**S-100 – Part 9a**

**Portrayal (Lua)**

Page intentionally left blank

**Contents**

9a-1 Scope 1

9a-2 Conformance 1

9a-3 Normative references 1

9a-4 Portrayal Catalogue 2

9a-5 General Portrayal Model 2

9a-5.1 The Portrayal Process 2

9a-5.2 Lua Portrayal Process 3

9a-5.2.1 Portrayal Initialization 4

9a-5.2.2 Generating a Portrayal 4

9a-6 Package Overview 5

9a-7 Data input schema 5

9a-8 Information objects 5

9a-9 Feature objects 5

9a-10 Portrayal processing 5

9a-11 Drawing Instructions 6

9a-11.1 The concepts of drawing instructions 6

9a-11.1.1 General concept 6

9a-11.2 Model of the Drawing Instructions 6

9a-11.2.1 Drawing Commands 6

9a-11.2.2 State Commands 9

9a-12 Symbol Definitions 22

9a-13 The Portrayal Library 22

9a-14 Portrayal Domain Specific Functions 23

9a-14.1 Portrayal Domain Specific Catalogue Functions 23

9a-14.1.1 Boolean PortrayalMain(String[] featureIDs) 23

9a-14.1.2 void PortrayalInitializeContextParameters(ContextParameter[] c*ontextParameters*) 23

9a-14.1.3 ContextParameter PortrayalCreateContextParameter(String *contextParameterName*, String *contextParameterType*, String *defaultValue*) 23

9a-14.1.4 void PortrayalSetContextParameter(String *contextParameterName*, String *value*) 24

9a-14.2 Portrayal Domain Specific Host Functions 24

9a-14.2.1 Boolean HostPortrayalEmit(String *featureID*, String *drawingInstructions*, String *observedParameters*) 24

Page intentionally left blank

# Scope

This part defines the changes to S-100 Part 9 necessary to implement portrayal using the scripting mechanism defined in S-100 Part 13. Products which specify use of a portrayal catalogue as described in this part must also require implementation of S-100 Part 13.

# Conformance

This part of the specification conforms to S-100 Part 13.

# Normative references

The following referenced documents are required for the application of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including amendments) applies.

*Lua 5.1 Reference Manual*, <https://www.lua.org/manual/5.1/>

# Portrayal Catalogue

There are no changes to the Part 9 portrayal catalogue overview.

# General Portrayal Model

There are no changes to the Part 9 general portrayal model. A Lua portrayal follows the general portrayal model described in 9-5. Figure 9a-1 illustrates the general portrayal model.

Figure 9a- – General portrayal model

Portrayal Engine

Rendering Engine

Feature Data

Portrayal functions

Drawing Instructions

Portrayal Output

Symbol Definitions

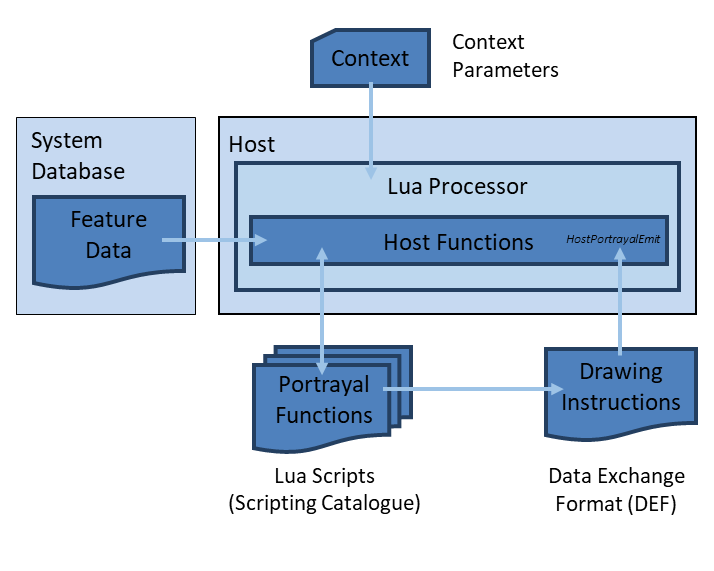
## The Portrayal Process

As illustrated in Figure 9a-2, a Lua portrayal requires the following changes to the portrayal process described in Part 9, clause 9-5.1 and captured in Table 9a-1:

Table 9a- - Changes to the portrayal process

|  |  |
| --- | --- |
| **Part 9** | **Part 9a** |
| Portrayal functions are written in the XSLT programming language. | Portrayal functions are written in the Lua programming language. |
| Host provides an XSLT implementation. | Host provides a Lua interpreter or Lua virtual machine. |
| Feature data is exposed to the portrayal functions via an XML document which must describe all features to be portrayed, along with all attribution, spatial relations, information associations, and all other information which may be used by the portrayal functions. | Feature data is not initially exposed to the portrayal functions. Instead, the host provides a list of the feature IDs to be portrayed; the portrayal functions will request attribution, spatial relations, information associations, and all other information as needed via host call-back functions. |
| Drawing instructions are returned to the host as an XML document, which is the result of an XSL transformation applied to the input feature data. | Drawing instructions are returned to the host via host call-back function *HostPortrayalEmit*. |

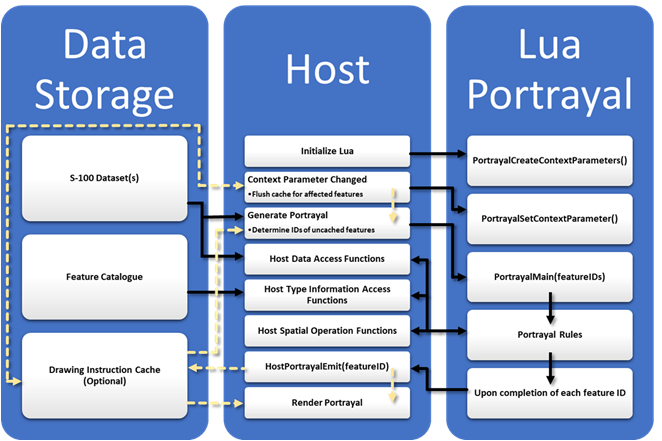
Figure 9a-2 – Portrayal process



## **Lua** Portrayal Process

This section describes the Part 9a portrayal process in detail, and indicates where there are changes to Part 9. The Lua portrayal process is shown in Figure 9a-3.

Figure 9a-3 - Lua Portrayal Process



### Portrayal Initialization

Prior to calling Lua portrayal functions, the host must register the domain specific scripting catalogue functions by loading a portrayal catalogue *TopLevelTemplate* rule file (a Lua script file). In order to prevent name collisions on *PortrayalMain*, the host must instantiate and initialize a new Lua runtime environment each time the *TopLevelTemplate* is changed. Alternatively, the host can maintain multiple Lua runtimes, one for each *TopLevelTemplate*.

After registering the scripting catalogue functions, the host calls

*PortrayalInitializeContextParameters*, passing in the name and default value for each portrayal context parameter defined by the portrayal catalogue. The portrayal context parameter values are associated with the given dataset and stay in effect until the scripting session is closed, or the values are changed via *PortrayalSetContextParameter*.

### Generating a Portrayal

Portrayal script function *PortrayalMain* (see clause 9a-14.1.1) is used to generate drawing instructions for a set of feature instances. The host passes in a set of feature IDs to *PortrayalMain*;the portrayal scripts will iterate over the feature IDs and generate drawing instructions for each.

As each feature instance is processed, the portrayal engine will call standard host functions to request attribute, spatial, or other information as needed. Upon completion of processing for a feature instance the portrayal engine will call *HostPortrayalEmit* (see clause 9a-14.2.1) and provide the drawing instructions for that feature instance to the host application.

The portrayal for a given *S100\_Dataset* is complete when the call to *PortrayalMain* returns. If the portrayal completed successfully, *PortrayalMain* returns true, otherwise *PortrayalMain* returns false along with a message indicating why the portrayal did not run to completion.

A host can terminate a portrayal prior to processing all feature instances by returning false from *HostPortrayalEmit*.

Calling *PortrayalMain* with all feature IDs from a given dataset will generate drawing instructions for the entire dataset. Drawing instructions for a subset of a dataset can be (re)generated by passing in feature IDs corresponding to the subset. This is useful when the host needs to regenerate a set of cached drawing instructions, or if the host is portraying a subset of a dataset such as a single *S100\_DataCoverage*.

#### Implementing a Portrayal Cache

In order to speed up the rendering process the host can optionally implement a portrayal cache. A portrayal cache is used to cache the drawing instructions which are output from the portrayal. Caching the drawing instructions for each feature instance allows the host to re-render feature instances without re-generating their portrayal. A cached drawing instruction only needs to be re-generated when one or more context parameters which were used to generate the drawing instruction changes.

When the portrayal scripts return the drawing instructions for a feature instance they also return a list of “observed” portrayal context parameters (see clause 9a-14.2.1). The observed context parameters are those context parameters which were evaluated during the generation of drawing instructions for a particular feature. For more detail on context parameters refer to Part 9 clause 9-13.3.22.

A notional portrayal cache is shown in Figure 9a-4. To implement, the host should cache the value of observed context parameters along with the generated drawing instructions and associate both with the feature instance. Note that a feature instance may have any number of observed context parameters, including zero.

Any changes to a context parameter requires that the host regenerate the drawing instructions for all feature instances with a matching observed context parameter. Alternatively, the host may use cached drawing instructions which were previously generated for the new value of the changed context parameter(s). Features which have no observed parameters can persist in the cache until a new portrayal catalogue is issued.

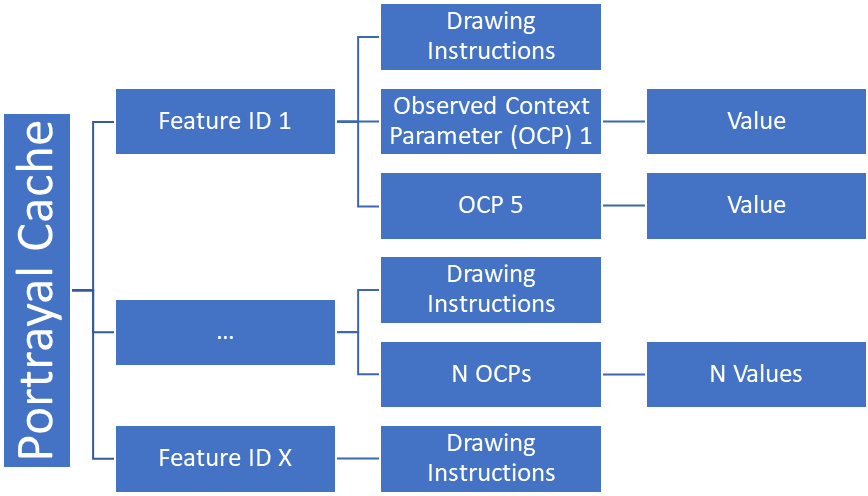


Figure 9a-4 - Notional Portrayal Cache

#### Pre-processing a Portrayal

Implementing a portrayal cache allows the host to pre-generate the drawing instructions for a given set or sets of context parameters. This would typically be implemented as part of the hosts data import functionality.

# Package Overview

There is no change to the Part 9 package overview, although most packages are unused by Part 9a due to the removal of the portrayal input schema.

# Data input schema

This part does not use a data input schema as defined in Part 9 clause 9-7. Data is passed between a 9a portrayal and a host as described in Part 13.

# Information objects

Information objects as described in Part 9 are unused in Part 9a. Instead, information associated with features to be portrayed is obtained as described in Part 13.

# Feature objects

Feature objects as described in Part 9 are unused in Part 9a. Instead, all features are retrieved from the host as described in Part 13.

# Portrayal processing

The XSLT processing described in Part 9 clause 9-10 is replaced with Lua as described in Part 13.

# Drawing Instructions

Drawing instructions are provided to the host using DEF as described in Part 13 clause 13-6.1. A single drawing instruction is equivalent to a single DEF element.

This section describes the model and schema for drawing instructions.

## The concepts of drawing instructions

### General concept

As in Part 9, the output of the portrayal engine is a set of drawing instructions. These typically link the feature instance to a symbol and/or alert reference. The geometry is either taken from the feature type or can be generated by the portrayal functions. The latter is supported by the concept of augmented geometry as described in Part 9 clause 9-11.1.13 Augmented Geometry.

The conceptual model for Part 9a drawing instructions is a command-driven state machine. This model is consistent with both SVG and S-52 DAI, but differs from Part 9 which uses stateless drawing instructions.

To implement Part 9a drawing instructions, the host must maintain state while executing the drawing instructions for a given feature instance. For example, if a drawing instruction sets a pen colour, that pen colour should also be used for subsequent draw instructions. The state must be reset prior to executing the drawing instructions for each feature instance.

## Model of the Drawing Instructions

As in Part 9, this section describes the output of the portrayal functions. A single domain-specific scripting host function, see clause 9a-14.2.1 *HostPortrayalEmit*, provides the drawing instructions for each feature instance.

Each drawing instruction is encoded in a DEF element as described in Part 13, clause 13-6.1. A drawing instruction is an ordered pair comprised of a command and a parameter list. The command is encoded in a DEF item, and the commands parameters are encoded in a DEF parameter list.

Table 9a- – DEF encoding of Drawing Instructions

|  |  |  |
| --- | --- | --- |
| **Portrayal Item** | **DEF Encoding** | **Example** |
| Drawing Instruction | Element | FillColor:CHBRN,0 |
| Command | Item | FillColor |
| Parameter List | Parameter List | CHBRN,0 |
| Parameter | Parameter | CHBRN |

Each drawing instruction contains a single case sensitive command. Each command has zero or more parameters.

There are two types of commands: drawing commands and state commands. Drawing commands instruct the host to render graphics. State commands instruct the host to set the state for subsequent drawing commands.

Each command and its parameters are described in the following sub-sections, grouped by purpose. In the tables which follow, the Type column is as described in Part 13 table 13-7. The **X-Ref** column refers to the equivalent Part 9 drawing instruction concept. The Part 9 reference may contain relevant information such as range of expected values or units.

### Drawing Commands

Drawing commands are used to render graphics. They are analogous to realizations of the Part 9 clause 9-11.2 *DrawingInstruction* class. The drawing commands are listed in Table 9a-3 and each command is described on the following pages.

Table 9a- – Drawing Commands

|  |  |  |  |
| --- | --- | --- | --- |
| **Command** | **Parameters** | **Parameter Type** | **Part 9 Reference** |
| PointInstruction | Symbol | String | 9-11.2.6  9-11.2.12 |
| LineInstruction | lineStyle | String | 9-11.2.7  9-11.2.14  9-11.2.15 |
| LineInstructionUnsuppressed | lineStyle | String | 9-11.2.7  9-11.2.14  9-11.2.15 |
| ColorFill | Token | String | 9-12.5.1.4  9-11.2.16 |
| transparency | Double |
| AreaFillReference | Reference | String | 9-12.5.1.3  9-11.2.16 |
| PixmapFill | Reference | String | 9-12.5.1.5  9-11.2.16 |
| SymbolFill | Symbol | String | 9-12.5.1.6  9-11.2.16 |
| v1 | Vector |
| v2 | Vector |
| HatchFill | direction | Vector | 9-12.5.1.7  9-11.2.16 |
| distance | Double |
| lineStyle | String |
| TextInstruction | text | String | 9-11.2.9  9-11.2.11 |
| CoverageFill | attributeCode | String | 9-11.1.12  9-11.2.10 |
| Uom | String |
| NullInstruction | - | - | 9-11.2.5 |

The graphic rendering of each drawing command can be modified by preceding state commands, as described in clause 9a-11.2.2.

##### PointInstruction:*symbol*

Instructs the host to draw a Portrayal Catalogue symbol, placed as follows:

Table 9a-4 – PointInstruction Symbol Placement

|  |  |
| --- | --- |
| **Geometry Type** | **Symbol Placement** |
| Point | At the point, then apply *LocalOffset* |
| Line | Along the line by *LinePlacement*, then apply *LocalOffset* |
| Area | At *AreaCRS*, then apply *LocalOffset*. Note that this can cause the symbol to be drawn at multiple locations |

##### LineInstruction:*lineStyle[,lineStyle,…]*

Instructs the host to stroke a line or area geometry using the specified linestyle(s).

The host must ensure line segments with lower drawing priority are suppressed (not drawn) when coincident line segments with higher drawing priority are drawn.

Each linestyle parameter refers to either a linestyle defined within the Portrayal Catalogue or to a linestyle created by a preceding *LineStyle* command.

**Note: Part 10 clause 10a-5.10.1 defines how masked spatial elements are encoded in a dataset. When executing this instruction the host must suppress the portrayal of masked spatial elements.**

##### LineInstructionUnsuppressed:*lineStyle[,lineStyle,…]*

Instructs the host to stroke a line or area geometry using the specified linestyle(s).

The line segments should be drawn without regard for coincident line segments.

Each linestyle parameter refers to either a linestyle defined within the Portrayal Catalogue or to a linestyle created by a preceding *LineStyle* command.

**Note: Part 10 clause 10a-5.10.1 defines how masked spatial elements are encoded in a dataset. When executing this instruction the host must suppress the portrayal of masked spatial elements.**

##### ColorFill:*token[,transparency]*

Instructs the host to fill an area using the given colour token and transparency. If transparency is not given, a value of zero is assumed.

##### AreaFillReference:*reference*

Instructs the host to fill an area using *areaFill* (Part 9 clause 9-13.3.10) defined within the Portrayal Catalogue.

##### PixmapFill:*reference*

Instructs the host to fill an area using *pixmap* (Part 9 clause 9-13.3.5) defined within the Portrayal Catalogue.

A preceding *AreaCRS* command may set the origin of the pattern.

##### SymbolFill:*symbol,v1,v2*

Instructs the host to fill an area using a symbol defined within the Portrayal Catalogue. A preceding *AreaCRS* command may set the origin of the pattern.

*symbol* The symbol used for the pattern.

*v1* The offset of the next symbol in the first dimension of the pattern according to the local CRS.

*v2* The offset of the next symbol in the second dimension of the pattern according to the local CRS.

##### HatchFill:*direction,distance,lineStyle[,lineStyle]*

Instructs the host to fill an area using a hatch symbol defined within the Portrayal Catalogue. Direction and distance are as defined in Part 9 clause 9-12.5.1.8.

Each linestyle parameter refers to either a linestyle defined within the Portrayal Catalogue or to a linestyle created by a preceding *LineStyle* command.

A preceding *AreaCRS* command may set the origin of the pattern.

*direction* The vector defining the direction of the set of lines.

*distance* The distance between the lines measure perpendicular to the direction.

*lineStyle* A reference to a line style used for each hatch line.

##### TextInstruction:*text[,textViewingGroup[,textPriority]]*

Instructs the host to draw the specified text placed as follows:

Table 9a- – TextInstruction Initial Placement

|  |  |
| --- | --- |
| **Geometry Type** | **Initial Placement** |
| Point | Relative to the point |
| Line | Relative to the line as determined by *LinePlacement* |
| Area | Relative to *AreaCRS*. Note that this can cause the text to be drawn at multiple locations |

Once the initial positioning is determined, the text is offset as specified by state commands *LocalOffset* and *TextVerticalOffset*. The text is aligned as specified by state commands *TextAlignHorizontal* and *TextAlignVertical*.

If preceded by a *FontReference* command the font is as specified in the Portrayal Catalogue. Otherwise the host should construct a font using the values specified by preceding *FontColor, FontSize, FontProportion, FontWeight, FontSlant, FontSerifs* and *FontStrikethrough* state commands.

*text* The text to display.

*textViewingGroup* If present, defines an additional viewing group that must be selected in order for the text to be displayed.

*textPriority* If present, defines the display prioritry of the text. If not present, the display priority indicated by the *DisplayPriority* instruction is used.

##### CoverageFill:*attributeCode[,uom]*

Instructs the host to fill a coverage using the lookup table entries created via the *LookupEntry* state command. The host must clear the coverage lookup list upon completion.

*attributeCode* Specifies which of the features attributes to use for the lookup.

*uom* If present, specifies the unit of measure for the range values in the lookup table. If not present, the range values and attribute value share the same unit of measure as defined in the Feature Catalogue.

NOTE: When associating alerts with coverage values there may or may not be portrayal elements present in the coverage lookup list.

##### NullInstruction

Used to indicate a feature is purposefully not portrayed. Can associate an alert with a feature or geometry which is not portrayed, or whose alert geometry is different than the portrayed geometry..

### State Commands

State commands are used to set or modify the state for drawing commands which follow. To implement the portrayal the host should associate each parameter of a state command with a variable; each state command modifies the value of one or more of these variables.

The host should set the initial state as indicated in the tables of the following subsections. The state should be reset prior to executing the drawing instructions for each feature instance.

For each state command listed in the following sub-sections the applicability is given; this indicates which commands use the variables set by the state command.

Table 9a-6 shows the different types of state commands.

**Table 9a-6 – Types of State Commands**

|  |  |  |
| --- | --- | --- |
| **Command Type** | **Command** | **Purpose** |
| Visibility | ViewingGroup | Modifies the visibility and drawing order of drawing commands |
| DisplayPlane |
| DrawingPriority |
| ScaleMinimum |
| ScaleMaximum |
| Transform | LocalOffset | Applies transformations to elements drawn by drawing commands |
| LinePlacement |
| AreaPlacement |
| AreaCRS |
| Rotation |
| ScaleFactor |
| Pen Style | PenColor | Modifies the appearance of lines drawn by drawing commands |
| PenWidth |
| Line Style | LineStyle | Defines linestyles for use by drawing commands |
| LineSymbol |
| Dash |
| Text Style | FontColor | Modifies the appearance of text drawn by drawing commands |
| FontSize |
| FontProportion |
| FontWeight |
| FontSlant |
| FontSerifs |
| FontUnderline |
| FontStrikethrough |
| FontUpperline |
| FontReference |
| TextAlignHorizontal |
| TextAlignVertical |
| TextVerticalOffset |
| Colour Override | OverrideColor | Overrides the colours defined within a symbol or pixmap referenced by drawing commands |
| OverrideAll |
| Geometry | SpatialReference | Defines new geometries (augmented geometry) or restricts the geometry used by drawing commands |
| AugmentedPoint |
| AugmentedRay |
| AugmentedPath |
| Polyline |
| Arc3Points |
| ArcByRadius |
| Annulus |
| ClearAugmented |
| Coverage | LookupEntry | Defines lookup entries which can be referenced by the *CoverageFill* drawing command |
| NumericAnnotation |
| SymbolAnnotation |
| CoverageColor |
| Alert | AlertReference | Associates geometries with alert catalogue entries |

#### Visibility Commands

Visibility commands affect the visibility and drawing order of all subsequent drawing commands. They correspond to attributes of the Part 9 clause 9-11.2.2 *DrawingInstruction* class.

**Table 9a-7 – Visibility Commands**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Command** | **Parameters** | **Type** | **Initial State** | **Part 9** | **Notes** |
| ViewingGroup | viewingGroup | String | "" | 9-11.1.3 | For example: 21000 |
| DisplayPlane | displayPlane | String | "" | 9-11.1.5 | For example: overRadar |
| DrawingPriority | drawingPriority | Integer | 0 | 9-11.1.6 |  |
| ScaleMinimum | scaleMinimum | Integer | max integer | 9-11.2.2 |  |
| ScaleMaximum | scaleMaximum | Integer | min integer | 9-11.2.2 |  |

##### ViewingGroup:*viewingGroup*

Sets the viewing group for drawing commands which follow.

**Applicability**: All drawing commands except *NullInstruction*

##### DisplayPlane:*displayPlane*

Sets the display plane for drawing commands which follow.

**Applicability**: All drawing commands except *NullInstruction*

##### DrawingPriority:*drawingPriority*

Sets the drawing priority for drawing commands which follow.

**Applicability**: All drawing commands except *NullInstruction*

##### ScaleMinimum:*scaleMinimum*

Sets the scale denominator defining the minimum scale for drawing commands which follow.

**Applicability**: All drawing commands except *NullInstruction*

##### ScaleMaximum:*scaleMaximum*

Sets the scale denominator defining the maximum scale for drawing commands which follow.

**Applicability**: All drawing commands except *NullInstruction*

#### Transform Commands

Transform commands apply transformations to elements, such as symbols, rendered by applicable drawing commands which follow.

Table 9a- – Transform Commands

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Command** | **Parameters** | **Type** | **Initial State** | **Part 9 Reference** |
| LocalOffset | xOffsetMM | Double | 0 | 9-12.2.2.7 |
| yOffsetMM | Double | 0 |
| LinePlacement | linePlacementMode | String | Relative | 9-12.3.1.5 |
| Offset | Double | 0.5 |
| AreaPlacement | areaPlacementMode | String | VisibleParts | 9-12.3.1.6 |
| AreaCRS | areaCRSType | String | GlobalGeometry | 9-12.5.1.9 |
| Rotation | rotationCRS | String | PortrayalCRS | 9-12.2.2.7  9-12.3.1.1  9-12.4.1.4  9-12.6.3.5 |
| Rotation | Double | 0 |
| ScaleFactor | scaleFactor | Double | 1.0 | 9-12.2.2.6 |

##### LocalOffset:*xOffsetMM,yOffsetMM*

Specifies an offset from the geographic position using the Local CRS to be applied to subsequent drawing commands.

**Applicability:** *PointInstruction, SymbolFill, TextInstruction*

##### LinePlacement:*linePlacementMode,offset*

Specifies the placement along a line for symbols or text output by subsequent drawing commands.

*linePlacementMode*

*Relative* *offset* is in homogenous coordinates, 0 for the start and 1 for the end of the curve.

*Absolute offset* specifies the distance from the start of the curve.

**Applicability:** *PointInstruction, LineInstruction, LineInstructionUnsuppressed, TextInstruction*

##### AreaPlacement:*areaPlacementMode*

Specifies the placement within an area for symbols or text output by subsequent drawing commands.

*areaPlacementMode* – one of:

*VisibleParts* The symbol or text is to be placed at a representative position in each visible part of the surface.

*Geographic* The symbol or text is to be placed at a representative position of the geographic object.

**Applicability**: *PointInstruction, TextInstruction*

##### AreaCRS:*areaCRSType*

Specifies how fill patterns output by subsequent drawing commands are anchored.

*areaCRSType* – one of:

*Global* The anchor point is consistent with a location on the drawing device; for example, starting with the corner of the screen. As the screen pans the pattern will appear to shift/move through the object on screen.

*LocalGeometry* The anchor point is consistent with the local geometry of the object being depicted, for example the upper left corner of the object. Patterns of adjacent objects may not match.

*GlobalGeometry* The anchor point of the fill pattern is defined at a common location such that patterns remain consistent relative to all area objects.

**Applicability**: *AreaFillReference, PixmapFill, SymbolFill, HatchFill, TextInstruction*

##### Rotation:*rotationCRS,rotation*

Specifies the rotation angle for symbols or text output by subsequent drawing commands.

*rotationCRS* – one of:

*GeographicCRS* A geographic CRS with axis latitude and longitude measured in degrees. *rotation* is defined as clockwise from the true north direction.

*PortrayalCRS* A Cartesian coordinate system with the y-axis pointing upwards. *rotation* is defined in degrees clockwise from the positive y-axis.

*LocalCRS* A Cartesian coordinate system originated at a local geometry. *rotation* is in degrees clockwise from the positive y-axis.

*LineCRS* A none-Cartesian coordinate system where the x-axis is following the geometry of a curve and the y-axis is perpendicular to the x-axis (positive to the left of the x-axis).

Units on the axes and for distances are millimetres. Angles are measured in degrees clockwise from the positive y-axis.

See Part 9 clause 9-12.2.2.7 for details.

**Applicability**: *PointInstruction, SymbolFill, TextInstruction, CoverageFill*

##### ScaleFactor:*scaleFactor*

Specifies a scale factor to be applied to symbols or text output by subsequent drawing commands.

**Applicability**: *PointInstruction, SymbolFill, TextInstruction, CoverageFill*

#### Line Style Commands

Line style commands create linestyles which may be referenced by subsequent drawing commands. These commands are part of the functionality of the *LineStyles* package described in Part 9 clause 9-12.4.

Table 9a- – LineStyle Commands

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Command** | **Parameters** | **Type** | **Initial State** | **Part 9** | **Notes** |
| Dash | Start | Double | - | 9-12.4.1.3 | Units: millimetres |
| Length | Double | - |
| LineSymbol | reference | Double | - | 9-12.4.1.4 |  |
| position | Double | - |
| rotation | Double | 0 |
| crsType | CRSType | LocalCRS |
| scaleFactor | Double | 1.0 |
| LineStyle | Name | String | - | 9-12.4.1.1 |  |
| intervalLength | Double | - |
| Width | Double | - |
| Token | String | - |
| transparency | Double | 0 |
| capStyle | String | Butt |
| joinStyle | String | Bevel |
| offset | Double | 0.0 |

##### Dash:*start,length*

Specifies a dash pattern for a single subsequent *LineStyle* command. Can be repeated to specify that multiple dash patterns apply to the single *LineStyle* command.

NOTE: This command does not set the state for any drawing command; it only sets the state for the *LineStyle* command.

*start* The start of the dash measured from the start of the line along the x-axis of the line CRS (units in millimetres).

*length* The length of the dash along the x-axis of the line CRS (units in millimetres).

**Applicability**: *LineStyle*

##### LineSymbol:*reference,position[,rotation[,crsType[,scaleFactor]]]*

Specifies the use of a symbol for a single subsequent *LineStyle* command. Can be repeated to specify that multiple symbols apply to the *LineStyle* command.

*reference* A reference to an external definition of the symbol graphic. This refers to an identifier of a portrayal catalogue item.

*position* The position of the symbol measured from the start of the repeating interval, along the x-axis of the line CRS (units in millimetres).

*rotation* The rotation angle of the symbol.

*crsType* The type of the CRS where the symbol has to be transformed. Possible values are LocalCRS and LineCRS.

*scaleFactor* The scale factor of the symbol.

**Applicability**: *LineStyle*

##### LineStyle:*name,intervalLength,width,token[,transparency[,capStyle[,joinStyle[,offset]]]]*

Creates a named linestyle for use by subsequent drawing commands. May be preceded by zero or more *Dash* and/or *LineSymbol* commands which apply to the linestyle. If no *Dash* commands precede the *LineStyle* command, a solid line is created.

*name* A name assigned to the linestyle and used to reference the linestyle from a *LineInstruction*. In the event of a name collision between a Portrayal Catalogue linestyle and a *LineStyle* command, the *LineStyle* command takes precedence.

*intervalLength* The length of a repeating interval of the line style along the x-axis of the line CRS (units in mm). Can be omitted if a solid is being defined.

*width* Pen width in mm used to draw this line style.

*token* Specifies the colour used to draw this line style.

*transparency* Specifies the transparency used to draw this line style.

*capStyle* The decoration that is applied where a line segment ends. One of *Butt*, *Square*, or *Round*. See Part 9 clause 9-12.4.1.8 *CapStyle*.

*joinStyle* The decoration that is applied where two line segments meet. One of *Bevel*, *Miter*, or *Round*. See part 9 clause 9-12.4.1.7 *JoinStyle*.

*offset* An offset perpendicular to the direction of the line. The value refers to the y-axis of the line CRS (positive to the left, millimetres).

**Applicability**: *LineInstruction, LineInstructionUnsuppressed, HatchFill*

#### Text Style Commands

Text style commands modify the appearance of text drawn by subsequent drawing commands.

Table 9a- – Text Style Commands

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Command** | **Parameters** | **Type** | **Initial State** | **Part 9** | **Notes** |
| FontColor | token | String | "" | 9-12.6.3.8  9-12.2.2.3 | Opaque |
| transparency | Double | 0 |
| FontBackgroundColor | token | String | "" | 9-12.6.3.8  9-12.2.2.3 | Transparent |
| transparency | Double | 1 |
| FontSize | bodySize | Double | 10 | 9-12.6.3.8 |  |
| FontProportion | proportion | String | Proportional | 9-12.6.3.11 |  |
| FontWeight | weight | String | Medium | 9-12.6.3.10 |  |
| FontSlant | slant | String | Upright | 9-12.6.3.9 |  |
| FontSerifs | serifs | Boolean | false | 9-12.6.3.2 |  |
| FontUnderline | underline | Boolean | false | 9-12.6.3.12 |  |
| FontStrikethrough | strikethrough | Boolean | false | 9-12..6.3.12 |  |
| FontUpperline | upperline | Boolean | false | 9-12.6.3.12 |  |
| FontReference | fontReference | String | "" | 9-12.6.3.3 |  |
| TextAlignHorizontal | horizontalAlignment | String | Start | 9-12.6.3.14 |  |
| TextAlignVertical | verticalAlignment | String | Baseline | 9-12.6.3.13 |  |
| TextVerticalOffset | verticalOffset | Double | 0 | 9-12.6.3.8 |  |

##### FontColor:*token[,transparency]*

Specifies the colour and transparency for glyphs drawn by subsequent drawing commands.

**Applicability**: *TextInstruction*

##### FontBackgroundColor:*token,transparency*

Specifies the colour and transparency used to fill the rectangle surrounding text drawn by subsequent drawing commands.

**Applicability**: *TextInstruction, CoverageFill*

##### FontSize:*bodySize*

Specifies the size in points for text drawn by subsequent drawing commands.

**Applicability**: *TextInstruction, CoverageFill*

##### FontProportion:*proportion*

Specifies a font proportion to be used for text drawn by subsequent drawing commands.

*proportion* – one of:

*MonoSpaced* A font where all typefaces have the same width should be selected. Also known as 'typewriter' fonts.

*Proportional* A font where each typeface can have a different width should be selected.

**Applicability**: *TextInstruction, CoverageFill*

##### FontWeight:*weight*

Specifies the font thickness for text drawn by subsequent drawing commands.

*weight* – one of:

*Light* Typefaces are depicted as thin (standard).

*Medium* Typefaces are depicted thicker than *Light*, but not as thick as *Bold*.

*Bold* Typefaces are depicted more prominently (**Bold**).

**Applicability**: *TextInstruction, CoverageFill*

##### FontSlant:*slant*

Specifies the slant to be used for text drawn by subsequent drawing commands.

*slant* – one of:

*Upright* Typefaces are upright.

*Italics* Typefaces are slanted to the right.

**Applicability**: *TextInstruction, CoverageFill*

##### FontSerifs:*serifs*

Specifies whether the font used for text drawn by subsequent drawing commands should contain serifs.

**Applicability**: *TextInstruction, CoverageFill*

##### FontUnderline:*underline*

Specifies whether text drawn by subsequent drawing commands should be underlined.

**Applicability**: *TextInstruction*

##### FontStrikethrough:*strikethrough*

Specifies whether text drawn by subsequent drawing commands should be depicted with a line through the center of the text.

**Applicability**: *TextInstruction*

##### FontUpperline:*upperline*

Specifies whether text drawn by subsequent drawing commands should be depicted with a line above the text.

**Applicability**: *TextInstruction*

##### FontReference:*fontReference*

Specifies text drawn by subsequent drawing commands should be depicted using the specified font from the Portrayal Catalogue. *fontReference* is the identifier for the external file within the Portrayal Catalogue.

**Applicability**: *TextInstruction*

##### TextAlignHorizontal:*horizontalAlignment*

Specifies the text placement relative to the anchor point in the horizontal direction for subsequent drawing commands.

*horizontalAlignment* – one of:

*Start* The anchor point is at the start of the text.

*Center* The anchor point is at the (horizontal) centre of the text.

*End* The anchor point is at the end of the text.

**Applicability**: *TextInstruction*

##### TextAlignVertical:*verticalAlignment*

Specifies the text placement relative to the anchor point in the vertical direction for subsequent drawing commands.

*verticalAlignment* – one of:

*Top* The anchor point is at the top of the em square.

*Center* The anchor point is at the (vertical) centre of the em square.

*Baseline* The anchor point is at the baseline of the font.

*Bottom* The anchor point is at the bottom of the em square.

**Applicability**: *TextInstruction*

##### TextVerticalOffset:*verticalOffset*

Specifies the vertical offset in mm above the anchor point of the text drawn by subsequent *TextInstruction* commands. Used to generate subscripts or superscripts.

**Applicability**: *TextInstruction*

#### Colour Override Commands

Colour override commands modify the colour of symbols and pixmaps drawn by subsequent drawing commands.

Table 9a- – Colour Override Commands

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Command** | **Parameters** | **Type** | **Initial State** | **Part 9** | **Notes** |
| OverrideColor | colorToken | String | N/A | 9-12.2.2.6  9-12.3.1.2 |  |
| colorTransparency | Double | N/A |
| overrideToken | String | N/A |
| overrideTransparency | Double | N/A |
| OverrideAll | token | String | N/A | 9-12.2.2.5  9-12.3.1.1 |  |
| transparency | Double | N/A |
| ClearOverride |  |  |  |  |  |

##### OverrideColor:*colorToken,colorTransparency,overrideToken,overrideTransparency*

Specifies an override colour which should be used to replace the original colour in a symbol or pixmap rendered via a drawing command. This command can be issued multiple times to specify more than one color substitution.

**Applicability**: *PointInstruction, AreaFillReference, PixmapFill, SymbolFill*

##### OverrideAll:*token,transparency*

Substitutes all non-transparent colours with the given colour. This command supercedes any *OverrideColor* commands.

**Applicability**: *PointInstruction, AreaFillReference, PixmapFill, SymbolFill*

##### ClearOverride

Removes all colour substitutions.

**Applicability**: *PointInstruction, AreaFillReference, PixmapFill, SymbolFill*

#### Geometry Commands

All drawing commands defined in clause 9a-11.2.1 reference geometries. Normally, this is the geometry of the feature (analogous to Part 9 clause 9-11.2.3 *DrawingInstruction::featureReference*). The host determines the features geometry using the feature reference provided when drawing instructions are returned from the portrayal via *HostPortrayalEmit* as described in clause 9a-14.2.1. The geometry commands defined in this section allow the normal behaviour to be overridden.

One method of overriding the normal behaviour is to constrain drawing commands so that they reference either individual geometric elements of a feature; or any other geometries defined in the dataset (analogous to Part 9 clause 9-11.2.3 *DrawingInstruction::spatialReference*).

The second method of overriding the normal behaviour is to create an augmented geometry (Part 9 clause 9-11.1.13 Augmented Geometry) using a geometry command. Augmented geometry is used when the spatial to be referenced is not present in the dataset. Augmented geometry created by a geometry command will be used by subsequent drawing commands, overriding the features geometry.

This Part does not define separate augmented drawing instructions as in Part 9. Instead, all drawing commands are to use augmented geometry whenever augmented geometry is available.

To deterimine the geometry to be used by a drawing command:

* If an augmented geometry command precedes the drawing command, the most recently defined augmented geometry should be used.
* Otherwise, if the spatial references list is not empty, the drawing command is applied to each spatial reference.
* Otherwise, the features geometry should be used.

To implement augmented paths, the host should maintain a segment list into which the geometries created by the *Polyline*, *Arc3Points*, *ArcByRadius* and *Annulus* commands are placed. This list maintains the order in which the geometries are created.

Applied geometry commands are removed via the *ClearGeometry* command, which also clears the segment list. Using *ClearGeometry* allows switching between referencing the features geometry, augmented geometry, and spatial references.

The geometry commands are listed in the table below. The type *point* indicates a pair of doubles are passed as parameters.

Table 9a- – Geometry Commands

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Command** | **Parameters** | **Type** | **Initial State** | **Part 9** | **Notes** |
| SpatialReference | reference | String | - | 9-11.2.4 |  |
| forward | Boolean | true |
| AugmentedPoint | crs | CRSType | - | 9-11.2.12 |  |
| x | Point | - |
| y | - |
| AugmentedRay | crsDirection | CRSType | - | 9-112.14 |  |
| direction | Double | - |
| crsLength | CRSType | - |
| length | Double | - |
| AugmentedPath | crsPosition | CRSType | - | 9-11.2.15 |  |
| crsAngle | CRSType | - | 9-11.2.15 |  |
| crsDistance | CRSType | - | 9-11.2.15 |  |
| Polyline | point1 | Point[] | - | 9-12.2.2.11 |  |
| … |
| pointN |
| Arc3Points | startPointX | Point | - | 9-12.2.2.13 |  |
| startPointY |
| medianPointX | Point | - |
| medianPointY |
| endPointX | Point | - |
| endPointY |
| ArcByRadius | centerX | Point | - | 9-12.2.2.14 |  |
| centerY |
| radius | Double | - |
| startAngle | Double | 0 |
| angularDistance | Double | 360 |
| Annulus | centerX | Point | - | 9-12.2.2.15 |  |
| centerY |
| outerRadius | Double | - |
| innerRadius | Double | outerRadius |
| startAngle | Double | 0 |
| angularDistance | Double | 360 |
| ClearGeometry | - | - | - | - |  |

##### SpatialReference:*reference[,forward]*

Specifies a reference to the spatial type components of the feature that defines the geometry used for the depiction of drawing commands which follow. Not used when the entire geometry of the feature should be depicted. Each time this command is called, a new spatial reference is added to the spatial references list maintained by the host. The spatial references list can be cleared by calling *ClearGeometry.*

*reference* The identifier of the spatial type as defined in Part 13 clause 13-8.

*forward* If true the spatial object is used in the direction in which it is stored in the data. Only applies to curves and should be ignored for all other spatial types.

**Applicability**: All drawing commands

##### AugmentedPoint:*crs,x,y*

Specifies a position*.* Clears any active *AugmentedRay* and *AugmentedPath* instructions.

*crs* – one of:

*GeographicCRS* A geographic CRS with axis latitude and longitude measured in degrees.

*PortrayalCRS* A Cartesian coordinate system with the y-axis pointing upwards. Units on the axes and for distances are millimetres.

*LocalCRS* A Cartesian coordinate system originated at a local geometry. Units on the axes and for distances are millimetres.

*x,y* Coordinates of the point.

**Applicability**: *PointInstruction,* *TextInstruction, NullInstruction*

##### AugmentedRay:*crsDirection,direction,crsLength,length*

Augments the geometry of a point feature. Specifies a line from the position of the point feature to another position. The position is defined by the direction and the length attributes. Clears any active *AugmentedPoint* and *AugmentedPath* instructions.

If *crsDirection* is *PortrayalCRS* or *LocalCRS* then *crsLength* must be *PortrayalCRS* or *LocalCRS.* Similarly, if *crsLength* is *GeographicCRS* then *crsDirection* must be *GeographicCRS.*

*crsDirection* and *crsLength* – each one of:

*GeographicCRS* Angles are defined clockwise from the true north direction. Distances will be measured in metres.

*PortrayalCRS* A Cartesian coordinate system with the y-axis pointing upwards. Units on the axes and for distances are millimetres. Angles are measured in degrees clockwise from the positive y-axis.

*LocalCRS* A Cartesian coordinate system originated at a local geometry. Units on the axes and for distances are millimetres. Angles are measured in degrees clockwise from the positive y-axis.

*direction* The direction of the ray relative to the CRS specified.

*length* The length of the ray in units depending on the CRS specified.

**Applicability**: *LineInstruction, LineInstructionUnsuppressed, TextInstruction, NullInstruction*

##### AugmentedPath:*crsPosition,crsAngle,crsDistance*

Instructs the host to gather all segments previously created by *Polyline*, *Arc3Points*, *ArcByRadius* and *Annulus* commands and group them as a single augmented geometry. The host must then clear the segment list. Clears any active *AugmentedPoint* and *AugmentedRay* instructions.

To implement an augmented path, the host must maintain a segment list. Each call to *Polyline*, *Arc3Points*, *ArcByRadius* and *Annulus* results in the host placing the geometry on the segment list. These items taken in order they are added to the segment list define the augmented path.

The CRS is specified separately for positions, angles and distances.

*crsPosition, crsAngle* and *crsDistance* – each one of:

*GeographicCRS* A geographic CRS with axis latitude and longitude measured in degrees. Angles are defined clockwise from the true north direction. Distances will be measured in metres.

*PortrayalCRS* A Cartesian coordinate system with the y-axis pointing upwards. Units on the axes and for distances are millimetres. Angles are measured in degrees clockwise from the positive y-axis.

*LocalCRS* A Cartesian coordinate system originated at a local geometry. Units on the axes and for distances are millimetres. Angles are measured in degrees clockwise from the positive y-axis.

**Applicability**: All drawing commands except *PointInstruction*

##### Polyline:*positionXstart,positionYstart,positionXto,positionYto[,positionXto,positionYto…]*

Instructs the host to add a polyline to the segment list.

*positionXstart,positionYstart,positionXto,positionYto* Coordinates of the segments of the polyline.

**Applicability**: *AugmentedPath*

##### Arc3Points:*startPointX,startPointY,medianPointX,medianPointY,endPointX,endPointY*

Instructs the host to add an arc defined by three points to the segment list.

*startPointX,startPointY* The point where the arc starts.

*medianPointX,medianPointY* An arbitrary point on the arc.

*endPointX,endPointY* The point where the arc ends.

**Applicability**: *AugmentedPath*

##### ArcByRadius:*centerX,centerY,radius[,startAngle,angularDistance]*

Instructs the host to add an arc defined by a radius to the segment list.

*centerX,centerY* The centre of the arc.

*radius* The radius of the circle.

*startAngle,angularDistance* The sector defining where the arc starts and ends. If not present the arc is a full circle.

**Applicability**: *AugmentedPath*

##### Annulus:*centerX,centerY,outerRadius[,innerRadius[,startAngle,angularDistance]]*

Instructs the host to add an annulus to the segment list. An annulus is a ring-shaped region bounded by two concentric circles. It can optionally be bounded by two radii of the circle.

Note that the presence of *startAngle* and *angularDistance* parameters does not imply that *innerRadius* must be present. The following is a valid command: Annulus:0,1,2.34,,56,78

*centerX,centerY* The centre of the annulus.

*outerRadius* The radius of the larger circle.

*innerRadius* The radius of the smaller circle. If not present the segment describes a sector of a circle.

*startAngle,angularDistance* The sector of an annulus segment.

**Applicability**: *AugmentedPath*

##### ClearGeometry

Clears any preceding geometry commands and empties the segment and spatial references lists.

**Applicability**: *AugmentedPath, SpatialReference*

#### Coverage Commands

Coverage commands define lookup entries which are referenced by the *CoverageFill* drawing command. These commands are part of the functionality of the *Coverage* package described in Part 9 clause 9-12.7. The coverage commands are listed in Table 9a-13 below.

Table 9a- - Coverage Commands

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Command** | **Parameters** | **Type** | **Initial State** | **Part 9** | **Notes** |
| NumericAnnotation | decimals | Integer | - | 9-12.7.4.4 |  |
| championChoice | ChampionChoice | - |
| buffer | Double | 0 |
| SymbolAnnotation | symbolRef | String | - | 9-12.7.4.5 |  |
| rotationAttribute | String | - |
| scaleAttribute | String | - |
| rotationCRS | CRSType | PortrayalCRS |
| rotationOffset | Double | 0 |
| rotationFactor | Double | 1 |
| scaleFactor | Double | 1 |
| CoverageColor | startToken | String | - | 9-12.7.4.3 |  |
| startTransparency | Double | 0 |
| endToken | String | - |
| endTransparency | Double | 0 |
| penWidth | Double | 0 |
| LookupEntry | label | String | - | 9-12.7.4.2  1-4.5.3.4 |  |
| lower | Double | - |
| upper | Double | - |
| closure | S100\_IntervalType | - |

##### NumericAnnotation:*decimals,championChoice[,buffer]*

Specifies the numeric representation of a coverage instruction. When executing the *CoverageFill* drawing command, the numeric value should be drawn using the currently defined font. However, instead of using the font colour set by *FontColor, CoverageColor* should be used.

*decimals* Number of decimal digits to show in subscript.

*championChoice* – one of:

*Largest* Display the largest value in case of collision.

*Smallest* Display the smallest value in case of collision.

*buffer* Buffer to apply for collision detection in portrayal units.

**Applicability**: *LookupEntry*

##### SymbolAnnotation:*symbolRef,rotationAttribute,scaleAttribute[,rotationCRS,rotationOffset[,rotationFactor[,scaleFactor]]]*

Specifies the symbol representation of a coverage instruction.

*symbolRef* The symbol from the Portrayal Catalogue to draw.

*rotationAttribute* The attribute code of the Coverage Attribute to use for the symbol rotation value.

*scaleAttribute* The attribute code of the Coverage attribute to use for scaling the symbol size.

*rotationCRS* Specifies the coordinate reference system for the rotation.

*rotationOffset* Used to adjust the ‘rotationAttribute’ value by addition before applying. This offset is applied after *rotationFactor.* If no *rotationAttribute* is given, this value represents the rotation value to apply to the symbol. A value of 0 indicates no adjustment.

*rotationFactor* Used to adjust the ‘rotationAttribute’ value by multiplication before applying. This factor is applied before *rotationOffset.* A value of 1 indicates no adjustment.

*scaleFactor* Used to adjust the ‘scaleAttribute’ value by multiplication before applying. A value of 1 indicates no adjustment.

EXAMPLE: Assume a coverage has wind speed and direction attributes and the portrayal wishes to draw an arrow showing wind direction and whose length is proportion to the wind speed. In this example the wind direction indicates the compass direction of where the wind is coming from and the portrayal wants to indicate the direction the wind is blowing towards. Additionally, the portrayal wants a 20 knot wind speed to be indicated by drawing the arrow at its normal scale. In this case the portrayal needs to rotate the arrow by 180 degrees and scale the arrow by 1/20. The following commands could be used to accomplish the portrayal of the arrow:

SymbolAnnotation:ARROW,windDirection,windSpeed,PortrayalCRS,180,1.0,0.05;  
LookupEntry:Wind,0,360,closedInterval;  
CoverageFill:windDirection

**Applicability**: *LookupEntry*

##### CoverageColor:*startToken,startTransparency[,endToken,endTransparency][,penWidth]*

Specifies the colour range to use for a coverage instruction. If *endToken* and *endTransparency* are not specified, then a single colour is used.

*startToken,startTransparency* The color to assign to the matching range or to use as start point in a color ramp when ‘endColor’ is defined.

*endToken,endTransparency* If given, the colour to use as the stopping point in a color ramp. The range of values is spread linearly across the range of colours from ‘startColor’ to ‘endColor’ to produce a gradient effect.

*penWidth* Pen width to apply for dot color used for discrete points.

**Applicability**: *LookupEntry*

##### LookupEntry:*label,lower,upper,closure*

Creates a lookup entry for use by a single subsequent *CoverageFill* drawing command. This instruction is used to associate preceding *NumericAnnotation, SymbolAnnotation* and *CoverageColor* commands with a single lookup table entry.

NOTE: To support associating alerts with coverage values which differ from portrayal, preceding *NumericAnnotation*, *SymbolAnnotation*, or *CoverageColor* commands are not required.

NOTE: subsequent *LookupEntry* commands require redefinition of *NumericAnnotation*, *SymbolAnnotation*, and *CoverageColor*; for example the state of the other coverage commands should be reset after processing *LookupEntry*.

*label* String used as a display label or legend field.

*lower* Lower value of lookup range.

*upper* Upper value of lookup range.

*Closure* Interval closure for range. See Part 1 clause 1-4.5.3.4.

**Applicability**: *CoverageFill*

#### Alert Commands

Alert commands associate alert information with the geometry of a drawing instruction.

Table 9a-14 – Alert Commands

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Command** | **Parameters** | **Type** | **Initial State** | **Part 9** | **Notes** |
| AlertReference | alertReference | string | - | 9-11.2.2  9-11.2.5 |  |
| plan | string | - |
| monitor | string | - |

##### AlertReference*[:alertReference[,plan[,monitor]]]*

Specifies alert information to be associated with the geometry of drawing instructions which follow.

*alertReference* A reference to an alert in the alert catalogue. If not provided, clears any previously applied alert reference.

*plan* The viewing group the alert highlight is assigned to when active in route planning. If not provided, no viewing group is assigned and the highlight cannot be disabled.

*monitor* The viewing group the alert highlight is assigned to when active in route monitoring. If not provided, no viewing group is assigned and the highlight cannot be disabled.

**Applicability**: All drawing commands

# Symbol Definitions

The symbol definitions described in Part 9 clause 9-12 are implemented within the Model of the Drawing Instructions (see clause 9a-11.2).

# The Portrayal Library

There is no change to the organization structure of the portrayal library as defined in Part 9 clause 9-13.2. The “Rules” folder XSLT contents of Part 9 clause 9-13.2 are replaced with Lua script files. *FileType:rules d*escribed in Part 9 clause 9-13.3.27 is used to identify each of the Lua script files.

# Portrayal Domain Specific Functions

The Lua portrayal is an instance of a Part 13 scripting domain. The functions described below are specific to this scripting domain; they are domain specific functions to be used in conjunction with the standard functions detailed in Part 13.

## Portrayal Domain Specific Catalogue Functions

The functions listed on the following clauses are implemented within the Portrayal Catalogue rule files. They can be called by the host, and augment the standard catalogue functions described in Part 13.

### Boolean PortrayalMain(String[] featureIDs)

**Return Value:**

true Portrayal completed successfully.

false Portrayal was terminated by the host (host returned false from *HostPortrayalEmit*).

**Parameters:**

*featureIDs*: String[]

An array containing the IDs of the features for which to generate drawing instructions. If this parameter is nil (or missing), the portrayal will generate drawing instructions for all feature instances in the dataset.

**Remarks:**

This function is called by the host to start the portrayal process for a dataset instance. Subsequently, the portrayal scripts will repeatedly call *HostPortrayalEmit*, providing the host with the drawing instructions for each feature instance portrayed.

The function returns once the portrayal scripts have run to completion; an error is thrown; or the host returns false from *HostPortrayalEmit*.

If using a portrayal cache as outlined in clause 9a-5.2.2.1, the host only needs to pass in uncached featureIDs, or featureIDs associated with context parameters whose values have changed.

### void PortrayalInitializeContextParameters(ContextParameter[] c*ontextParameters*)

**Return Value:**

void

**Parameters:**

*contextParameters*: ContextParameter[]

An array of ContextParameter objects.

**Remarks:**

Provides the portrayal scripts with the default value for each portrayal context parameter defined within the Portrayal Catalogue. *PortrayalCreateContextParameter* should be used to create each entry. The host is responsible for retrieving the portrayal context parameters from the Portrayal Catalogue.

### ContextParameter PortrayalCreateContextParameter(String *contextParameterName*, String *contextParameterType*, String *defaultValue*)

**Return Value:**

A ContextParameter storing the *defaultValue* with the *contextParameterName*.

**Parameters:**

*contextParameterName*: String

The name of a portrayal context parameter. Valid names are defined in the Portrayal Catalogue.

*contextParameterType*: String

The type of the portrayal context parameter. Valid values are *Boolean, Integer, Real, Text* and *Date*.

*defaultValue*: String

The default value for the portrayal context parameter. This value is encoded as described in Part 13 clause 13-8.1.

**Remarks:**

Creates a ContextParameter object for use within the scripting environment.

### void PortrayalSetContextParameter(String *contextParameterName*, String *value*)

**Return Value:**

void

**Parameters:**

*contextParameterName*: String

The name of a portrayal context parameter.

*value*: String

The new value for the portrayal context parameter. This value is encoded as described in Part 13 clause 13-8.1.

**Remarks:**

Allows the host to modify the value of a portrayal context parameter. The context parameter must be created via *PortrayalInitializeContextParameters* prior to being modified.

## Portrayal Domain Specific Host Functions

The host must implement the function described in the following clause in order to support portrayal. This function is called from the portrayal domain specific catalogue functions, and augments the standard host functions described in Part 13.

### Boolean HostPortrayalEmit(String *featureID*, String *drawingInstructions*, String *observedParameters*)

**Return Value:**

True Continue script processing. The portrayal engine will continue to process feature instances.

False Terminate script processing. No additional feature instances will be processed by the portrayal engine.

**Parameters:**

*featureID*: String

Used by the host to uniquely identify a feature instance.

*drawingInstructions*:String

All of the drawing instructions generated for the feature instance identified by *featureID*. This string is in Data Exchange Format (DEF) as described in Part 13.

*observedParameters*: String

The context parameters that were observed during the generation of the drawing instructions for this feature. This string is in DEF.

**Remarks:**

This function is called from the Portrayal Catalogue once per feature instance to provide drawing instructions to the host.

The host can optionally use the observed context parameters to perform drawing instruction caching.

Page intentionally left blank