

## 8 Diagram Common

The Diagram Common (DC) package contains abstractions shared by the Diagram Interchange and the Diagram Graphics packages.

### 8.1 Overview

The Diagram Common (DC) package contains a number of common primitive types as well as structured data types that are used by the other DD packages, namely the Diagram Graphics(DI) package (Clause 9) and the Diagram Interchange(DG) package (Clause 10). The DC package itself does not depend on other packages.

The following sub clauses discuss common assumptions that are made by DC and the other DD packages.

#### 8.1.1 Measurement Unit

All coordinates and lengths defined by the DD packages are assumed to be in user units. A user unit is a value in the user coordinate system, which initially (before any transformation is applied) aligns with the device's coordinate system (for example, a pixel grid of a display). A user unit, therefore, represents a logical rather than physical measurement unit. Since some applications might specify a physical dimension for a diagram as well (mainly for printing purposes), a mapping from a user unit to a physical unit can be specified as a diagram's resolution. (Inch is chosen in this specification to avoid variability but tools can easily convert from/to other preferred physical units.) Resolution specifies how many user units fit within one physical unit (for example, a resolution of 300 specifies that 300 user units fit within 1 inch on the device).

#### 8.1.2 Coordinate System

This specification assumes a two-dimensional x-y coordinate system that has its origin at coordinate  $x=0, y=0$ . The x-axis is horizontal and its coordinate values increases to the right with negative coordinates allowed. Similarly, the y-axis is vertical and its coordinate values increases to the bottom with negative coordinates allowed.

---

**Issue: 16909 – DiagramElement ownership**

---

#### 8.1.3 Rotation

Rotations specified throughout this specification are made in degrees and can be positive (clock-wise) or negative (counter-clock-wise).

## 8.2 Abstract Syntax

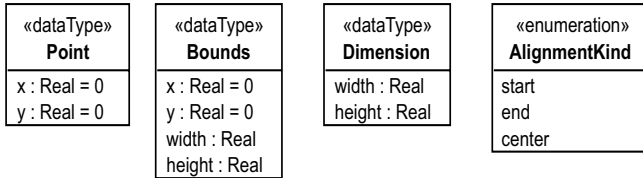
---

**Issue: 17453 – Use MOF Primitive Types**

---



**Figure 8.1 Dependencies of the DC package**



**Figure 8.2 - The layout data types**



**Figure 8.3 - The color data type**

## 8.3 Classifier Descriptions

### 8.3.1 AlignmentKind [Enumeration]

AlignmentKind enumerates the possible options for alignment for layout purposes.

#### Description

AlignmentKind enumerates the possible kinds for alignment for layout purposes (e.g., for text alignment within a bounding box).

## Diagrams

- Figure 8.2 (Layout Types)

## Literals

- start - an alignment to the start of a given length.
- end - an alignment to the end of a given length
- center - an alignment to the center of a given length

### 8.3.2 Bounds [DataType]

Bounds specifies a rectangular area in some x-y coordinate system that is defined by a location (x and y) and a size (width and height).

#### Description

Bounds is used to specify a rectangular area in some x-y coordinate system. The area is specified with a (x, y) location, representing the distance of the area's top-left corner from the origin, and a size (width and height) along the x-y axes.

## Diagrams

- Figure 8.2 (Layout Types)

## Attributes

- x : Real [1] = 0 - a real number ( $\geq 0$  or  $\leq 0$ ) that represents the x-coordinate of the bounds
- y : Real [1] = 0 - a real number ( $\geq 0$  or  $\leq 0$ ) that represents the y-coordinate of the bounds
- width : Real [1] - a real number ( $\geq 0$ ) that represents the width of the bounds
- height : Real [1] - a real number ( $\geq 0$ ) that represents the height of the bounds

## Constraints

- non\_negative\_size: the width and height of bounds cannot be negative  
[OCL] width  $\geq 0$  and height  $\geq 0$

### 8.3.3 Color [DataType]

Color is a data type that represents a color value in the RGB format.

#### Description

Color is used as a type for attributes that represent color. The color value is encoded using the RGB format as three separate integers in the range (0...255) representing the red, green, and blue components of the color. For example the color yellow is (red=255, green=255, blue=0).

## Diagrams

- Figure 8.3 (Color Type)

### Attributes

- red : Integer [1] - the red component of the color in the range (0..255)
- green : Integer [1] - the red component of the color in the range (0..255)
- blue : Integer [1] - the red component of the color in the range (0..255)

### Constraints

- valid\_rgb: the red, green, and blue components of the color must be in the range (0..255).  
[OCL] red >= 0 and red <=255 and green >= 0 and green <=255 and blue >= 0 and blue <=255

## 8.3.4 Dimension [DataType]

Dimension specifies two lengths (width and height) along the x and y axes in some x-y coordinate system.

### Description

Dimension is used to specify two lengths, a width along the x-axis and a height along the y-axis, in a x-y coordinate system.

### Diagrams

- Figure 8.2 (Layout Types)

### Attributes

- width : Real [1] - a real number (>=0) that represents a length along the x-axis.
- height : Real [1] - a real number (>=0) that represents a length along the y-axis.

### Constraints

- non\_negative\_dimension: the width and height of a dimension cannot be negative  
[OCL] width >= 0 and height >=0

## 8.3.5 KnownColor [Enumeration]

KnownColor is an enumeration of 17 known colors.

### Description

KnownColor enumerates 17 known colors, defined by the CSS specification, which are: aqua, black, blue, fuchsia, gray, green, lime, maroon, navy, olive, orange, purple, red, silver, teal, white, and yellow.

### Diagrams

- Figure 8.3 (Color Type)

### Literals

- maroon - a color with a value of #800000
- red - a color with a value of #FF0000

- orange - a color with a value of #FFA500
- yellow - a color with a value of #FFFF00
- olive - a color with a value of #808000
- purple - a color with a value of #800080
- fuchsia - a color with a value of #FF00FF
- white - a color with a value of #FFFFFF
- lime - a color with a value of #00FF00
- green - a color with a value of #008000
- navy - a color with a value of #000080
- blue - a color with a value of #0000FF
- aqua - a color with a value of #00FFFF
- teal - a color with a value of #008080
- black - a color with a value of #000000
- silver - a color with a value of #C0C0C0
- gray - a color with a value of #808080

### 8.3.6 Point [DataType]

A Point specifies a location in some x-y coordinate system.

#### Description

Point is used to specify a coordinate that is at a given distance (along the x and y axes) from the origin of some x-y coordinate system. The point (0, 0) is considered to be at the origin of that coordinate system. Coordinates increase towards the right of the x-axis and towards the bottom of the y-axis.

#### Diagrams

- Figure 8.2 (Layout Types)

#### Attributes

- x : Real [1] = 0 - a real number ( $\leq 0$  or  $\geq 0$ ) that represents the x-coordinate of the point.
- y : Real [1] = 0 - a real number ( $\leq 0$  or  $\geq 0$ ) that represents the y-coordinate of the point.

---

**Issue: 17453 – Use MOF Primitive Types**

---



# 9 Diagram Interchange

The Diagram Interchange (DI) package enables interchange of graphical information that language users have control over, such as position of nodes and line routing points. Language specifications specialize elements of DI to define diagram interchange elements for a language.

## 9.1 Overview

---

### Issue: 16628 – DiagramElements cannot represent multiple model Elements

---

The Diagram Interchange (DI) package contains a number of types used in the definition of diagram interchange models. The package imports the Diagram Common package (Clause 8) and MOF, as shown in Figure 9.1, that contains various relevant data types. The DI package contains many abstract types for extension and refinement by concrete types in domain-specific DI packages. DI is a framework meant for extension rather than a component ready to be used out of the box. It provides typically needed diagram interchange information for customized DI models in specific graphical domains.

## 9.2 Abstract Syntax

---

### Issue: 16628 – DiagramElements cannot represent multiple model Elements

---

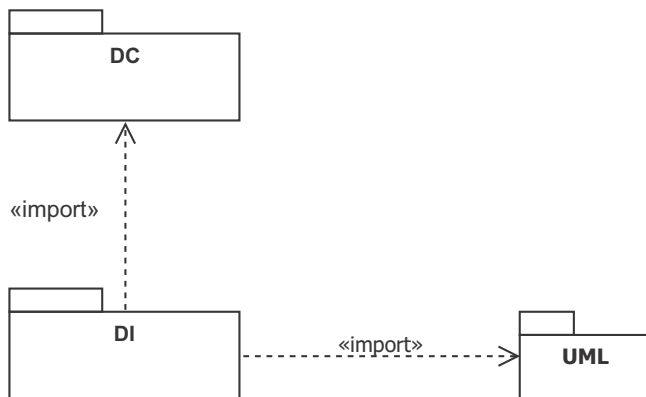


Figure 9.1 - Dependencies of the DI package

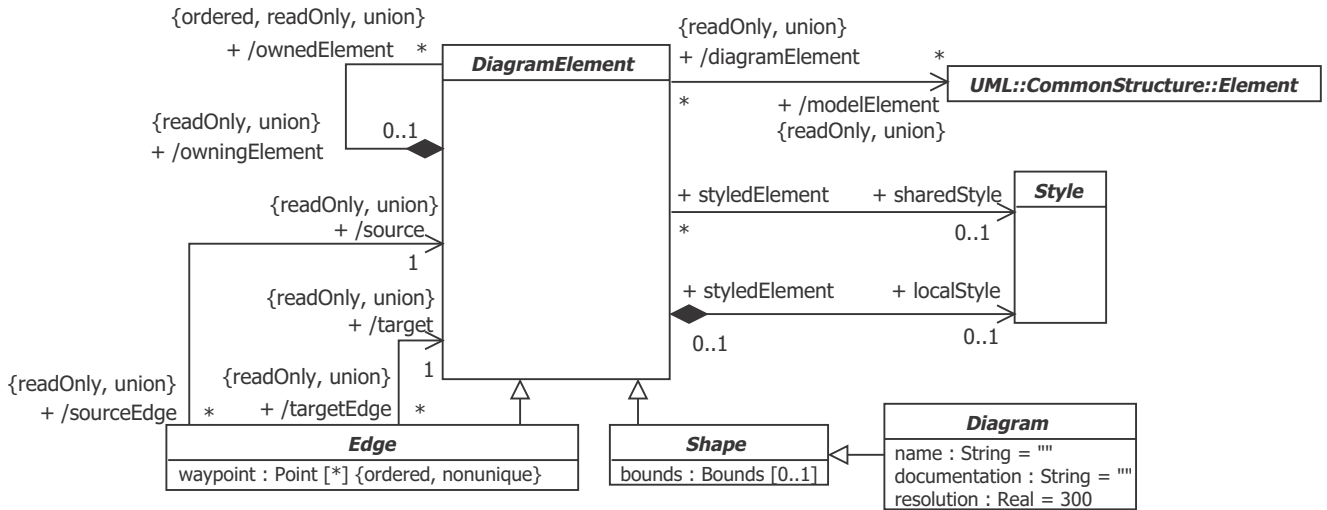


Figure 9.2 - Diagram Element

## 9.3 Classifier Descriptions

### 9.3.1 Diagram [Class]

Diagram is an abstract container of a graph of diagram elements. Diagrams are diagram elements with an origin point in the x-y coordinate system. Their elements are laid out relative to their origin point.

#### Description

A diagram does not need to be nested. It can be persisted in the same resource as the abstract syntax model or in a different resource. It can also be owned by elements of the abstract syntax model, or by no element at all (like being the root of the resource).

A diagram represents a two dimensional x-y coordinate system that is used to layout nested and inter-connected diagram elements. A diagram has an origin point (0, 0) on the x and y. The coordinate system of a diagram increases along the x-axis from left to right and along the y-axis from top to bottom. All the nested diagram elements are laid out relative to their nesting diagram's origin.



As a kind of diagram element, a diagram may reference model elements from an abstract syntax model, in which case the whole diagram is considered a depiction of those elements (e.g., an activity diagram is a depiction of a UML activity). Alternatively, a diagram without such a reference is simply a layout container for its diagram elements (e.g., a class diagram is a container for UML class shapes and edges).

A diagram can have a name and a documentation. This information is not shown as part of the rendering of the diagram itself but can be used in an application to label a diagram (e.g., “DI Package Diagram”) in a browser and show its intent (e.g., “A diagram that shows the classes of the DI package”).

A diagram also specifies a resolution expressed in units per inch. The resolution specifies the conversion ratio between the logical units used by the diagram and a unit of physical measurement (an inch in this case). For example, a resolution value of 300 specifies that every 300 logical unit of length map to an inch. The resolution value is mainly used when printing diagrams or when rendering diagrams on display in their physical size.

Styles contain combinations of style property values used by different elements across the diagram. This allows a large number of elements in a diagram to reference a small number of styles, which would dramatically reduce a diagram’s footprint.

## Diagrams

- Figure 9.2 (DI Package)

## Generalizations

- Shape [Abstract Class]

## Attributes

- name : String [1] = “” - the name of the diagram.
- documentation : String [1] = “” - the documentation of the diagram.
- resolution : Real [1] = 300 - the resolution of the diagram expressed in user units per inch.

### 9.3.2 DiagramElement [Abstract Class]

DiagramElement is the abstract super type of all elements in diagrams, including diagrams themselves. When contained in a diagram, diagram elements are laid out relative to the diagram’s origin.

## Description

---

### Issue: 16628 – DiagramElements cannot represent multiple model Elements

---

A diagram element can be useful on its own (i.e., purely notational) or more commonly used as a depiction of other MOF-based elements from an abstract syntax model (like a UML model). In the latter case, the diagram element references the depicted model elements and defines notational properties for those elements. An example of a depicting diagram element is a Class shape on a UML diagram that specifies the bounds of the class, its colors, its compartments...etc. An example of a purely notational diagram element is a Note shape on a UML diagram that provides a textual description of part of the diagram. The diagram element’s reference to model elements is defined abstractly as derived union to allow language-specific extensions of DI to refine it further to suit their purposes (like specializing its type).

---

### Issue: 16909 – DiagramElement ownership

---

A diagram element can own other diagram elements in a tree-like hierarchy. The collection of owned elements is defined abstractly as a derived union to allow language-specific extensions of DI to define the allowed topologies for their diagram elements (e.g., a UML class shape can own UML compartments). This collection is ordered, to specify z-order for owned elements. Diagram elements may overlap in some situations (their renderings may intersect), in which case it is important to determine which ones appear below or more hidden (have lower z-order) and which ones appear above or more visible (have higher z-order). Z-order of owned diagram elements is determined as follows:

- Owned diagram elements are higher in z-order than their owning diagram elements.
- Diagram elements that appear earlier in the ordered collection have higher z-order than those that appear later.

More specialized diagram element types define properties that characterize their nature. However, a subset of those properties is stylistic in nature and tends to have similar values across many diagram elements. Examples of such properties are fill properties, stroke properties, and font properties. To minimize the footprint of diagram interchange models, those stylistic properties are not defined on diagram elements directly but are rather defined on Style elements that can be owned and/or shared by diagram elements. Shared style elements are owned by other elements, which might be packaging elements in the language incorporating diagram interchange. Style property values are calculated based on a well-defined algorithm given in “Style [Abstract Class]” on page 20.

## Diagrams

- Figure 9.2 (DI Package)

## Specializations

- Edge [Abstract Class]
- Shape [Abstract Class]

## Association Ends

---

**Issue: 16628 – DiagramElements cannot represent multiple model Elements**

---

---

**Issue: 16909 – DiagramElement ownership**

---

- `/modelElement` : Element [\*] {readOnly, union} - a reference to a depicted model elements, which can be any MOF-based element.
- `/owningElement` : DiagramElement [0..1] {readOnly, union} - a reference to the diagram element that directly owns this diagram element.
- `♦/ownedElement` : DiagramElement [\*] {readOnly, union, ordered} - an ordered collection of diagram elements that are directly owned by this diagram element.
- `♦ localStyle` : Style [0..1] - a reference to an optional locally-owned style for this diagram element.
- `sharedStyle` : Style [0..1] - a reference to an optional shared style element for this diagram element.

### 9.3.3 Edge [Abstract Class]

Edge is a diagram element that renders as a polyline, connecting a source diagram element to a target diagram element, and is positioned relative to the origin of the diagram.

## Description

Edge represents a diagram element defined with a sequence of connected waypoints forming a polyline that connects two diagram elements: a source element and a target element (could be the same as the source as in self connection). The waypoints are positioned relative to the origin of the nesting diagram, specifying a route for the polyline on the diagram.

An edge can be purely notational, i.e., does not reference any model element. An example is the line attaching a comment to a UML element. On the other hand, an edge can be a depiction of a relational element from an abstract syntax model. Examples include UML generalization edge or a BPMN message flow edge. In that case, the edge's source and target reference diagram elements depicting the relationship's source and target elements (or its two related elements if the relationship is not directed) respectively. The edge's source and target references are defined abstractly as derived unions. In an extending language-specific DI metamodel, these references need to be refined. In case the source and target references can be derived unambiguously from the model element, the properties can be redefined with that derivation logic. Otherwise, the properties can be specialized with concrete settable properties.

## Diagrams

- Figure 9.2 (DI Package)

## Generalizations

- DiagramElement [Abstract Class]

## Attributes

- waypoint : Point [\*] {ordered, nonunique} - an optional list of points relative to the origin of the nesting diagram that specifies the connected line segments of the edge.

## Association Ends

- /source : DiagramElement [1] {readOnly, union} - the edge's source diagram element, i.e., where the edge starts from.
- /target : DiagramElement [1] {readOnly, union} - the edge's target diagram element, i.e., where the edge ends at.

### 9.3.4 Shape [Abstract Class]

Shape is a diagram element with given bounds that is laid out relative to the origin of the diagram.

## Description

Shape is an abstract class that is expected to be further sub classed in a language-specific DI metamodel.

A shape can be purely notational, i.e., does not reference any model element. Examples include a note shape on a UML class diagram with some text describing the diagram and an overlay shape with some semi-transparent fill enclosing a bunch of shapes on the diagram to make them stand out. On the other hand, a shape can be a depiction of a component (non-relational) element from an abstract syntax model. Examples include a UML class shape and a BPMN activity shape.

## Diagrams

- Figure 9.2 (DI Package)

## Generalizations

- DiagramElement [Abstract Class]

## Specializations

- Diagram [Class]

## Attributes

- bounds : Bounds [0..1] - the optional bounds of the shape relative to the origin of its nesting plane.

### 9.3.5 Style [Abstract Class]

Style contains formatting properties that affect the appearance or style of diagram elements, including diagram themselves.

#### Description

A Style is a set of properties (e.g., `fontName`, `fillColor` or `strokeWidth`) that affect the appearance or style of diagram elements rather than their intrinsic semantics. Style is defined as an abstract class without prescribing any style properties to leave it up to language-specific DI extensions to define concrete style classes with their own properties that are applicable to their diagram element types.

A style element can either be local to (owned by) a diagram element or shared between (referenced by) several diagram elements, in which case it is owned elsewhere (e.g., by packaging elements in the language incorporating diagram interchange). A value set to a local style property in a diagram element overrides one that is set to the same property on a shared style referenced by the same diagram element.

Style properties are typically defined as optional to allow the state of “unset” to be legal. This is needed to implement cascading style, where an unset style property in one diagram element gets its value from the closest diagram element in its owning element chain that has a value set for that property.

The above semantics effectively specify that a value for a style property is based on the following mechanisms (in order of precedence):

- if there is a cascading value set on a local style, use it.
- Otherwise, if there is a cascading value set on a shared style, use it.
- Otherwise, if a cascading value is available from a diagram element in the owning element chain, use it from the closest owning element.
- Otherwise, use the style property’s default value.

#### Diagrams

- Figure 9.2 on page 16 (DI Package)

# 10 Diagram Graphics

The Diagram Graphics (DG) package contains a model of graphical primitives that can be instantiated when mapping from a language abstract syntax models and diagram interchange (DI) models to visual presentations. The mapping effectively defines the concrete syntax of a language. This specification does not restrict how the mappings are done, or what languages are used to define them.

## 10.1 Overview

The Diagram Graphics (DG) package provides a technology and platform independent model of two-dimensional graphical information that can be instantiated in a mapping from abstract syntax models and diagram interchange (DI) models of a given modeling language, effectively defining the concrete graphical syntax of the language. This specification does not restrict how mappings are done or what mapping languages are used to define them (QVT is one such mapping language).

In addition to mapping information contained in the interchanged models (abstract syntax and DI models) that users have control over, the mapping can also specify the aspects of the concrete syntax that users do not have a control over, such as specific geometric shapes and line styles that are fixed (made normative) by language specifications. This information is not interchanged, because it is the same in all diagrams conforming to the language.

The design of the DG package borrows to a good degree from the Scalable Vector Graphics (SVG) specification and other relevant specifications. This is done to ease the mapping of DG to existing industry standard graphical packages. DG imports the Diagram Common package (Clause 8), as shown in Figure 10.1, that contains relevant data types used by DG.

## 10.2 Abstract Syntax

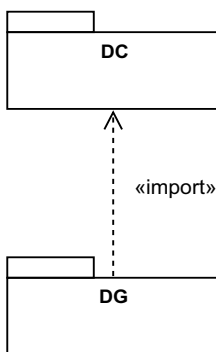


Figure 10.1 - Dependencies of the DG package

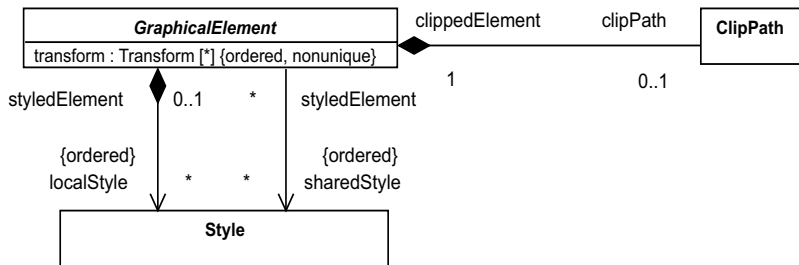


Figure 10.2 - Graphical Element

Issue: 18675 – MarkedElement should be abstract in provided CMOF files

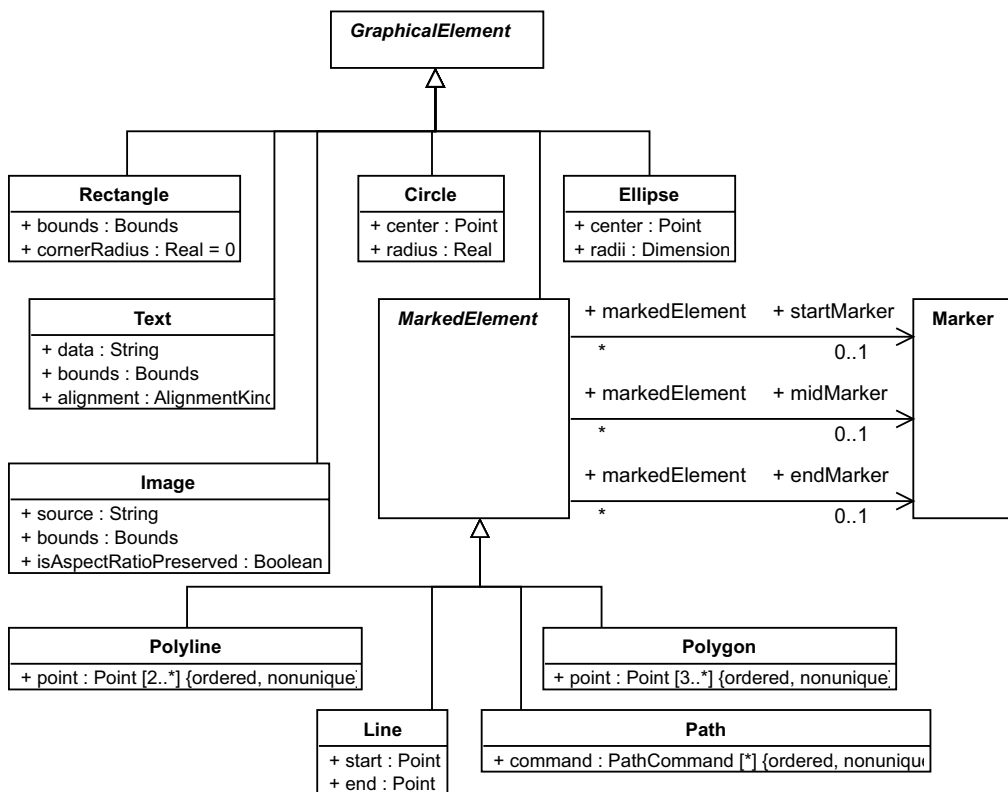


Figure 10.3 - Primitive Elements

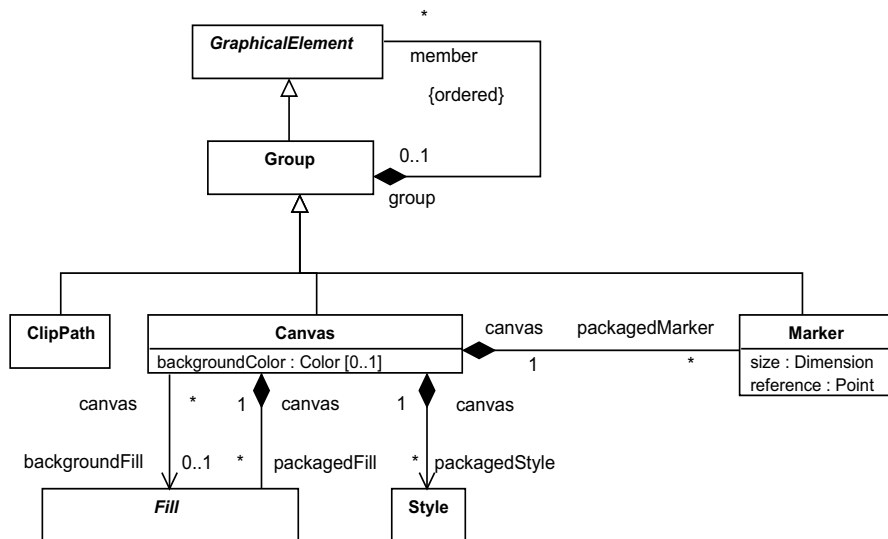


Figure 10.4 - Group Elements

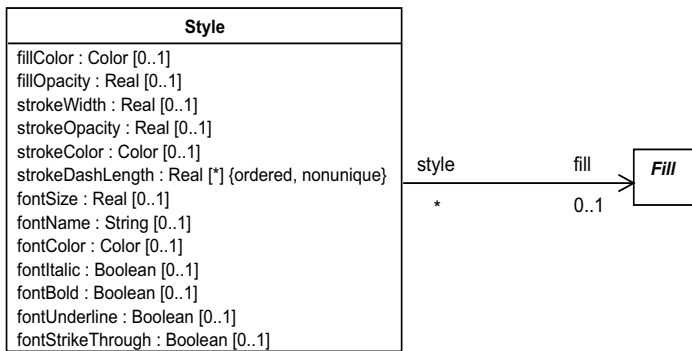


Figure 10.5 - Style

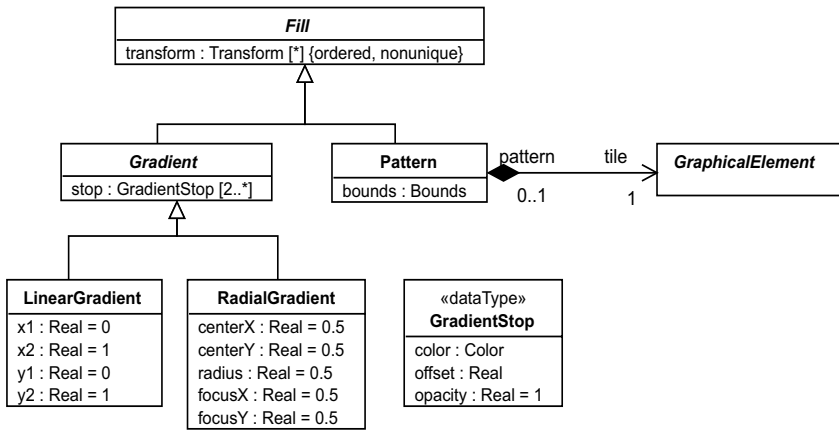


Figure 10.6 - Fills

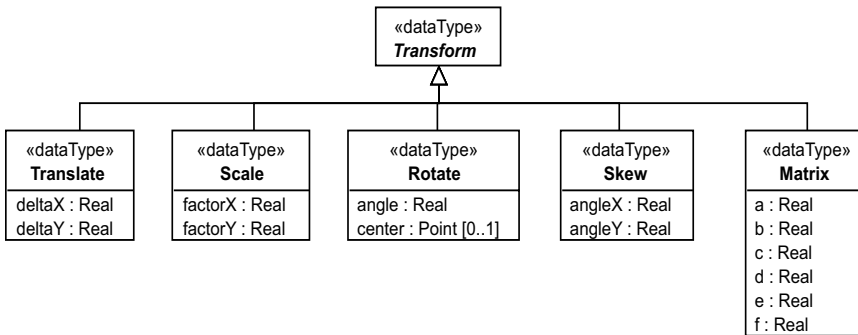


Figure 10.7 - Transforms



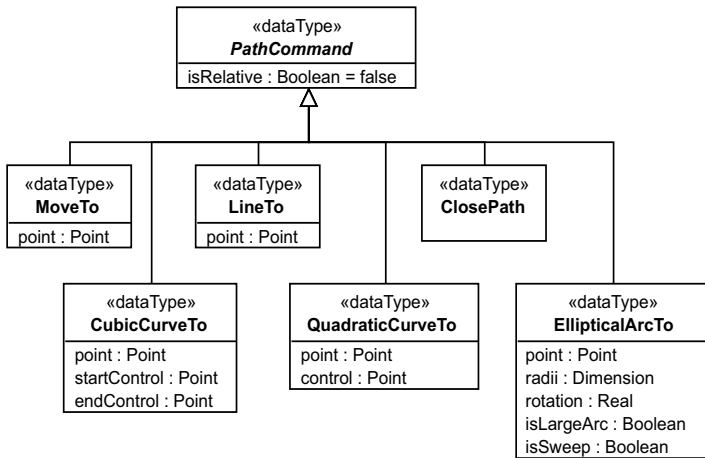


Figure 10.8 - Path Commands

## 10.3 Classifier Descriptions

### 10.3.1 Canvas [Class]

Canvas is a kind of group that represents the root of containment for all graphical elements that render one diagram.

#### Description

Canvas is a kind of group that is used as a root container of a hierarchy of graphical elements used to render the same diagram. A canvas has a two-dimensional x-y coordinate system with a (x=0, y=0) origin point and an infinite size. The coordinate system increases along the x-axis from left to right and along the y-axis from top to bottom, with negative coordinates allowed. The coordinates of graphical elements nested in the canvas member hierarchy are relative to the origin of the canvas. Unlike a group, a canvas has a visual manifestation in the form of a background that can be filled separately from its member elements.

#### Diagrams

- Figure 10.4 (Group Elements)

#### Generalizations

- Group [Class]

#### Attributes

- backgroundColor : Color [0..1] - a color that is used to paint the background of the canvas itself. A backgroundColor value is exclusive with a backgroundFill value. If both are specified, the backgroundFill value is used. If none is specified, no fill is applied (i.e., the canvas becomes see-through).

## Association Ends

- `backgroundFill` : `Fill` [0..1] - a reference to a fill that is used to paint the background of the canvas itself. A `backgroundFill` value is exclusive with a `backgroundColor` value. If both are specified, the `backgroundFill` value is used. If none is specified, no fill is applied (i.e., the canvas becomes see-through).
- ♦ `packagedFill` : `Fill` [\*] - a set of fills packaged by the canvas and referenced by graphical elements in the canvas.
- ♦ `packagedMarker` : `Marker` [\*] - A set of markers packaged by the canvas and referenced by marked elements in the canvas.
- ♦ `packagedStyle` : `Style` [\*] - a set of styles packaged by the canvas and referenced by graphical elements in the canvas as shared styles.

### 10.3.2 Circle [Class]

Circle is a graphical element that defines a circular shape with a given center point and a radius.

#### Description

Circle is a graphical element that renders as a circle shape with a given center point and a radius in the x-y coordinate system.

#### Diagrams

- Figure 10.3 (Primitive Elements)

#### Generalizations

- `GraphicalElement` [Abstract Class]

#### Attributes

- `center` : `Point` [1] - the center point of the circle in the x-y coordinate system.
- `radius` : `Real` [1] - a real number ( $\geq 0$ ) that represents the radius of the circle.

#### Constraints

- `non_negative_radius`: the radius cannot be negative  
[OCL] `radius >= 0`

### 10.3.3 ClipPath [Class]

ClipPath is a kind of group whose members collectively define a painting mask for its referencing graphical elements.

#### Description

ClipPath represents a special kind of group element that is owned by a graphical element to define its clipping mask (or stencil). A clip path does not render as a normal graphical element but is only used to specify the regions that can be painted in its owning element.

The raw geometry of each member element (exclusive of its style) of a clip path conceptually defines a 1-bit mask, which represents the silhouette of the graphics associated with that element. Anything outside the outline of the element is masked out. When the clip path contains multiple member elements, their silhouettes are logically OR'ed together to create a single silhouette, which is then used to restrict the region onto which paint can be applied. Thus, a point is inside a clip path if it is inside any of the member elements of the clip path.

The following are more rules that affect the calculation of the final clipping path of a graphical element:

- For a given graphical element, the final clipping path is defined by intersecting its owned clip path, if any, with any clip paths owned by any elements in its group chain.
- If a clip path owns itself another clip path element, the resulting clipping path is the intersection of the two.
- If any member element of the clip path owns another clip path, the given member is clipped by its own clip path first before OR'ing its silhouette with the silhouettes of the other members.

The coordinate system of the clip path is the same as the one used by its owner (e.g., if the coordinates of the owner is relative to the canvas, the coordinates of its clip path is also relative to the canvas). In addition, any transforms that are defined on the graphical element are also applied to the clip path.

## Diagrams

- Figure 10.2 (Graphical Element)
- Figure 10.4 (Group Elements)

## Generalizations

- Group [Class]

## Association Ends

- `clippedElement` : `GraphicalElement` [1] - a reference to the owning element that is clipped by this clip path.

### 10.3.4 ClosePath [DataType]

`ClosePath` is a kind of path command that ends the current subpath and causes an automatic straight line to be drawn from the current point to the initial point of the current subpath.

## Description

`ClosePath` is a kind of path command that ends the current subpath and causes an automatic straight line to be drawn from the current point to the initial point of the current subpath. If a `ClosePath` command is followed immediately by a `MoveTo` command, then the `MoveTo` identifies the start point of the next subpath. If a `ClosePath` is followed immediately by any other command, then the next subpath starts at the same initial point as the current subpath.

## Diagrams

- Figure 10.8 (Path Commands)

## Generalizations

- `PathCommand` [Abstract DataType]

### 10.3.5 CubicCurveTo [DataType]

CubicCurveTo is a kind of path command that draws a cubic bézier curve from the current point to a new point using a start and an end control points.

#### Description

CubicCurveTo is a kind of path command that draws a cubic bézier curve from the current point to a new point using two control points: startControl and endControl. Multiple CubicCurveTo commands may be specified in a row to draw a polybézier. At the end of the command, the provided point becomes the new current point in the coordinate system. Examples of cubic bézier curves are shown in Figure 10.9.

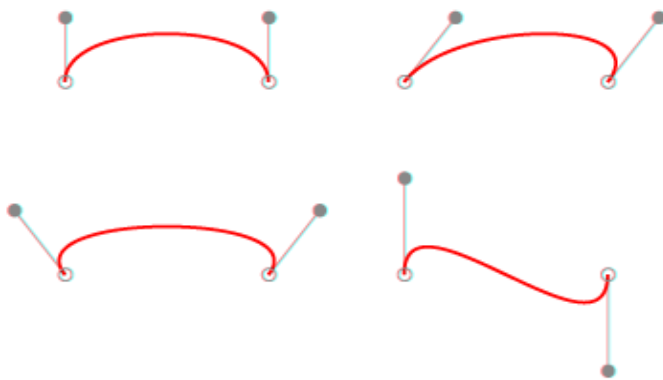


Figure 10.9 - Examples of cubic bézier curves

#### Diagrams

- Figure 10.8 (Path Commands)

#### Generalizations

- PathCommand [Abstract DataType]

#### Attributes

- point : Point [1] - a point to draw a cubic bézier curve to/from the current point in the coordinate system.
- startControl : Point [1] - the start control point of the cubic bézier curve.
- endControl : Point [1] - the end control point of the cubic bézier curve.

### 10.3.6 Ellipse [Class]

Ellipse is a graphical element that defines an elliptical shape with a given center point and two radii on the x and y axes.

#### Description

Ellipse is a graphical element that renders as an ellipse shape with a given center point and two radii in the x-y coordinate system.

## Diagrams

- Figure 10.3 (Primitive Elements)

## Generalizations

- GraphicalElement [Abstract Class]

## Attributes

- center : Point [1] - the center point of the ellipse in the x-y coordinate system.
- radii : Dimension [1] - a dimension that specifies the two radii of the ellipse (a width along the x-axis and a height along the y-axis)

### 10.3.7 EllipticalArcTo [DataType]

EllipticalArcTo is a kind of path command that draws an elliptical arc from the current point to a new point in the coordinate system.

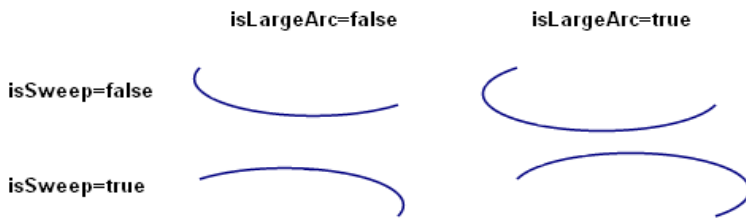
#### Description

EllipticalArcTo is a kind of path command that draws an elliptical arc from the current point to a new point in the coordinate system. The EllipticalArcTo command is also specified two radii, a rotation and two flags (isLargeArc flag and isSweep). The rotation is used to rotate the ellipse that the arc is created from. This rotation maintains (does not move) the start and end points, as shown in Figure 10.10.



**Figure 10.10 - Elliptical arc rotation**

The two flags control which part (sweep) of the ellipse is used to cut the arc, as shown in Figure 10.11. These are needed because there are four possible arcs, based on the arc sweep angle and direction, that can be specified between the same start and end points.



**Figure 10.11 - Elliptical arc sweeps**

### Diagrams

- Figure 10.8 (Path Commands)

### Generalizations

- PathCommand [Abstract DataType]

### Attributes

- point : Point [1] - a point to draw an elliptical arc to/from the current point in the coordinate system.
- radii : Dimension [1] - the two radii of the ellipse from which the arc is created.
- rotation : Real [1] - a real number representing a rotation (in degrees) of the ellipse from which the arc is created.
- isLargeArc : Boolean [1] - whether the arc sweep is equal to or greater than 180 degrees (the large arc).
- isSweep : Boolean [1] - whether the arc is drawn in a positive-angle direction.

### 10.3.8 Fill [Abstract Class]

Fill is the abstract super class of all kinds of fills that are used to paint the interior of graphical elements.

#### Description

Fill defines a paint that can be used to fill enclosed areas of graphical elements. A fill is owned by a canvas and is referenced by graphical elements in the canvas. A fill can also be transformed (translated, scaled, rotated, or skewed) with a sequence of transforms (see “Translate [DataType]” on page 47 for more details).

#### Diagrams

- Figure 10.4 (Group Elements)
- Figure 10.5 (Style)
- Figure 10.6 (Fills)

#### Specializations

- Pattern [Class]
- Gradient [Abstract Class]

## Attributes

- transform : Transform [\*] {ordered, nonunique} - a list of zero or more transforms to apply to this fill.

## Association Ends

- canvas : Canvas [1] - a reference to the canvas that owns this fill.

### 10.3.9 Gradient [Abstract Class]

Gradient is a kind of fill that paints a continuously smooth color transition along the gradient range from one color to the next.

#### Description

Gradient is the abstract super class of kinds of fill that paint a continuously smooth color transition from one color to another, possibly followed by additional transitions to other colors. The range of colors to use on a gradient is defined by GradientStop (see 10.3.11) that are nested by the gradient. Every stop defines a main color transition, its offset and opacity. The exact semantics of the stop offset is defined by the gradient sub classes.

#### Diagrams

- Figure 10.6 (Fills)

#### Generalizations

- Fill [Abstract Class]

#### Specializations

- RadialGradient [Class]
- LinearGradient [Class]

#### Attributes

- stop : GradientStop [2..\*] - a list of two or more gradient stops defining the color transitions of the gradient.

### 10.3.10 GradientStop [DataType]

GradientStop defines a color transition along the distance from a gradient's start to its end offsets.

#### Description

GradientStop represents a color transition for a gradient. Two or more stops are owned by a gradient. Each gradient stop defines a color, an opacity and an offset. The offset is a ratio that indicates where the gradient stop is placed. For linear gradients, the offset represents a ratio along the gradient vector. For radial gradients, it represents a ratio from the focus point (0%) to the edge of the largest (outermost) circle (100%).

#### Diagrams

- Figure 10.6 (Fills)

## Attributes

- color : Color [1] - the color to use at this gradient stop.
- offset : Real [1] - a real number ( $\geq 0$  and  $\leq 1$ ) representing the offset of this gradient stop as a ratio of the distance between the start and end positions of the gradient.
- opacity : Real [1] = 1 - a real number ( $\geq 0$  and  $\leq 1$ ) representing the opacity of the color at the stop. A value of 0 means totally transparent, while a value of 1 means totally opaque.

## Constraints

- valid\_offset: the offset must be between 0 and 1.  
[OCL] offset $\geq 0$  and offset $\leq 1$
- valid\_opacity: the opacity must be between 0 and 1.  
[OCL] opacity $\geq 0$  and opacity $\leq 1$

### 10.3.11 GraphicalElement [Abstract Class]

GraphicalElement is the abstract superclass of all graphical elements that can be nested in a canvas.

#### Description

GraphicalElement represents a unit of graphical information that is used to build the concrete graphical syntax of modeling languages. It is the abstract super class of all graphical elements and can be nested by (a member of) a Group element and organized in a hierarchy rooted with a Canvas element (a special kind of Group).

A graphical element can reference local and/or shared styles that define how the element is styled (formatted). When a style property is not set by either local (higher precedence) or shared (lower precedence) styles, it gets its value from the closest group element that provides a value for this style property (see “Text [Class]” on page 46 for more details). If the same style property is set by multiple local styles or multiple shared styles, then the property on the style earlier in the ordering has higher precedence.

The geometry of a graphical element can optionally be transformed using a sequence of transforms (see “Translate [DataType]” on page 47). Those transforms apply on top of other transforms defined on elements in the group chain of the element. The result of applying those transforms, in sequence, defines the final state of geometry for a graphical element.

A graphical element can optionally own a clip path element to restrict its regular painting with a mask defined by the clip path. A clip path is a group of graphical elements, whose collective geometry define a mask to restrict the painting of graphical elements. Refer to “Transform [Abstract DataType]” on page 47 for more details.

#### Diagrams

- Figure 10.2 (Graphical Element)
- Figure 10.3 (Primitive Elements)
- Figure 10.4 (Group Elements)
- Figure 10.6 (Fills)



## Specializations

---

### Issue: 18675 – MarkedElement should be abstract in provided CMOF files

---

- MarkedElement [Abstract Class]
- Circle [Class]
- Ellipse [Class]
- Image [Class]
- Rectangle [Class]
- Group [Class]
- Text [Class]

## Attributes

- transform : Transform [\*] {ordered, nonunique} - a list of zero or more transforms to apply to this graphical element.

## Association Ends

- group : Group [0..1] - the group element that owns this graphical element.
- ♦ localStyle : Style [\*] {ordered} - a list of locally-owned styles for this graphical element.
- sharedStyle : Style [\*] {ordered} - a list of shared styles for this graphical element.
- ♦ clipPath : ClipPath [0..1] - an optional reference to a clip path element that masks the painting of this graphical element.

### 10.3.12 Group [Class]

Group defines a group of graphical elements that can be styled, clipped, and/or transformed together.

## Description

---

### Issue: 18679 – Styling capabilities of Canvas ambiguous

---

Group is a graphical element that applies common styles, transforms, and/or clip paths to its member elements. Specialized groups can introduce visual representations, for example see “Canvas [Class]” on page 23, but Group does not define any itself. The (local or shared) styles defined on a group apply to its member elements, not to the group (see “Style [Class]” on page 44 for more details). Similarly, the transforms defined on a group element are applied to its member elements (see “Translate [DataType]” on page 47 for more details). Additionally, a clip path defined on a group element applies to its member elements (see “Transform [Abstract DataType]” on page 47 for more details).

---

### Issue: 16909 – DiagramElement ownership

---

The collection of members is ordered, to specify z-order for owned elements. Graphical elements may overlap in some situations (their renderings may intersect), in which case it is important to determine which ones appear below or more hidden (have lower z-order) and which ones appear above or more visible (have higher z-order). Z-order of owned graphical elements is determined as follows:

- Owned graphical elements are higher in z-order than their owning groups.
- Graphical elements that appear earlier in the ordered collection have higher z-order than those that appear later.

## Diagrams

- Figure 10.4 (Group Elements)

## Generalizations

- GraphicalElement [Abstract Class]

## Specializations

- ClipPath [Class]
- Marker [Class]
- Canvas [Class]

## Association Ends

- ♦ member : GraphicalElement [\*] {ordered} - the list of graphical elements that are members of (owned by) this group.

### 10.3.13 Image [Class]

Image is a graphical element that defines a shape that paints an image with a given URL within given bounds.

## Description

Image is a graphical element that renders a referenced image file (with a given URL) within a given bounding box in the x-y coordinate system. Image can refer to a raster image file (e.g., a PNG or a JPEG file) or to an SVG file (i.e., one with MIME type of “image/svg+xml”). The original size of a raster image is defined by the image data, while the original size of a SVG image is given by the ‘viewBox’ attribute on the outermost SVG element.

When an image is rendered, the top-left of the image is aligned with the top-left of the bounding box. When the original size of the image matches the size of the bounding box, the image is rendered with that original size. When the two sizes are different, the image is scaled as large as possible to fit within the bounding box. The scale factors for the width and height depend on the value of the isAspectRatioPreserved flag. When the flag is set to true, the scale factors for both width and height of the image are the same. Otherwise, the scale factors may be calculated differently such that the bottom-right of the image exactly aligns with bottom-right of the bounding box. If the image’s original size cannot be retrieved (e.g., if the ‘viewBox’ attribute is not set in the SVG file), the isAspectRatioPreserved flag is assumed to be false.

## Diagrams

- Figure 10.3 (Primitive Elements)

## Generalizations

- GraphicalElement [Abstract Class]

## Attributes

- source : String [1] - the URL of a referenced image file.

- bounds : Bounds [1] - the bounds within which the image is rendered.
- isAspectRatioPreserved : Boolean [1] - whether to preserve the aspect ratio of the image upon scaling, i.e. the same scale factor for width and height.

### 10.3.14 Line [Class]

Line is a marked element that defines a shape consisting of one straight line between two points.

#### Description

Line is a marked element that renders as a straight line between two points, a start and an end, in the x-y coordinate system.

#### Diagrams

- Figure 10.3 (Primitive Elements)

#### Generalizations

---

**Issue: 18675 – MarkedElement should be abstract in provided CMOF files**

---

- MarkedElement [Abstract Class]

#### Attributes

- start : Point [1] - the starting point of the line in the x-y coordinate system.
- end : Point [1] - the ending point of the line in the x-y coordinate system.

### 10.3.15 LineTo [DataType]

LineTo is a kind of path command that draws a straight line from the current point to a new point.

#### Description

LineTo is a kind of path command that draws a straight line from the current point to a new point. The effect is as if a pen was pressed and moved to a new location in a straight line. Multiple LineTo commands can be specified in a row to draw a polyline. At the end of the command, the provided point becomes the new current point in the coordinate system.

#### Diagrams

- Figure 10.8 (Path Commands)

#### Generalizations

- PathCommand [Abstract DataType]

#### Attributes

- point : Point [1] - a point to draw a straight line to/from the current point in the coordinate system.

### 10.3.16 LinearGradient [Class]

LinearGradient is a kind of gradient that fills a graphical element by smoothly changing color values along a vector.

#### Description

LinearGradient is a kind of gradient that fills a graphical element by smoothly changing color values between gradient stops along a vector. The vector is defined by start and end positions expressed as ratios ( $x1$ ,  $x2$ ) of the width of the element and start and end positions expressed as ratios ( $y1$ ,  $y2$ ) of the height of the element.

Linear gradients can be defined as horizontal, vertical, or angular gradients:

- Horizontal gradients are created when  $y1$  and  $y2$  are equal and  $x1$  and  $x2$  differ.
- Vertical gradients are created when  $x1$  and  $x2$  are equal and  $y1$  and  $y2$  differ.
- Angular gradients are created when  $x1$  and  $x2$  differ and  $y1$  and  $y2$  differ.

#### Diagrams

- Figure 10.6 (Fills)

#### Generalizations

- Gradient [Abstract Class]

#### Attributes

- $x1$  : Real [1] = 0 - a real number ( $\geq 0$  and  $\leq 1$ ) representing a ratio of the graphical element's width that is the x start point of the gradient.
- $x2$  : Real [1] = 1 - a real number ( $\geq 0$  and  $\leq 1$ ) representing a ratio of the graphical element's width that is the x end point of the gradient.
- $y1$  : Real [1] = 0 - a real number ( $\geq 0$  and  $\leq 1$ ) representing a ratio of the graphical element's height that is the y start point of the gradient.
- $y2$  : Real [1] = 1 - a real number ( $\geq 0$  and  $\leq 1$ ) representing a ratio of the graphical element's height that is the y end point of the gradient.

#### Constraints

- `valid_gradient_vector`: all the components of the gradient vector must be between 0 and 1.  
[OCL]  $x1 \geq 0$  and  $x1 \leq 1$  and  $x2 \geq 0$  and  $x2 \leq 1$  and  $y1 \geq 0$  and  $y1 \leq 1$  and  $y2 \geq 0$  and  $y2 \leq 1$

---

Issue: 18675 – MarkedElement should be abstract in provided CMOF files

---

### 10.3.17 MarkedElement [Abstract Class]

MarkedElement is a graphic element that can be decorated at its vertices with markers (e.g., arrowheads).

## Description

MarkedElement represents a graphical element that can optionally be decorated with markers (e.g., arrowheads) at its vertices (points of line intersection). It is an abstract super class of all graphical elements whose vertices are explicitly specified and ordered. A start marker decorates the first vertex, an end marker decorates the last vertex, and a mid marker decorates every other vertex in between. A marker has a higher z-order than its marked element.

## Diagrams

- Figure 10.3 (Primitive Elements)

## Generalizations

- GraphicalElement [Abstract Class]

## Specializations

- Polyline [Class]
- Path [Class]
- Polygon [Class]
- Line [Class]

## Association Ends

- startMarker : Marker [0..1] - an optional start marker that aligns with the first vertex of the marked element.
- endMarker : Marker [0..1] - an optional end marker that aligns with the last vertex of the marked element.
- midMarker : Marker [0..1] - an optional mid marker that aligns with all vertices of the marked element except the first and the last.

### 10.3.18 Marker [Class]

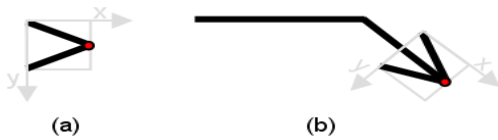
Marker is a kind of group that is used as a decoration (e.g., an arrowhead) for the vertices of a marked graphical element.

## Description

Marker is a kind of group that decorates a given vertex (a point of line intersection) of a marked element. A marker has its own private coordinate system whose origin is at point  $(x=0, y=0)$  and whose size is specified. The origin and the size define a clipping rectangle for the marker's member elements, which are laid out relative to the origin of the marker's coordinate system.

Additionally, a marker specifies a reference point within its bounds that is used to position the marker such that this point aligns exactly with the marked vertex. When the marker is positioned at a vertex, it also gets oriented (the axes of its coordinate system get rotated) to match the slope of the line at the vertex. For example, an arrow head marker can have a size of 10,10 (Figure 10.12 a) and a reference point of 10,5 (the small circle). When the marker is applied to the end vertex of a polyline (Figure 10.12 b), it gets aligned exactly with the end vertex and rotated to match the slope of the line at that vertex.

A marker does not render as a normal graphical element but is only used to decorate the vertices of its referencing marked elements. A marker is owned by the canvas and can be referenced by marked elements in three possible ways: as a start marker, an end marker, or a mid marker (see “Marker [Class]” on page 35 for more details). The styles of marked elements cascade to their referenced markers in each case. A marker can still define its own style overrides.



**Figure 10.12 - Marker example: a) an arrowhead marker with its reference point in red b) the marker positioned at the end vertex of a polyline and rotated to match the rotation at that vertex**

### Diagrams

- Figure 10.3 (Primitive Elements)
- Figure 10.4 (Group Elements)

### Generalizations

- Group [Class]

### Attributes

- size : Dimension [1] - the size of the marker
- reference : Point [1] - a point within the bounds of the marker that aligns exactly with the marked element’s vertex.

### Association Ends

- canvas : Canvas [1] - a reference to the canvas that owns this marker.

### 10.3.19 Matrix [DataType]

Matrix is a kind of transform that represents any transform operation with a 3x3 transformation matrix.

### Description

Matrix is a kind of transform that represents any transform operation with a 3x3 transformation matrix of the form shown in Figure 10.13. Since only six values are used in this 3x3 matrix, a matrix is also expressed as a vector [a b c d e f].

$$\begin{bmatrix} a & c & e \\ b & d & f \\ 0 & 0 & 1 \end{bmatrix}$$

**Figure 10.13 - Transform Matrix**

Transformation matrix map coordinates and lengths from a new coordinate system into a previous coordinate system, as shown in Figure 10.14.

$$\begin{bmatrix} X_{\text{prevCoordSys}} \\ Y_{\text{prevCoordSys}} \\ 1 \end{bmatrix} = \begin{bmatrix} a & c & e \\ b & d & f \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} X_{\text{newCoordSys}} \\ Y_{\text{newCoordSys}} \\ 1 \end{bmatrix}$$

**Figure 10.14 - Transform Matrix Multiplication**

### Diagrams

- Figure 10.7 (Transforms)

### Generalizations

- Transform [Abstract DataType]

### Attributes

- a : Real [1] - the a value of the transform matrix.
- b : Real [1] - the b value of the transform matrix.
- c : Real [1] - the c value of the transform matrix.
- d : Real [1] - the d value of the transform matrix.
- e : Real [1] - the e value of the transform matrix.
- f : Real [1] - the f value of the transform matrix.

### 10.3.20 MoveTo [DataType]

MoveTo is a kind of path command that establishes a new current point in the coordinate system.

### Description

MoveTo is a kind of path command that establishes a new current point. The effect is as if a pen was lifted and moved to a new location. A MoveTo command is always the first command in a path (in which case the new point is considered absolute regardless of the value of the isRelative flag). Subsequent MoveTo commands (i.e., when it is not the first command) represent the start of new subpaths (e.g., a doughnut shape consists of two subpaths, one for the outer circle and one for the inner circle).

### Diagrams

- Figure 10.8 (Path Commands)

### Generalizations

- PathCommand [Abstract DataType]

## Attributes

- point : Point [1] - a point to move to in the coordinate system.

### 10.3.21 Path [Class]

Path is a marked element that defines a custom shape whose geometry is specified with a sequence of path commands.

#### Description

Path is a marked element that renders as a custom shape whose geometry is specified with a sequence of path commands. A path command is an instruction to manipulate (move or press) a drawing pen on the canvas in a specific way. The sequence of pen instructions builds the outline of the custom shape and always starts with a MoveTo command to position the drawing pen at the start position. Multiple MoveTo commands may appear in the sequence effectively defining subpaths.

An example of a path element that draws a triangle is:

- MoveTo (50, 0) LineTo (0, 50)
- LineTo (100, 50)
- ClosePath

#### Diagrams

- Figure 10.3 (Primitive Elements)

#### Generalizations

---

**Issue: 18675 – MarkedElement should be abstract in provided CMOF files**

---

- MarkedElement [Abstract Class]

## Attributes

- command : PathCommand [\*] {ordered, nonunique} - a list of path commands that define the geometry of the custom shape.

### 10.3.22 PathCommand [Abstract DataType]

PathCommand is the abstract super type of all commands that participate in specifying a path element.

#### Description

PathCommand represents a command that participates in defining the geometry of a path element. It is the abstract super class of all path commands. The coordinates specified by a path command are either relative to the current point before the command or absolute (relative to the origin point of the coordinate system) based on the truth value of the isRelative flag.

#### Diagrams

- Figure 10.8 (Path Commands)



## Specializations

- ClosePath [DataType]
- EllipticalArcTo [DataType]
- MoveTo [DataType]
- CubicCurveTo [DataType]
- QuadraticCurveTo [DataType]
- LineTo [DataType]

## Attributes

- isRelative : Boolean [1] = false  
whether the coordinates specified by the command are relative to the current point (when true) or to the origin point of the coordinate system (when false).

## 10.3.23 Pattern [Class]

Pattern is a kind of fill that paints a graphical element (a tile) repeatedly at fixed intervals in x and y axes to cover the areas to be filled.

### Description

Pattern is a kind of fill that paints a graphical element (a tile) repeatedly inside a filled area at a fixed interval along the x and y axes. The interval is defined by the bounds of the pattern, which establishes its own private coordinate system for the pattern's tile to be relative to. The bounds of the pattern also define a rectangular clipping region for the tile restricting it from painting outside.

A pattern's tile does not render on its own as a normal graphical element but is only painted repeatedly within the pattern to fill enclosed areas of graphical elements. The styles of those graphical elements cascade to the pattern tiles in each case. A tile can still define its own style overrides.

### Diagrams

- Figure 10.6 (Fills)

### Generalizations

- Fill [Abstract Class]

### Attributes

- bounds : Bounds [1] - the bounds of the pattern that define a private coordinate system for the pattern's tile.

### Association Ends

- ♦ tile : GraphicalElement [1] - a reference to a graphical element, owned by the pattern, that works as a tile to be painted repeatedly at a fixed interval to fill a closed area.

### 10.3.24 Polygon [Class]

Polygon is a marked element that defines a closed shape consisting of a sequence of connected straight line segments.

#### Description

Polygon is a marked element that renders a closed shape consisting of a sequence of straight line segments defined by a list of three or more points in the x-y coordinate system. The sequence results in a closed shape as a last line is automatically defined from the last point to the first point.

#### Diagrams

- Figure 10.3 (Primitive Elements)

#### Generalizations

---

**Issue: 18675 – MarkedElement should be abstract in provided CMOF files**

---

- MarkedElement [Abstract Class]

#### Attributes

- point : Point [3..\*] {ordered, nonunique} - a list of 3 or more points making up the polygon.

### 10.3.25 Polyline [Class]

Polyline is a marked element that defines a shape consisting of a sequence of connected straight line segments.

#### Description

Polyline is a marked element that renders a shape consisting of a sequence of straight line segments defined by a list of two or more points in the x-y coordinate system.

#### Diagrams

- Figure 10.3 (Primitive Elements)

#### Generalizations

---

**Issue: 18675 – MarkedElement should be abstract in provided CMOF files**

---

- MarkedElement [Abstract Class]

#### Attributes

- point : Point [2..\*] {ordered, nonunique} - a list of 2 or more points making up the polyline.

### 10.3.26 QuadraticCurveTo [DataType]

QuadraticCurveTo is a kind of path command that draws a quadratic bézier curve from the current point to a new point using a single control point.

## Description

QuadraticCurveTo is a kind of path command that draws a quadratic bézier curve from the current point to a new point using a single control point. Multiple QuadraticCurveTo commands may be specified in a row to draw a polybézier. At the end of the command, the provided point becomes the new current point in the coordinate system. An example of a quadratic bézier curve is shown in Figure 10.15.

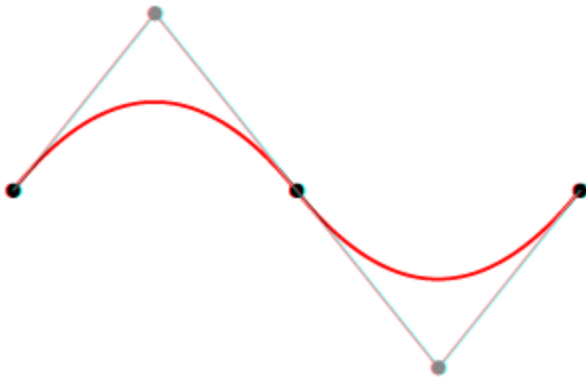


Figure 10.15 - Example of quadratic bézier curve

## Diagrams

- Figure 10.8 (Path Commands)

## Generalizations

- PathCommand [Abstract DataType]

## Attributes

- point : Point [1] - a point to draw a quadratic bézier curve to/from the current point in the coordinate system.
- control : Point [1] - the control point of the quadratic bézier curve.

## 10.3.27 RadialGradient [Class]

RadialGradient is a kind of gradient that fills a graphical element by smoothly changing color values in a circle.

## Description

RadialGradient is a kind of gradient that fills a graphical element by smoothly changing color values between gradient stops in a circle. The change occurs from a focus point in the circle (which does not have to be at the center) to its outside radius. The center point of the circle and its radius define the largest (outer most) circle for the gradient, while the focus point defines the smallest (inner most) circle.

The center point and focus point are expressed with a ratio, centerX, and focusX of the width of the graphical element and a ratio, centerY, and focusY of the height of the graphical element. The radius is expressed as a ratio of the size (width and height) of the graphical element.

## Diagrams

- Figure 10.6 (Fills)

## Generalizations

- Gradient [Abstract Class]

## Attributes

- centerX : Real [1] = 0.5 - a real number ( $\geq 0$  and  $\leq 1$ ) representing a ratio of the graphical element's width that is the x center point of the gradient.
- centerY : Real [1] = 0.5 - a real number ( $\geq 0$  and  $\leq 1$ ) representing a ratio of the graphical element's width that is the y center point of the gradient.
- radius : Real [1] = 0.5 - a real number ( $\geq 0$  and  $\leq 1$ ) representing a ratio of the graphical element's size that is the radius of the gradient.
- focusX : Real [1] = 0.5 - a real number ( $\geq 0$  and  $\leq 1$ ) representing a ratio of the graphical element's width that is the x focus point of the gradient.
- focusY : Real [1] = 0.5 - a real number ( $\geq 0$  and  $\leq 1$ ) representing a ratio of the graphical element's width that is the y focus point of the gradient.

## Constraints

- valid\_center\_point: the center point coordinates must be between 0 and 1  
[OCL] centerX $\geq 0$  and centerX $\leq 1$  and centerY $\geq 0$  and centerY $\leq 1$
- valid\_focus\_point: the focus point coordinates must be between 0 and 1  
[OCL] focusX $\geq 0$  and focusX $\leq 1$  and focusY $\geq 0$  and focusY $\leq 1$
- valid\_radius: the radius must be between 0 and 1  
[OCL] radius $\geq 0$  and radius $\leq 1$

### 10.3.28 Rectangle [Class]

Rectangle is a graphical element that defines a rectangular shape with given bounds. A rectangle may be given rounded corners by setting its corner radius.

## Description

Rectangle is a graphical element that renders as a rectangle shape with given bounds in the x-y coordinate system. A rectangle may have rounded corners by specifying a radius for its corners. A radius of 0 specifies a regular rectangle with sharp corners.

## Diagrams

- Figure 10.3 (Primitive Elements)

## Generalizations

- GraphicalElement [Abstract Class]

## Attributes

- bounds : Bounds [1] - the bounds of the rectangle in the x-y coordinate system.
- cornerRadius : Real [1] = 0 - a radius for the rectangle's rounded corners. When the radius is 0, the rectangle is drawn

with sharp corners.

### 10.3.29 Rotate [DataType]

Rotate is a kind of transform that rotates a graphical element by a given angle about a given center point in the x-y coordinate system.

#### Description

Rotate is a kind of transform that rotates a graphical element by a given angle (in degrees) about a given center point in the x-y coordinate system. The center point is optional and when not specified, it is considered to be the origin (0,0) point of the x-y coordinate system. Rotate is equivalent to the Matrix transform  $[\cos(\text{angle}) \sin(\text{angle}) -\sin(\text{angle}) \cos(\text{angle}) 0 0]$  around the origin (see “MoveTo [DataType]” on page 37).

#### Diagrams

- Figure 10.7 (Transforms)

#### Generalizations

- Transform [Abstract DataType]

#### Attributes

- angle : Real [1] - a real number representing the angle (in degrees) of rotation. Both positive (clock-wise) and negative (counter-clock-wise) values are allowed.
- center : Point [0..1] - a point in the x-y coordinate system about which the rotation is performed. If the point is not specified, it is assumed to be the origin of the x-y coordinate system.

### 10.3.30 Scale [DataType]

Scale is a kind of transform that scales (resizes) a graphical element by a given factor in the x-y coordinate system.

#### Description

Scale is a kind of transform that resizes a graphical element by a given scaling factor along the x and y axes. Scale is equivalent to the Matrix transform  $[\text{factorX} 0 0 \text{factorY} 0 0]$  (see “MoveTo [DataType]” on page 37).

#### Diagrams

- Figure 10.7 (Transforms)

#### Generalizations

- Transform [Abstract DataType]

#### Attributes

- factorX : Real [1] - a real number ( $\geq 0$ ) representing a scale factor along the x-axis.
- factorY : Real [1] - a real number ( $\geq 0$ ) representing a scale factor along the y-axis.

## Constraints

- non-negative-scale: scale factors cannot be negative.  
[OCL]  $\text{factorX} \geq 0$  and  $\text{factorY} \geq 0$

### 10.3.31 Skew [DataType]

Skew is a kind of transform that skews (deforms) a graphical element by given angles in the x-y coordinate system.

#### Description

Skew is a kind of transform that skews a graphical element by two angles (in degrees) along the x axis and the y axis. Skew is equivalent to the Matrix transform  $[1 \tan(\text{angleY}) \tan(\text{angleX}) 1 0 0]$  (see “MoveTo [DataType]” on page 37).

#### Diagrams

- Figure 10.7 (Transforms)

#### Generalizations

- Transform [Abstract DataType]

#### Attributes

- `angleX` : Real [1] - a real number representing the angle (in degrees) of skew along the x-axis. Both positive (clock-wise) and negative (counter-clock-wise) values are allowed.
- `angleY` : Real [1] - a real number representing the angle (in degrees) of skew along the y-axis. Both positive (clock-wise) and negative (counter-clock-wise) values are allowed.

### 10.3.32 Style [Class]

Style contains formatting properties that affect the appearance or style of graphical elements.

#### Description

Style represents a bag of properties (e.g., `fontName`, `fillColor`, or `strokeWidth`) that affect the appearance or style of graphical elements rather than their geometry. A style can either be local to (owned by) a graphical element or shared between (referenced by) several graphical elements (e.g., all UML dependency connectors have dashed lines), in which case it is packaged by the canvas. Shared styles help reduce the footprint of graphical models. A value set to a local style property overrides one that is set to the same property on a shared style referenced by the same graphical element.

Style properties are typically defined as optional to allow the state of “unset” to be legal. This is needed to implement cascading styles, where an unset style property in one graphical element gets its value from the closest graphical element in its group chain that has a value set for that property.

The above semantics effectively specify that a value for a style property is based on the following mechanisms (in order of precedence):

- If there is a cascading value set on a local style, use it. If the same value is set by multiple local styles, then use the value earlier in the style ordering.
- Otherwise, if there is cascading value set on a shared style, use it. If the same value is set by multiple shared styles, then use the value earlier in the style ordering.

- Otherwise, if a cascading value is available from a graphical element in the group chain, use it from the closest group.
- Otherwise, use the style property's default value.

## Diagrams

- Figure 10.2 (Graphical Element)
- Figure 10.4 (Group Elements)
- Figure 10.5 (Style)

## Attributes

- **fillColor** : Color [0..1] - a color that is used to paint the enclosed regions of graphical element. A fillColor value is exclusive with a fill value. If both are specified, the fill value is used. If none is specified, no fill is applied (i.e., the element becomes see-through).
- **fillOpacity** : Real [0..1] - a real number ( $\geq 0$  and  $\leq 1$ ) representing the opacity of the fill or fillColor used to paint a graphical element. A value of 0 means totally transparent, while a value of 1 means totally opaque. The default is 1.
- **strokeWidth** : Real [0..1] - a real number ( $\geq 0$ ) representing the width of the stroke used to paint the outline of a graphical element. A value of 0 specifies no stroke is painted. The default is 1.
- **strokeOpacity** : Real [0..1] - a real number ( $\geq 0$  and  $\leq 1$ ) representing the opacity of the stroke used for a graphical element. A value of 0 means totally transparent, while a value of 1 means totally opaque. The default is 1.
- **strokeColor** : Color [0..1] - the color of the stroke used to paint the outline of a graphical element. The default is black (red=0, green=0, blue=0).
- **strokeDashLength** : Real [\*] {ordered, nonunique} - a list of real numbers specifying a pattern of alternating dash and gap lengths used in stroking the outline of a graphical element with the first one specifying a dash length. The size of the list is expected to be even. If the list is empty, the stroke is drawn solid. The default is empty list.
- **fontSize** : Real [0..1] - a real number ( $\geq 0$ ) representing the size (in unit of length) of the font used to render a text element. The default is 10.
- **fontName** : String [0..1] - the name of the font used to render a text element (e.g., "Times New Roman," "Arial," or "Helvetica"). The default is "Arial."
- **fontColor** : Color [0..1] - the color of the font used to render a text element. The default is black (red=0, green=0, blue=0).
- **fontItalic** : Boolean [0..1] - whether the font used to render a text element has an italic style. The default is false.
- **fontBold** : Boolean [0..1] - whether the font used to render a text element has a bold style. The default is false.
- **fontUnderline** : Boolean [0..1] - whether the font used to render a text element has an underline style. The default is false.
- **fontStrikeThrough** : Boolean [0..1] - whether the font used to render a text element has a strike-through style. The default is false.

## Association Ends

- **fill** : Fill [0..1] - a reference to a fill that is used to paint the enclosed regions of a graphical element. A fill value is

exclusive with a fillColor value. If both are specified, the fill value is used. If none is specified, no fill is applied (i.e., the element becomes see-through).

### Constraints

- valid\_font\_size: the font size is non-negative.  
[OCL] `fontSize >= 0`
- valid\_fill\_opacity: the stroke width is non-negative.  
[OCL] `fillOpacity >= 0 and fillOpacity <=1`
- valid\_stroke\_width: the stroke width is non-negative.  
[OCL] `strokeWidth >= 0`
- valid\_dash\_length\_size: the size of the stroke dash length list must be even.  
[OCL] `strokeDashLength->size().mod(2) = 0`
- valid\_stroke\_opacity: the opacity of the fill is non-negative.  
[OCL] `strokeOpacity >= 0 and strokeOpacity <=1`

### 10.3.33 Text [Class]

Text is a graphical element that defines a shape that renders a character string within a bounding box.

#### Description

Text is a graphical element that renders a given sequence of characters within a bounding box. This means the text could be wrapped into multiple lines (at the edges of the box) and/or, if there is no extra room in the box, the remaining text is summarized with ellipses (...) at the end.

The text lines are rendered along the width (x-axis) of the bounding box according to the chosen alignment option, as follows:

- Alignment::start: the text lines' start edges are aligned with the start edge of the bounding box.
- Alignment::end: the text lines' end edges are aligned with the end edge of the bounding box.
- Alignment::center: the text lines' centers are aligned with the center of the bounding box.

#### Diagrams

- Figure 10.3 (Primitive Elements)

#### Generalizations

- GraphicalElement [Abstract Class]

#### Attributes

- data : String [1] - the text as a string of characters.
- bounds : Bounds [1] - the bounds inside which the text is rendered (possibly wrapped into multiple lines).
- alignment : AlignmentKind [1] - the text alignment when wrapped into multiple lines.



### 10.3.34 Transform [Abstract DataType]

Transform defines an operation that changes the geometry of a graphical element in a specific way.

#### Description

Transform is an operation that changes the geometry of a graphical element in a specific way. When a transform is applied to a non-group element, it changes the coordinates and lengths defined on that element. When it is applied to a group element, the transform is applied to each member of the group.

#### Diagrams

- Figure 10.7 (Transforms)

#### Specializations

- Matrix [DataType]
- Rotate [DataType]
- Skew [DataType]
- Translate [DataType]
- Scale [DataType]

### 10.3.35 Translate [DataType]

Translate is a kind of transform that translates (moves) a graphical element by a given delta along the x-y coordinate system.

#### Description

Translate is a kind of transform that moves a graphical element by a given delta (deltaX, deltaY) along the x-y axes. Translate is equivalent to the Matrix transform  $[1 \ 0 \ 0 \ 1 \ \text{deltaX} \ \text{deltaY}]$  (see “MoveTo [DataType]” on page 37).

#### Diagrams

- Figure 10.7 (Transforms)

#### Generalizations

- Transform [Abstract DataType]

#### Attributes

- `deltaX` : Real [1] - a real number representing a translate delta along the x-axis. Both positive and negative values are allowed.
- `deltaY` : Real [1] - a real number representing a translate delta along the y-axis. Both positive and negative values are allowed.

