Win2d Mini Language Specification

Version 0.1.1

RATISH PHILIP

06-FEB-2017

References

1. SVG Path Specification
2. WPF Path Markup Syntax
3. Win2d API Reference

REVISION HISTORY

1.	VERSION 0.1	31-JAN-2017
2.	VERSION 0.1.1	06-FEB-2017

Contents

1.1 Introduction

Microsoft.Graphics.Canvas.Geometry.CanvasGeometry class facilitates the drawing and manipulation of complex geometrical shapes. These shapes can be outlined with a stroke and filled with a brush (which can be a solid color, a bitmap pattern or a gradient).

While the **CanvasGeometry** class provides various static methods to create predefined shapes like Circle, Ellipse, Rectangle, RoundedRectangle, the **CanvasPathBuilder** class provides several methods to create freeform CanvasGeometry objects.

Creation of a complex freeform geometric shape may involve invoking of several CanvasPathBuilder commands. For example, the following code shows how to create a triangle geometry using CanvasPathBuilder

```
CanvasPathBuilder pathBuilder = new CanvasPathBuilder(device);

pathBuilder.BeginFigure(1, 1);
pathBuilder.AddLine(300, 300);
pathBuilder.AddLine(1, 300);
pathBuilder.EndFigure(CanvasFigureLoop.Closed);

CanvasGeometry triangleGeometry = CanvasGeometry.CreatePath(pathBuilder);
```

Win2d Mini Language is a powerful and sophisticated language which facilitates specifying complex geometries, color, brushes, strokes and stroke styles in a more compact manner.

Using Win2d Mini-Language, the geometry in above example can be created in the following way

```
string pathData = “M 1 1 300 300 1 300 Z”;
CanvasGeometry triangleGeometry = CanvasObject.CreateGeometry(device, pathData);
```

This document is based on the SVG Path language specification and describes in detail the syntax and features of the Win2d Mini Language and how it can be used to create complex geometries, Color, brushes, strokes and stroke styles. The helper classes mentioned in this document are a part of the **CompositionProToolkit** project available in GitHub and NuGet.

1.2 Win2d Mini Language

Win2d Mini Language syntax derives from the SVG path syntax. It is a prefix notation (i.e., commands followed by parameters). In order to ensure that the data, specified in Win2d mini language, is concise the following rules must be followed

- The commands are can be expressed as **one**, **two** or **three** characters (e.g., a **Move** command is expressed as an **M**, a LinearGradientBrush command is expressed as **LG**).
- Superfluous white space and separators such as commas can be eliminated (e.g., **M 100 100 L 200 200** contains unnecessary spaces and could be expressed more compactly as **M100 100L200 200**).
- The command letter can be eliminated on subsequent commands if the same command is used multiple times in a row (e.g., you can drop the second "L" in "M 100 200 L 200 100 L -100 -200" and use "M 100 200 L 200 100 -100 -200" instead).
- Relative versions of all commands are available (uppercase means absolute coordinates, lowercase means relative coordinates).
- Most of the parameters are either integer or floating point values. The only allowable decimal point is a Unicode U+0046 FULL STOP (".") character (also referred to in Unicode as PERIOD, dot and decimal point) and no other delimiter characters are allowed [UNICODE]. (For example, the following is an invalid numeric value in a path data stream: "13,000.56". Instead, say: "13000.56".)
- For the relative versions of the commands, all coordinate values are relative to the current point at the start of the command.

In the following sections, the following notation is used:

- **()** indicates grouping of parameters.
- **+** indicates one or more of the given parameter(s) is required.
- **[]** indicates any one of the values specified within the square brackets. For e.g. [01] means either 0 or 1.
- **?** denotes an optional parameter

1.2.1 Path Mini Language

Paths represent the outline of a shape, which can be filled, stroked, used as a clipping path, or any combination of the three.

A path is described using the concept of a current point. In an analogy with drawing on paper, the current point can be thought of as the location of the pen. The position of the pen can be changed, and the outline of a shape (open or closed) can be traced by dragging the pen in either straight lines or curves.

A path can be defined using the following commands

1.2.1.1 Fill Behavior

Command	Parameter
`F or f`	`[01]`

Fill Behavior specifies how the intersecting areas of geometries or figures are combined to form the area of the composite geometry. It represents the **CanvasFilledRegionDetermination** enumeration which is based on D2D1_FILL_MODE enumeration. The Fill Behavior parameter can be either **0** or **1**.

The parameter value **0** indicates **CanvasFilledRegionDetermination.Alternate** which determines whether a point is in the fill region by drawing a ray from that point to infinity in any direction, and then counting the number of path segments within the given shape that the ray crosses. If this number is odd, the point is in the fill region; if even, the point is outside the fill region.

The parameter value **1** indicates **CanvasFilledRegionDetermination.Winding** which determines whether a point is in the fill region of the path by drawing a ray from that point to infinity in any direction, and then examining the places where a segment of the shape crosses the ray. Starting with a count of zero, add one each time a segment crosses the ray from left to right and subtract one each time a path segment crosses the ray from right to left, as long as left and right are seen from the perspective of the ray. After counting the crossings, if the result is zero, then the point is outside the path. Otherwise, it is inside the path.

Each Path data should have only one Fill Behavior specified. Otherwise an exception will be raised.

This command invokes the **CanvasPathBuilder.SetFilledRegionDetermination()** API.

1.2.1.2 MoveTo

Command	Parameter
`M (absolute)` `M (relative)`	`(x y)+`

The **MoveTo** command establishes a new current point. The effect is as if the "pen" were lifted and moved to a new location. A path data segment must begin with a Move command. Subsequent "moveto" commands (i.e., when the Move is not the first command) represent the start of a new *subpath.*

M (uppercase) indicates that absolute coordinates will follow; **m** (lowercase) indicates that relative coordinates will follow. If a **MoveTo** command is followed by multiple pairs of coordinates, the subsequent pairs are treated as implicit **LineTo** commands. Hence, implicit LineTo commands will be relative if the MoveTo command is relative and absolute if the MoveTo command is absolute. If a relative MoveTo (m) command appears as the first element of the path, then its parameters are treated as a pair of absolute coordinates. In

this case, subsequent pairs of coordinates are treated as relative even though the initial MoveTo is interpreted as an absolute MoveTo.

This command invokes the **CanvasPathBuilder.BeginFigure()** API.

1.2.1.3 LineTo

Command	Parameter
`L (absolute)` `l (relative)`	`(x y)+`

Draws a line from the current point to the given (x,y) coordinate which becomes the new current point. **L** (uppercase) indicates that absolute coordinates will follow; **l** (lowercase) indicates that relative coordinates will follow. A number of coordinates pairs may be specified to draw a polyline. At the end of the command, the new current point is set to the final set of coordinates provided.

This command invokes the **CanvasPathBuilder.AddLine()** API.

1.2.1.4 Horizontal LineTo

Command	Parameter
`H (absolute)` `h (relative)`	`x+`

Draws a horizontal line from the current point (cpx, cpy) to (x, cpy). **H** (uppercase) indicates that absolute coordinates will follow; **h** (lowercase) indicates that relative coordinates will follow. Multiple **x** values can be provided (although usually this doesn't make sense). At the end of the command, the new current point becomes (**x, cpy**) for the final value of x.

This command invokes the **CanvasPathBuilder.AddLine()** API.

1.2.1.5 Vertical LineTo

Command	Parameter
`V (absolute)` `v (relative)`	`y+`

Draws a vertical line from the current point (cpx, cpy) to (cpx, y). **V** (uppercase) indicates that absolute coordinates will follow; **v** (lowercase) indicates that relative coordinates will follow. Multiple y values can be provided (although usually this doesn't make sense). At the end of the command, the new current point becomes (cpx, y) for the final value of y.

This command invokes the **CanvasPathBuilder.AddLine()** API.

1.2.1.6 Cubic Bézier

Command	Parameter
`C (absolute)` `c (relative)`	`(x1 y1 x2 y2 x y)+`

Draws a cubic Bézier curve from the current point to (x,y) using (x1,y1) as the control point at the beginning of the curve and (x2,y2) as the control point at the end of the curve. **C** (uppercase) indicates that absolute

coordinates will follow; **c** (lowercase) indicates that relative coordinates will follow. Multiple sets of coordinates may be specified to draw a polybézier. At the end of the command, the new current point becomes the final (x,y) coordinate pair used in the polybézier.

This command invokes the **CanvasPathBuilder.AddCubicBezier()** API.

1.2.1.7 Smooth Cubic Bézier

Command	Parameter
`S (absolute)` `s (relative)`	`(x2 y2 x y)+`

Draws a cubic Bézier curve from the current point to (x,y). The first control point is assumed to be the reflection of the second control point on the previous command relative to the current point. (If there is no previous command or if the previous command was not a **C, c, S** or **s**, assume the first control point is coincident with the current point.) (x2,y2) is the second control point (i.e., the control point at the end of the curve). **S** (uppercase) indicates that absolute coordinates will follow; **s** (lowercase) indicates that relative coordinates will follow. Multiple sets of coordinates may be specified to draw a polybézier. At the end of the command, the new current point becomes the final (x,y) coordinate pair used in the polybézier.

This command invokes the **CanvasPathBuilder.AddCubicBezier()** API.

1.2.1.8 Quadratic Bézier

Command	Parameter
`Q (absolute)` `q (relative)`	`(x1 y1 x y)+`

Draws a quadratic Bézier curve from the current point to (x,y) using (x1,y1) as the control point. **Q** (uppercase) indicates that absolute coordinates will follow; **q** (lowercase) indicates that relative coordinates will follow. Multiple sets of coordinates may be specified to draw a polybézier. At the end of the command, the new current point becomes the final (x,y) coordinate pair used in the polybézier.

This command invokes the **CanvasPathBuilder.AddQuadraticBezier()** API.

1.2.1.9 Smooth Quadratic Bézier

Command	Parameter
`T (absolute)` `t (relative)`	`(x y)+`

Draws a quadratic Bézier curve from the current point to (x,y). The control point is assumed to be the reflection of the control point on the previous command relative to the current point. (If there is no previous command or if the previous command was not a **Q, q, T** or **t**, assume the control point is coincident with the current point.) **T** (uppercase) indicates that absolute coordinates will follow; **t** (lowercase) indicates that relative coordinates will follow. At the end of the command, the new current point becomes the final (x,y) coordinate pair used in the polybézier.

This command invokes the **CanvasPathBuilder.AddQuadraticBezier()** API.

1.2.1.10 Arc

Command	Parameter
A (absolute) a (relative)	(radius radiusY angle IsLargeFlag SweepDirection x y)+ where IsLargeFlag – [01] SweepDirection – [01]

Draws an elliptical arc from the current point to (**x**, **y**). The size and orientation of the ellipse are defined by two radii (**rx**, **ry**) and an **x-axis-rotation**, which indicates how the ellipse as a whole is rotated relative to the current coordinate system. The center (**cx**, **cy**) of the ellipse is calculated automatically to satisfy the constraints imposed by the other parameters. **SweepDirection** specifies the direction that an elliptical arc is drawn (0 – CounterClockwise and 1 – Clockwise). **IsLargeFlag** specifies whether an arc should take the longer or shorter way, around the ellipse to join its start and end points (0 denotes the arc's sweep should be π or less whereas 1 denotes the arc's sweep should be π or more).

This command invokes the **CanvasPathBuilder.AddArc()** API.

1.2.1.12 Close Path

Command	Parameter
Z or z	None

Closes the current subpath by drawing a straight line from the current point to current subpath's initial point. Since the Z and z commands take no parameters, they have an identical effect.

This command invokes the **CanvasPathBuilder.EndFigure()** API.

1.2.1.13 Ellipse Figure

Command	Parameter
O (absolute) o (relative)	(radiusX radiusY x y)+

Adds an Ellipse Figure to the path. The **radiusX** and **radiusY** parameters denote the elliptical radii on the x-axis and y-axis respectively. (**x y**) denotes the center of the Ellipse. The current point remains unchanged.

This command internally invokes the **CanvasPathBuilder.AddEllipseFigure()** extension method.

1.2.1.14 Polygon Figure

Command	Parameter
P (absolute) p (relative)	(numSides radius x y)+

Adds an n-sided Polygon to the path. The **radius** parameter denotes the radius of the circle circumscribing the polygon vertices. (**x y**) denotes the center of the polygon. The current point remains unchanged.

This command internally invokes the **CanvasPathBuilder.AddPolygonFigure()** extension method.

1.2.1.15 Rectangle Figure

Command	Parameter
`R (absolute)` `r (relative)`	`(x y width height)+`

Adds a Rectangle to the path. (**x y**) denotes the top left corner of the Rectangle. The current point remains unchanged.

This command internally invokes the **CanvasPathBuilder.AddRectangleFigure()** extension method.

1.2.1.16 RoundedRectangle Figure

Command	Parameter
`U (absolute)` `u (relative)`	`(x y width height radiusX radiusY)+`

Adds a RoundedRectangle to the path. (**x y**) denotes the top left corner of the RoundedRectangle. **radiusX** and **radiusY** denote the radii of the corner curves on the x-axis and y-axis respectively. The current point remains unchanged.

This command internally invokes the **CanvasPathBuilder.AddRoundedRectangleFigure()** extension method.

1.2.2 CanvasBrush Mini Language

This section describes in detail how a Win2d brush can be defined as a string and an instance created from it. Using the CanvasBrush Mini Language the following Win2d brushes can be created (they all implement the **ICanvasBrush** interface)

- SolidColorBrush
- CanvasLinearGradientBrush
- CanvasRadialGradientBrush

The Mini Language uses the following syntax to define a brush and its attributes

```
<BrushTypeCommand> (<BrushAttributeCommand> <BrushAttributeParameters>)+
```

Some of the attributes are optional while the others are mandatory. The order in which the attributes are specified should be maintained.

1.2.2.1 ICanvasBrush Attributes

This section describes the various brush attributes that will be used to define and construct an ICanvasBrush.

1.2.2.1.1 Start Point

Command	Parameter
M	(x y)

Denotes the point on the canvas where the gradient starts.

1.2.2.1.2 End Point

Command	Parameter
Z	(x y)

Denotes the point on the canvas where the gradient ends.

1.2.2.1.3 Opacity

Command	Parameter
O	opacity

Denotes the opacity of the brush. The **opacity** parameter should have a value in the range [0, 1].

1.2.2.1.4 Alpha Mode

Command	Parameter
A	**[012]**

Specifies the way in which an alpha channel affects color channels. This attribute corresponds to the CanvasAlphaMode enumeration. Default is **0** (**Premultiplied**).

Value	Member	Description
0	Premultiplied	The alpha value has been premultiplied. In blending, each color is scaled by the alpha value. Note that the alpha value itself is the same in both straight and premultiplied alpha. Typically, no color channel value is greater than the alpha channel value.
1	Straight	The alpha channel indicates the transparency of the color.
2	Ignore	The alpha value is ignored.

1.2.2.1.5 Buffer Precision

Command	Parameter
B	**[01234]**

Specifies the bit depth used for graphical computations. This attribute corresponds to the CanvasBufferPrecision enumeration. Default is **0** (**Precision8UIntNormalized**).

Value	Member	Description
Precision8UIntNormalized	0	Use 8-bit normalized integer per channel.
Precision8UIntNormalizedSrgb	1	Use 8-bit normalized integer standard RGB data per channel.
Precision16UIntNormalized	2	Use 16-bit normalized integer per channel.
Precision16Float	3	Use 16-bit floats per channel.
Precision32Float	4	Use 32-bit floats per channel.

1.2.2.1.6 Edge Behavior

Command	Parameter
E	**[012]**

Specifies the behavior for pixels which fall outside of the gradient's typical rendering area. This attribute corresponds to the CanvasEdgeBehavior enumeration. Default is **0** (**Clamp**).

Value	Member	Description
0	Clamp	Repeat the edge pixels of the brush's content.
1	Wrap	Tile the brush's content.
2	Mirror	Tile the the brush's content, and flip each alternate tile.

1.2.2.1.7 Pre Interpolation Color Space

Command	Parameter
P	**[012]**

Specifies the color space to be used before interpolation. This attribute corresponds to the CanvasColorSpace enumeration. Default value is **1** (**Srgb**).

Value	Member	Description
0	Custom	The color space is a custom ICC ColorManagementProfile.
1	Srgb	The color space is sRGB.
2	ScRgb	The color space is scRGB.

1.2.2.1.8 Post Interpolation Color Space

Command	Parameter
R	**[012]**

Specifies the color space to be used after interpolation. This attribute corresponds to the CanvasColorSpace enumeration. Default value is **1** (**Srgb**).

Value	Member	Description
0	Custom	The color space is a custom ICC ColorManagementProfile.
1	Srgb	The color space is sRGB.
2	ScRgb	The color space is scRGB.

1.2.2.1.9 Origin Offset

Command	Parameter
F	**(x y)**

Specifies a displacement from Center, used to form the brush's radial gradient.

1.2.2.1.10 GradientStop

Command	Parameter
S	**(pos hexColor)+**

Specifies one or more gradient stops (using Hexadecimal Color) in a gradient brush. The parameter **pos** refers to the position of the gradient stop and should be a floating point number between 0 and 1, inclusive. The parameter **hexColor** denotes the color specified in **#RRGGBB** or **#AARRGGBB** format.

Example:

```
“S 0.2 FF112233”
“S 0.1 #FF0000 0.4 #00FF00 0.9 #0000FF”
```

1.2.2.1.11 GradientStopHdr

Command	Parameter
S	**(pos x y z w)+**

Specifies one or more high dynamic range gradient stops in a gradient brush. The parameter **pos** refers to the position of the gradient stop and it should be a floating point number between 0 and 1, inclusive. The parameters **x, y, z** and **w** denote the components of the high dynamic range color.

Example:

```
"S 0.2 0.2 .5 .75 1"
"S 0.1 0.3 0.4 0.5 1.0 0.4 .3 .2 .1 0.9"
```

1.2.2.2 SolidColorBrush

Command	Parameter
SC	**hexColor (O opacity)?** **or** **x y z w (O opacity)?**

The SolidColorBrush is defined by the **SC** command. It has two attributes - **Color** and **Opacity**. The Color attribute is specified as either a Hexadecimal Color or a High Dynamic Range Color. The Opacity attribute is optional and can be specified with the **O** attribute command and a opacity value in the range [0, 1].

Example:

```
"SC #FFAABBCC"
"SC #FF1233AA O .75"
"SC 0.1 0.3 0.4 0.5"
"SC 0.1 0.3 0.4 0.5 O 0.9"
```

1.2.2.3 LinearGradientBrush

Command	Parameter
LG	**M x0 y0 Z x1 y1 (O opacity A [012] B [01234] E [012] P [012] R [012])? S (pos hexColor)+**

The LinearGradientBrush is defined by the **LG** command. It has the following mandatory attributes

- StartPoint (**M**)
- EndPoint (**Z**)
- GradientStops (**S**)

It also has the following optional attributes

- Opacity (**O**)
- Alpha Mode (**A**)
- Buffer Precision (**B**)
- Edge Behavior (**E**)

- Pre Interpolation Color Space (**P**)
- Post Interpolation Color Space (**R**)

Example:

```
“LG M 0 80 Z0 0 S 0.00 #ffee5124, 0.18 #fff05627, 0.26 #fff15b29, 0.72
#fff58535, 1.00 #fff9af41”
“LG M 0 80 Z0 0 O 0.75 A 1 E 2 S 0.00 #ffee5124, 0.3 #fff05627, 0.75
#fff58535, 1.00 #fff9af41”
```

1.2.2.4 LinearGradientBrush with GradientStopHdr

Command	Parameter
LH	**M x0 y0 Z x1 y1 (O opacity A [012] B [01234] E [012] P [012] R [012])? S (pos x y z w)+**

The LinearGradientBrush with GradientStopHdr is defined by the **LH** command. It has the following mandatory attributes

- StartPoint (**M**)
- EndPoint (**Z**)
- GradientStopHdrs (**S**)

It also has the following optional attributes

- Opacity (**O**)
- Alpha Mode (**A**)
- Buffer Precision (**B**)
- Edge Behavior (**E**)
- Pre Interpolation Color Space (**P**)
- Post Interpolation Color Space (**R**)

Example:

```
“LH M 0 80 Z0 0 P1 R1 S 0.00 0.9333333, 0.3176471, 0.1411765, 1, 0.18
0.9411765, 0.3372549, 0.1529412, 1, 0.26 0.945098, 0.3568628, 0.1607843, 1,
0.72 0.9607843, 0.5215687, 0.2078431, 1, 1.00 0.9764706, 0.6862745,
0.254902, 1”
```

1.2.2.5 RadialGradientBrush

Command	Parameter
RG	**radX radY x y (O opacity A [012] B [01234] E [012] F x1 y1 P [012] R [012])? S (pos hexColor)+**

The RadialGradientBrush is defined by the **RG** command. It has the following mandatory attributes

- RadiusX
- RadiusY

- CenterX (**x**)
- CenterY (**y**)
- GradientStops (**S**)

It also has the following optional attributes

- Opacity (**O**)
- Alpha Mode (**A**)
- Buffer Precision (**B**)
- Edge Behavior (**E**)
- Origin Offset (**F**)
- Pre Interpolation Color Space (**P**)
- Post Interpolation Color Space (**R**)

Example:

```
"RG 40 60 320 400 S 0.00 #ffee5124, 0.18 #fff05627, 0.26 #fff15b29, 0.72
#fff58535, 1.00 #fff9af41"
"RG 40 60 320 400 A 1 B2 E 2  S 0.00 #ffee5124, 0.3 #fff05627, 0.75
#fff58535, 1.00 #fff9af41"
```

1.2.2.6 RadialGradientBrush with GradientStopHdr

Command	Parameter
RH	**radX radY x y (O opacity A [012] B [01234] E [012] F x1 y1 P [012] R [012])? S (pos x y z w)+**

The RadialGradientBrush is defined by the **RG** command. It has the following mandatory attributes

- RadiusX
- RadiusY
- CenterX (**x**)
- CenterY (**y**)
- GradientStopHdrs (**S**)

It also has the following optional attributes

- Opacity (**O**)
- Alpha Mode (**A**)
- Buffer Precision (**B**)
- Edge Behavior (**E**)
- Origin Offset (**F**)
- Pre Interpolation Color Space (**P**)
- Post Interpolation Color Space (**R**)

Example:

```
"RH 400 400 320 320 O 0.94 A 1 E2 S 0.00 0.9333333, 0.3176471, 0.1411765,
1, 0.18 0.9411765, 0.3372549, 0.1529412, 1, 0.26 0.945098, 0.3568628,
0.1607843, 1, 0.72 0.9607843, 0.5215687, 0.2078431, 1, 1.00 0.9764706,
0.6862745, 0.254902, 1"
```

1.2.3 CanvasStrokeStyle Mini Language

The **Microsoft.Graphics.Canvas.Geometry.CanvasStrokeStyle** class defines a stroke style for drawing lines. The stroke style describes whether the line comprises of dashes, dots or solid line, how to join line segments, how to cap the ends etc.

This section describes in detail how a CanvasStrokeStyle can be defined as a string and a CanvasStrokeStyle instance created from it.

The Mini Language uses the following syntax to define a CanvasStrokeStyle and its attributes

```
CSS (<StrokeStyleAttributeCommand> <StrokeStyleAttributeParameter>)+
```

Some of the attributes are optional while the others are mandatory. The order in which the attributes are specified should be maintained.

1.2.3.1 CanvasStrokeStyle Attributes

This section describes the various brush attributes that are used to define and construct a CanvasStrokeStyle object.

1.2.3.1.1 Dash Style

Command	Parameter
DS	**[01234]**

Describes the sequence of dashes, dots, and gaps in a stroke style. This attribute is ignored if the **CustomDashStyle** attribute is set. This attribute corresponds to the CanvasDashStyle enumeration. Default value is **0** (**Solid**).

Value	Member	Description
0	Solid	A solid line with no breaks.
1	Dash	A dash followed by a gap of equal length. The dash and the gap are each twice as long as the stroke thickness. The equivalent custom dash style is {2, 2}.
2	Dot	A dot followed by a longer gap. The equivalent custom dash style is {0, 2}.
3	DashDot	A dash, followed by a gap, followed by a dot, followed by another gap. The equivalent custom dash style is {2, 2, 0, 2}.
4	DashDotDot	A dash, followed by a gap, followed by a dot, followed by another gap, followed by another dot, followed by another gap. The equivalent custom dash style is {2, 2, 0, 2, 0, 2}.

1.2.3.1.2 Line Join

Command	Parameter
LJ	[0123]

Describes the shape that joins two lines or segments. This attribute corresponds to the CanvasLineJoin enumeration. Default value is **0** (**Miter**).

Value	Member	Description
0	Miter	Regular angular vertices.
1	Bevel	Beveled vertices.
2	Round	Rounded vertices.
3	MiterOrBevel	Regular angular vertices unless the join would extend beyond the miter limit; otherwise, beveled vertices.

1.2.3.1.3 Miter Limit

Command	Parameter
ML	limit

Describes the limit on the ratio of the miter length to half the stroke's thickness.

1.2.3.1.4 Dash Offset

Command	Parameter
DO	offset

Describes how far into the dash sequence the stroke will start.

1.2.3.1.5 Start Cap

Command	Parameter
SC	[0123]

Describes the type of shape used at the beginning of a stroke. This attribute corresponds to the CanvasCapStyle enumeration. Default value is **0** (**Flat**).

Value	Member	Description
0	Flat	A cap that does not extend past the last point of the line.
1	Square	Half of a square that has a length equal to the line thickness.
2	Round	A semicircle that has a diameter equal to the line thickness.
3	Triangle	An isosceles right triangle whose hypotenuse is equal in length to the thickness of the line.

1.2.3.1.6 End Cap

Command	Parameter
EC	**[0123]**

Describes the type of shape used at the beginning of a stroke. This attribute corresponds to the CanvasCapStyle enumeration. Default value is **0** (**Flat**).

Value	Member	Description
0	Flat	A cap that does not extend past the last point of the line.
1	Square	Half of a square that has a length equal to the line thickness.
2	Round	A semicircle that has a diameter equal to the line thickness.
3	Triangle	An isosceles right triangle whose hypotenuse is equal in length to the thickness of the line.

1.2.3.1.7 Dash Cap

Command	Parameter
DC	**[0123]**

Describes the type of shape used at the beginning of a stroke. This attribute corresponds to the CanvasCapStyle enumeration. Default value is **0** (**Flat**).

Value	Member	Description
0	Flat	A cap that does not extend past the last point of the line. If **Dash Cap** is set to Flat, dots will have zero size so only dashes are visible.
1	Square	Half of a square that has a length equal to the line thickness.
2	Round	A semicircle that has a diameter equal to the line thickness.
3	Triangle	An isosceles right triangle whose hypotenuse is equal in length to the thickness of the line.

1.2.3.1.8 Transform Behavior

Command	Parameter
TB	**[012]**

Describes how the world transform, dots per inch (DPI), and stroke width affect the shape of the pen. This attribute corresponds to the CanvasStrokeTransformBehavior enumeration. Default value is **0** (**Normal**).

Value	Member	Description
0	Normal	The stroke respects the currently set world transform, the DPI, and the stroke width.
1	Fixed	The stroke does not respect the world transform but it does respect the DPI and stroke width.
2	Hairline	The stroke is forced to 1 pixel wide (in device space) and does not respect the world transform, the DPI, or the stroke width.

1.2.3.1.1 Custom Dash Style

Command	Parameter
CDS	**(dashSize spaceSize)+**

Describes an array describing a custom dash pattern. The array elements specify the length of each dash and space in the pattern. The first element sets the length of a dash, the second element sets the length of a space, and the third element sets the length of a dash, and so on. The length of each dash and space in the dash pattern is the product of the element value in the array and the stroke width. This overrides the DashStyle property, which is only used when **CustomDashStyle** is set to null. This array must contain an even number of elements.

Example:

```
"CDS 2 2 0 2 1 3"
```

1.2.3.2 Defining the CanvasStrokeStyle

Using the attributes defined in the previous section the CanvasStrokeStyle can be defined in the following way

Command	Parameter
CSS	**(DS [01234] LJ [0123] ML limit DO offset SC [0123] EC [0123] DC [0123] TB [012] CDS (dashSize spaceSize)+)?**

All the attributes of CanvasStrokeStyle are optional. If none of the attributes is specified in the CanvasStrokeStyle definition string, then the default CanvasStrokeStyle object is created.

Note: While specifiying the attributes, the order of the attributes must be maintained.

Example:

```
"CSS CDS 2 2 0 2 1 3"
"CSS"
"CSS DS2 DO2 SC 1 EC 2"
```

1.2.4 CanvasStroke Mini Language

1.2.4.1 ICanvasStroke interface and CanvasStroke class

In Win2d, the stroke, that is used to render an outline to a CanvasGeometry, is comprised of three components

- Stroke Width – defines the width of the stroke.
- Stroke Brush – defines the **ICanvasBrush** that will be used to render the stroke.
- Stroke Style – defines the **CanvasStrokeStyle** for the stroke.

ICanvasStroke interface, defined in the **CompositionProToolkit.Win2d** namespace, encapsulates these three components and the **CanvasStroke** class implements this interface.

```
public interface ICanvasStroke
{
    ICanvasBrush Brush { get; }
    float Width { get; }
    CanvasStrokeStyle Style { get; }
    Matrix3x2 Transform { get; set; }
}

public sealed class CanvasStroke : ICanvasStroke
{
    public ICanvasBrush Brush { get; }
    public float Width { get; }
    public CanvasStrokeStyle Style { get; }
    public Matrix3x2 Transform { get; set; }

    public CanvasStroke(ICanvasBrush brush, float strokeWidth = 1f);
    public CanvasStroke(ICanvasBrush brush, float strokeWidth,
                        CanvasStrokeStyle strokeStyle);
    public CanvasStroke(ICanvasResourceCreator device, Color strokeColor,
                        float strokeWidth = 1f);
    public CanvasStroke(ICanvasResourceCreator device, Color strokeColor,
                        float strokeWidth, CanvasStrokeStyle strokeStyle);
}
```

The **Transform** property in CanvasStroke gets or sets the Transform property of the stroke brush.
In order to define a CanvasStroke using the Mini Language, the **ST** command is used in the following way

```
ST strokeWidth <CanvasBrushML> <CanvasStrokeStyleML>
```

In this command string, **strokeWidth** is a positive floating point number, **CanvasBrushML** denotes the CanvasBrush Mini Language and the **CanvasStrokeStyleML** denotes the CanvasStrokeStyle Mini Language.

Example:

```
“ST 4.5 LG M 0 0 Z80 80 S 0.00 #ffff0000, 0.5 #ff00ff00, 0.99 #ff0000ff CSS
DS2 Do2 SC 1 EC 2 CDS 2 2 0 2 1 3”
“ST 2 SC #ff0000”
```

1.3 Parsing Win2d Mini Language

The **CompositionProToolkit.CanvasObject** static class provides APIs that parse the Win2d Mini Language and instantiate the appropriate objects.

This section describes about the various CanvasObject APIs.

1.3.1 Color

1.3.1.1 From Hexadecimal Color String or High Dynamic Range Color String

There are two APIs that convert the hexadecimal color string in **#RRGGBB** or **#AARRGGBB** format or the High Dynamic Range Color string in the **R G B A** format to the corresponding Color object. *The '#' character is optional in Hexadecimal color string. **R, G, B & A** should have value in the range between **0** and **1**, inclusive.*

```
public static Color CreateColor(string colorString);
public static bool TryCreateColor(string  colorString , out Color color);
```

The first API will raise an **ArgumentException** if the argument is not in the correct format while the second API will attempt to convert the color string without raising an exception.

1.3.1.2 From High Dynamic Range Color (Vector4)

The following API Converts a **Vector4** High Dynamic Range Color to Color object. Negative components of the Vector4 will be sanitized by taking the absolute value of the component. The HDR Color components should have value in the range [0, 1]. If their value is more than 1, they will be clamped at 1. **Vector4**'s X, Y, Z, W components match to **Color**'s R, G, B, A components respectively.

```
public static Color CreateColor(Vector4 hdrColor);
```

1.3.2 CanvasGeometry

The following API converts a CanvasGeometry path string to **CanvasGeometry** object

```
public static CanvasGeometry CreateGeometry(ICanvasResourceCreator resourceCreator,
                                            string pathData,
                                            StringBuilder logger = null);
```

The **logger** parameter in this method is an option argument of type **StringBuilder** that can be used to obtain the **CanvasPathBuilder** commands in text format. It is mainly intended for information/debugging purpose only.

1.3.3 ICanvasBrush

The following API converts a brush data string to **ICanvasBrush** object

```
public static ICanvasBrush CreateBrush(ICanvasResourceCreator resourceCreator,
                                       string brushData);
```

1.3.4 CanvasStrokeStyle

The following API converts a style data string to **CanvasStrokeStyle** object

```
public static CanvasStrokeStyle CreateStrokeStyle(string styleData);
```

1.3.5 ICanvasStroke

The following API converts a stroke data string to **ICanvasStroke** object

```
public static ICanvasStroke CreateStroke(ICanvasResourceCreator resourceCreator,
                                         string strokeData);
```