S-100 – Part 9a

Portrayal (Lua)

Page intentionally left blank

Contents

9A-1 Scope

This part defines the additions and changes to S-100 Part 9 necessary to implement portrayal using the scripting mechanism defined in S-100 Part 50. Products which specify use of a portrayal catalogue as described in this part must also require implementation of S-100 Part 50.

9A-2 Conformance

This part of the specification conforms to S-100 part 50.

9A-3 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/

9A-4 Portrayal Catalogue

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

9A-5 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.

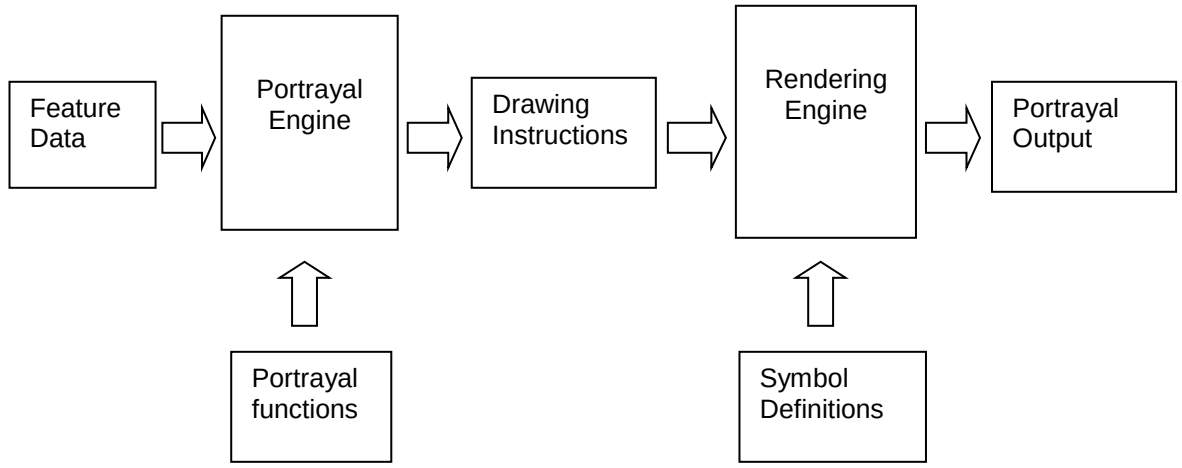

Figure 9A-1 - General portrayal model

9A-5.1 The Portrayal Process

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

Table 9A-1 - 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*

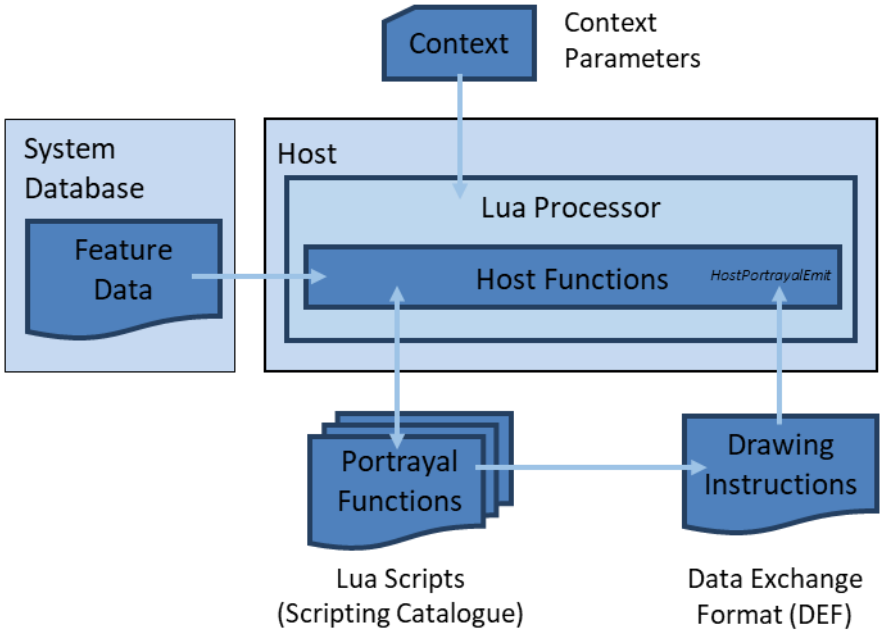

Figure 9A-2 - Portrayal process

9A-5.2 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

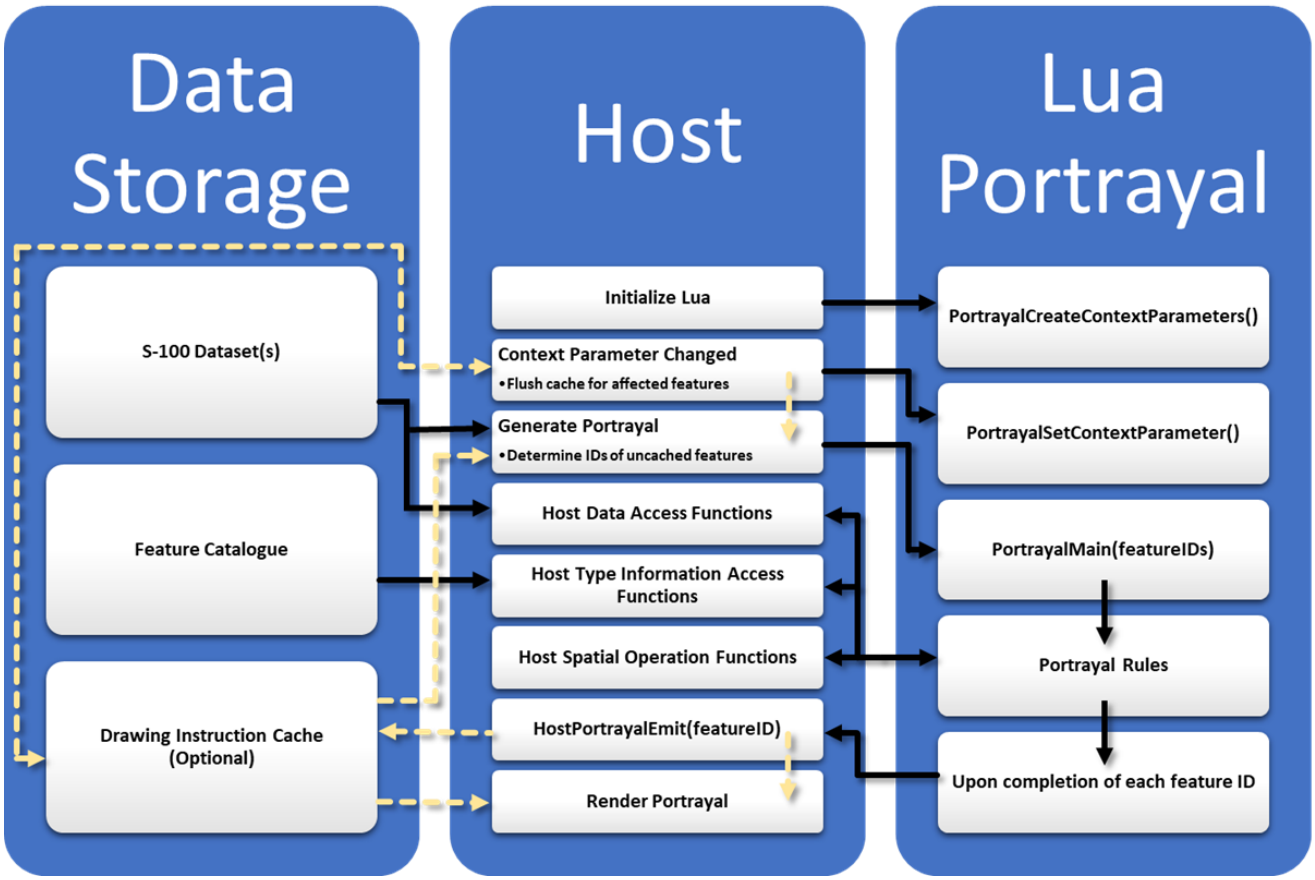

Figure 9A-3 - Lua Portrayal Process

9A-5.2.1 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*.

9A-5.2.2 Generating a Portrayal

Portrayal script function 9A-14.1.1 *PortrayalMain* 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 9A-14.2.1 *HostPortrayalEmit* 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*.

9A-5.2.2.1 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 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 9-13.3.20.

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.

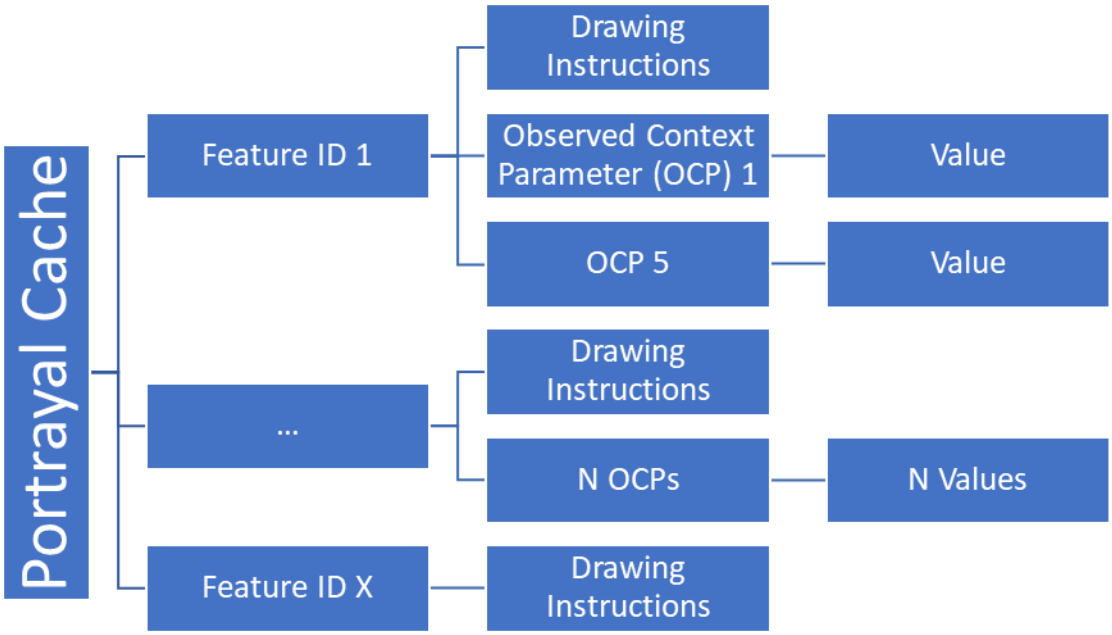

Figure 9A-4 - Notional Portrayal Cache

9A-5.2.2.2 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.

9A-6 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.

9A-7 Data input schema

This part does not use a data input schema as defined in 9-7. Data is passed between a 9a portrayal and a host as described in Part 50.

9A-8 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 50.

9A-9 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 50.

9A-10 Portrayal processing

The XSLT processing described in 9-10 is replaced with Lua as described in Part 50.

9A-11 Drawing Instructions

Drawing instructions are provided to the host using DEF as described in 50-7.1. A single drawing instruction is equivalent to a single DEF element.

This section describes the model and schema for drawing instructions.

9A-11.1 The concepts of drawing instructions

9A-11.1.1 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 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 9-11.1.12 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.

9A-11.2 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, 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 50-7.1. 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-2 – 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 50 table 3. 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.

9A-11.2.1 Drawing Commands

Drawing commands are used to render graphics. They are analogous to realizations of the 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-3 - Drawing Commands

Command	Parameters	Parameter Type	X-Ref
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	9-11.1.11 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 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.

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.

ColorFill:*token,transparency*

Instructs the host to fill an area using the given colour token and transparency.

AreaFillReference:*reference*

Instructs the host to fill an area using a 9-13.3.9 *areaFill* defined within the portrayal catalogue.

PixmapFill:*reference*

Instructs the host to fill an area using a 9-13.3.5 *pixmap* 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 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*

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

Table 9A-5 - 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.

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.

NullInstruction

The host performs no action. Used to indicate a feature is purposefully not portrayed.

9A-11.2.2 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

	AugmentedPoint	restricts the geometry used by drawing commands.
	AugmentedRay	
	AugmentedPath	
	Polyline	
	Arc3Points	
	ArcByRadius	
	Annulus	
	ClearAugmented	
Coverage	LookupEntry	Defines lookup entries which can be referenced by the *CoverageFill* drawing command.
	NumericAnnotation	
	SymbolAnnotation	
	CoverageColor	

9A-11.2.2.1 Visibility Commands

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

Table 9A-7 - Visibility Commands

Command	Parameters	Type	Initial State	X-Ref	Notes
ViewingGroup	viewingGroup	string	""	9-11.1.3	e.g. 21000
DisplayPlane	displayPlane	string	""	9-11.1.4	e.g. overRadar
DrawingPriority	drawingPriority	integer	0	9-11.1.5	
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*

9A-11.2.2.2 Transform Commands

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

Table 9A-8 - Transform Commands

Command	Parameters	Type	Initial State	X-Ref
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	Portrayal	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 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*

9A-11.2.2.3 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 9-12.4.

Table 9A-9 – LineStyle Commands

Command	Parameters	Type	Initial State	X-Ref	Notes
Dash	start	double	-	9-12.4.1.3	Units: mm
	length	double	-		
	intervalLength	double	start+length		
LineSymbol	reference	string	-	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	
	width	double	-		
	token	string	-		
	transparency	double	0		
	capStyle	string	butt		
	joinStyle	string	bevel		
	offset	double	0.0		

Dash:*start,length[,intervalLength]*

Specifies a dash pattern for a single subsequent *LineStyle* command. Can be repeated to specify 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 end of the last dash interval along the x-axis of the line CRS (units in mm).

length

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

intervalLength

The overall length of the dashed line (including the blank areas) along the x-axis of the line CRS (units in mm) include any space after the dash. If omitted, this indicates there is no space after dash and that the next dashed interval for the linestyle abuts the end of the dash.

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 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 mm).

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,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.

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.

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 9-12.4.1.8 *CapStyle*.

joinStyle

The decoration that is applied where two line segments meet. One of *Bevel*, *Miter*, or *Round*. See 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, mm).

Applicability: *LineInstruction, LineInstructionUnsuppressed, HatchFill*

9A-11.2.2.4 Text Style Commands

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

Table 9A-10 Text Style Commands

Command	Parameters	Type	Initial State	X-Ref	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*

9A-11.2.2.5 Colour Override Commands

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

Table 9A-11 - Colour Override Commands

Command	Parameters	Type	Initial State	X-Ref	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*

9A-11.2.2.6 Geometry Commands

All drawing commands defined in 9A-11.2.1 except *NullInstruction* render geometries. Normally, this is the geometry of the feature (analogous to 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 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 render either individual geometric elements of a feature, or any other geometries defined in the dataset (analogous to 9-11.2.3 *DrawingInstruction::spatialReference*).

The second method of overriding the normal behaviour is to create an augmented geometry (9-11.1.12 Augmented Geometry) using a geometry command. Augmented geometry is used when the spatial to be portrayed is not present in the dataset. Augmented geometry created by a geometry command will be rendered 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 be rendered using augmented geometry whenever augmented geometry is available.

To deterimine the geometry to be rendered 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 is applied to each spatial reference.
- Otherwise, the features geometry should be rendered.

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 portrayal to switch between rendering 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-12 – Geometry Commands

Command	Parameters	Type	Initial State	X-Ref	Notes
SpatialReference	reference	string	-	9-11.2.4	
	forward	boolean	true		
AugmentedPoint	crs	CRSType	-	9-11.2.12	
	x	point	-		
	y		-		
AugmentedRay	crs	CRSType	-	9-112.14	
	direction	double	-		
	length	double	-		
AugmentedPath	crs	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 50-9.

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 except *NullInstruction*

AugmentedPoint:*crs,x,y*

Specifies the position of any following *PointInstruction* or *TextInstruction.*

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*

AugmentedRay:*crs,direction,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.

crs – 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*

AugmentedPath:*crs*

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.

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.

crs – 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* and *NullInstruction.*

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*

9A-11.2.2.7 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 9-12.7. The coverage commands are listed in the table below.

Table 9A-13 - Coverage Commands

Command	Parameters	Type	Initial State	X-Ref	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 knott 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: subsequent *LookupEntry* commands require redefinition of *NumericAnnotation*, *SymbolAnnotation*, and *CoverageColor*; e.g. 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 1-4.5.3.4.

Applicability: *CoverageFill*

9A-12 Symbol Definitions

The symbol definitions described in 9-12 are implemented within the 9A-11.2 Model of the Drawing Instructions.

9A-13 The portrayal library

There is no change to the organization structure of the portrayal library as defined in section 9-13.2. The “Rules” folder XSLT contents of section 9-13.2 are replaced with Lua script files. *FileType:rules d*escribed in 9-13.3.25 is used to identify each of the Lua script files.

This part adds a Lua file format identifier to 9-13.3.24 *FileFormat*:

9A-13.1 FileFormat

Role Name	Name	Description
Type	FileFormat	The format of an external file
Enumeration	xml	
Enumeration	svg	
Enumeration	xslt	
Enumeration	ttf	
Enumeration	css	
Enumeration	lua	

9A-14 Portrayal Domain Specific Functions

The Lua portrayal is an instance of a Part 50 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 50.

9A-14.1 Portrayal Domain Specific Catalogue Functions

The functions listed on the following pages are implemented within the portrayal catalogue rule files. They can be called by the host, and augment the standard catalogue functions described in Part 50.

9A-14.1.1 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 9A-5.2.2.1, the host only needs to pass in uncached featureIDs, or featureIDs associated with context parameters whose values have changed.

9A-14.1.2 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.

9A-14.1.3 ContextParameter PortrayalCreateContextParameter(string *contextParameterName*, variant *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.

defaultValue: variant

The default value for the portrayal context parameter. The type of this parameter must match the type specified in the portrayal catalogue.

Remarks

Creates a ContextParameter object for use within the scripting environment.

9A-14.1.4 void PortrayalSetContextParameter(string *contextParameterName*, variant *value*)

Return Value

void

Parameters

contextParameterName: string

The name of a portrayal context parameter.

value: variant

The new value for the portrayal context parameter. The type of this parameter must match the type specified in the portrayal catalogue.

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.

9A-14.2 Portrayal Domain Specific Host Functions

The host must implement the function described on the following page 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 50.

9A-14.2.1 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 50.

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.