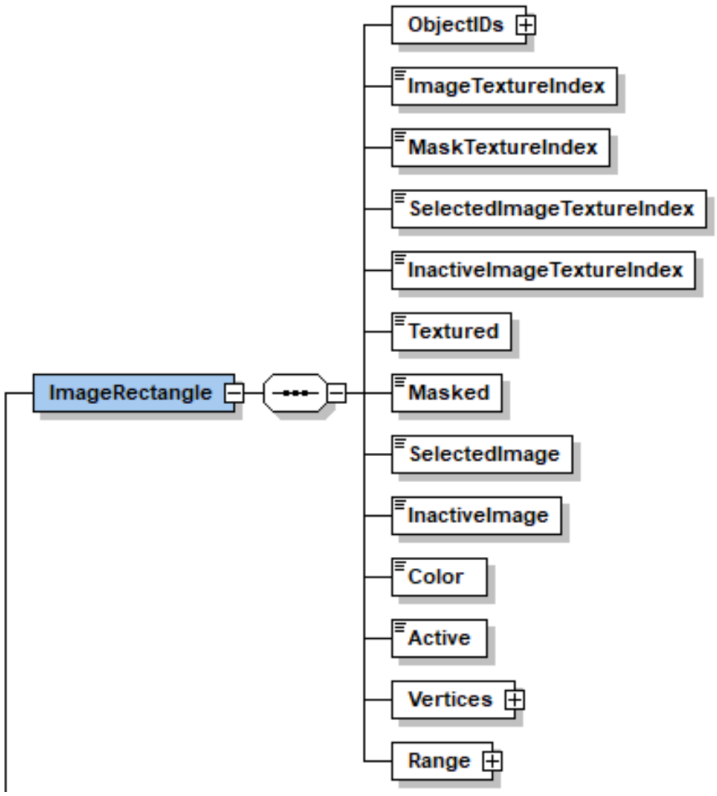
HVAC Component

The **HVACComponent** element is the base element from which all HVAC components are derived. The element incorporates UI information (bitmap data, screen location, connection data, etc.) together with specific component settings in the form of attribute data. The element includes sub-elements indicating the loop type (air, plant, condenser) in which the component resides, the sub-loop type (supply, demand) and the plant loop type where relevant. The **AirLoopDuctType** sub-element specifies single/dual duct in the case of air loops.


attributes
ImageRectangle
ConnectingPlantLoopHandle
ConnectingAirLoopHandle
LoopType
SubLoopType
PlantLoopType
AirLoopDuctType
ComponentType
Location
ConnectionOffset
HVACComponent
Editable
1..∞
ZoneBranchFlag
Orientation
WaterInConnectionOrientation
WaterOutConnectionOrientation
AirInConnectionOrientation
AirOutConnectionOrientation
WaterInConnection
WaterOutConnection
AirInConnection
AirOutConnection
WaterInConnected
WaterOutConnected
AirInConnected
AirOutConnected
FanPlacement
WaterInConnectionCoordinate
WaterOutConnectionCoordinate
AirInConnectionCoordinate
AirOutConnectionCoordinate
Attributes
ZoneComponentAttributeList

HVAC Component Image Rectangle Element

In order to display an image of an HVAC component on the screen, each component has an associated image bitmap, the data for which is stored in the **ImageRectangle** sub-element of the **HVACComponent** element:



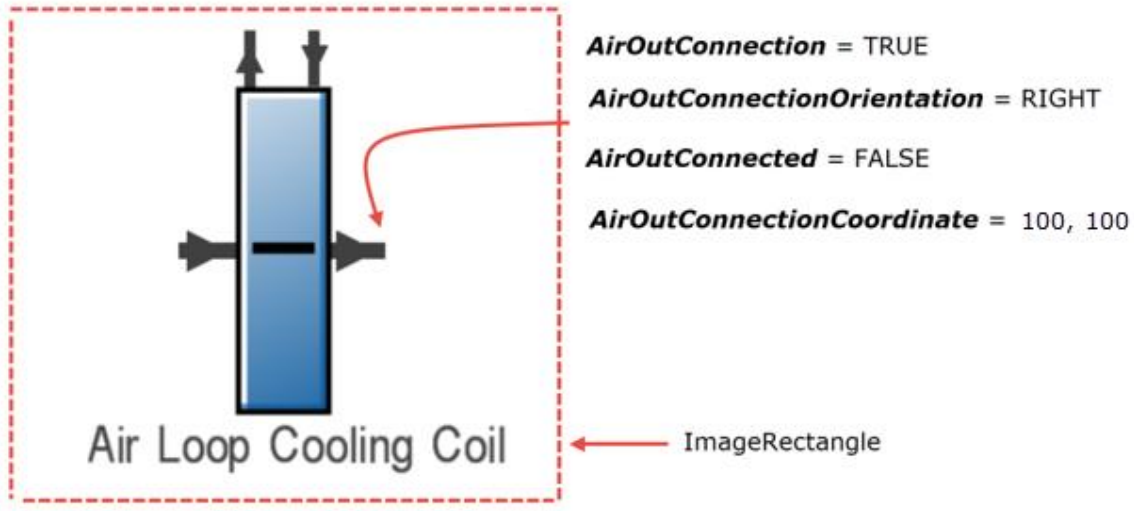
On startup, DesignBuilder HVAC loads a list of HVAC images (bitmaps) into an image vector. The image vector contains four separate images for each bitmap:

a) the HVAC component image itself,
b) a mask to enable overlay transparency,
c) a selected form of the image and
d) an inactive form of the image.

The **ImageTextureIndex**, **MaskTextureIndex**, **SelectedImageTextureIndex** and **InactiveTextureImageIndex** sub-elements point to the appropriate bitmaps in the image vector. The **ImageRectangle** contains Boolean elements indicating the presence of the various image types. The HVAC component image is rectangular and the **Vertices** sub-element contains the coordinates of corners of the rectangle.

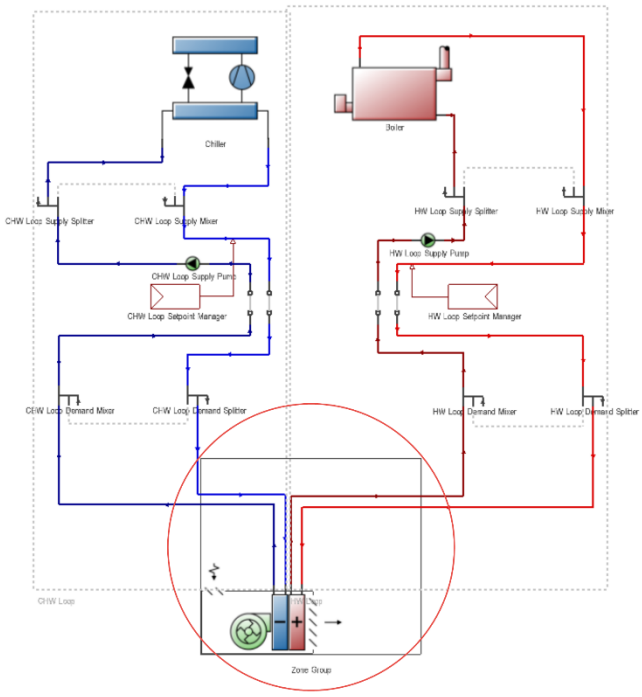
HVAC Component Connectivity Information

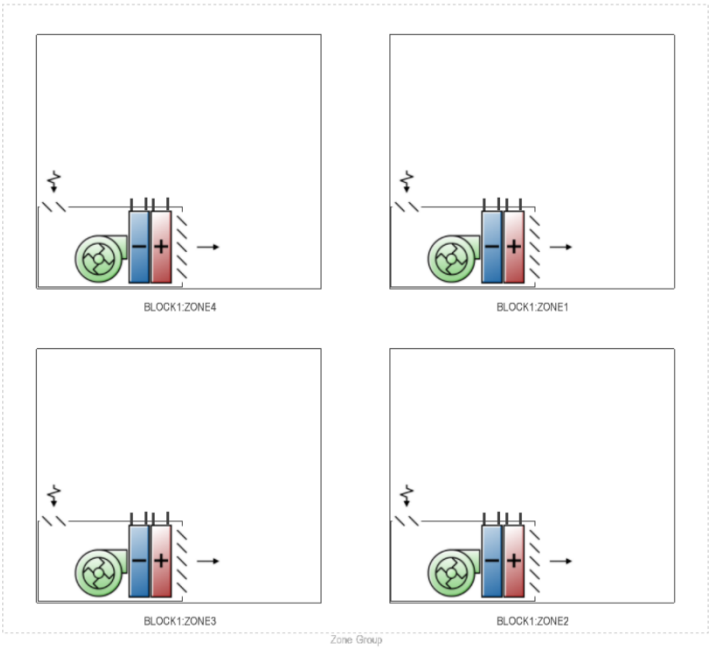
The **HVACComponent** element incorporates various sub-elements that define the connectivity of the component. These sub-elements include Boolean values indicating whether or not the component has air and water inlet and outlet connections (**AirInConnected**, etc.), the orientation of the air and water connections where relevant (whether the connection is orientated on the left, right, top or bottom of the image rectangle), the screen cartesian coordinates of the connections (used for connecting pipework/ductwork and establishing nodal connectivity) and lastly Boolean values indicating whether or not the connection is connected to pipework/ductwork.



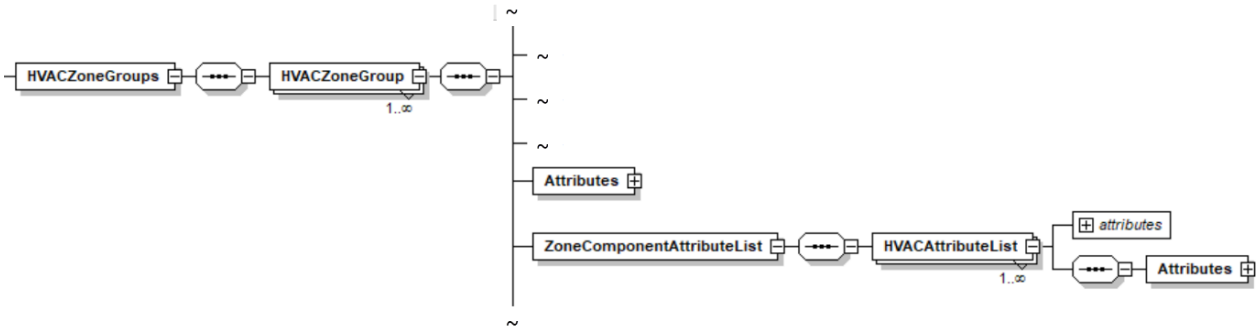
HVAC Zone Group

The **HVACZoneGroup** element is a bespoke DesignBuilder component and has no direct counterpart in EnergyPlus. The element houses a number of references to building zones, each of which contains identical items of HVAC equipment but where the equipment is of different ratings. This mechanism enables a much more compact and efficient way of representing and storing the data rather than using a separate HVAC zone for each building zone.





The mechanism uses a zone group attribute mechanism which is simply a list of attribute blocks where each attribute block is associated with the handle of the parent building zone. When an individual zone within a zone group is selected through the UI, the appropriate attribute block is loaded into the zone group attribute block, and then any attribute changes made via the UI are automatically assigned to the correct zone attribute block.



Optional Elements

Some elements can be omitted from the XML when exporting from a third party application. This section details elements that can be omitted and explains the consequences of any omission.

Optional Site Elements

The **Attributes** and **Tables** elements are currently required but these elements may be copied and pasted from a DesignBuilder XML template file. The **AssemblyLibrary** element may be dispensed with altogether but if so, the **AssemblyInstances** element must also be omitted from the **Building** and **BuildingBlock** elements. Note that at least one **Building** element must be included within the **Site Buildings** element.

Optional Building Elements

All of the following **Building** elements are optional and may be omitted altogether:

- **ComponentBlocks**
- **AssemblyInstances**
- **ProfileOutlines**
- **ConstructionLines**
- **BookmarkBuildings**
- **HVACNetwork**

The **Attributes** element may also be omitted in which case all of the building level settings will be inherited from the site level attribute data. Specific attributes may of course be included if required, e.g. a “Title” attribute could be included to assign a name to the building. Note that at least one **BuildingBlock** element must be included within the **Building BuildingBlocks** element.

Optional Building Block Elements

All of the following **BuildingBlock** elements are optional and may be omitted altogether:

- **ComponentBlocks**
- **AssemblyInstances**
- **ProfileOutlines**
- **CFDFans**

The **Attributes** element may also be omitted in which case all of the building block level settings will be inherited from the building level attribute data. Specific attributes may of course be included if required, e.g. a “Title” attribute could be included to assign a name to the building block.

The **VoidBodies** element may be omitted unless a courtyard or void is to be included within the building block.

Either the **ProfileBody** or **BaseProfileBody** must be included but not both. If the **BaseProfileBody** element is omitted, it will be simply copied from the **ProfileBody** element and vice versa.

The **Perimeter** element may be omitted in which case the perimeter will be automatically regenerated from the building block profile body and any included void bodies. Note that if the perimeter element is included, it must match the geometry of the profile body perimeter (and any void body perimeters as holes).

The **Zones** element may be omitted in which case the building block zones will be automatically regenerated from the profile body perimeter and any included partitions. If specific zone data or custom surface openings are to be included then the **Zones** element must be included and the perimeter geometry of any included zone must coincide with the geometry of the associated internal partitions and profile body perimeter segments. Note that if more than one zone is to be included within the building block, the **InternalPartitions** element must be included and any internal partitions together with associated building block perimeter segments must form closed loops representing the zone perimeters.