Paper for Consideration by the S-100 Working Group

S-100 Portrayal Support for Lua

Submitted by:	SPAWAR Atlantic
Executive Summary:	This paper describes how S-100 portrayal support for Lua can be implemented, and provides an overview of how Lua support has been implemented in S-100 Viewer 1.3.
Related Documents:	S-100, TSM4-43.3B
Related Projects:	S-100 Test Bed Projects (Simple Viewer, Shore Based ECDIS, ECDIS)

Introduction / Background

At the Rostock meeting, paper TSM4-4.3B ("Proposed Changes to the S-100 Portrayal") suggested how the Lua scripting language could be used in place of XSLT for processing of portrayal rules. Following from that paper, SPAWAR added Lua-based portrayal to the latest version of S-100 Viewer (Version 1.3, available on www.basecamp.com). For additional background, see the above referenced paper.

To support the comparison of XSLT and Lua portrayal catalogues, the S-100 Viewer was modified to allow use of both catalogues simultaneously. The user simply selects the desired catalogue (XSLT or Lua) when a dataset is loaded. The same dataset can be loaded in multiple tabs, each tab using a different portrayal catalogue.

Analysis / Discussion

1. Lua Components Overview

S-100 Portrayal support for Lua is comprised of three primary components:

- Host Integration API
- Lua Portrayal API
- Lua Portrayal Rule File(s)

The "Host Integration API" defines the requirements of the host language to provide proper integration with Lua and the remaining two components listed above. This API is deliberately kept small (around 10 functions) to ensure ease of integration into any host environment. These functions are used by the Lua rule files to gain access to the data managed by the host environment.

The "Lua Portrayal API" defines a logical data access model to ease the writing of portrayal rules. This API is made available via a small number of Lua scripts, delivered as part of the portrayal catalogue. Those scripts, in turn, make use of the low-level "Host Integration API" to convert hosted data into script-friendly Lua objects. The scripts would be made available as appendixes within S-100, so that developers do not need to implement the Portrayal API themselves.

The final component is the set of portrayal rules that are executed for each feature in the dataset. These rule files are organized much like the XSLT templates are: a single top-level Lua script iterates over a list of features and calls the appropriate Lua "rule" file for each feature. Each rule file then makes use of the Portrayal API to access data and emit drawing instructions.

> **NOTE:** The detailed documentation for the Host Integration API and the Lua Portrayal API will be made available separately, initially as part of the S-100 Viewer package. The API documentation should eventually be added to the S-100 specification, pending approval of Lua support in S-100.

2. Comparison Between Lua and XSLT Rule Files

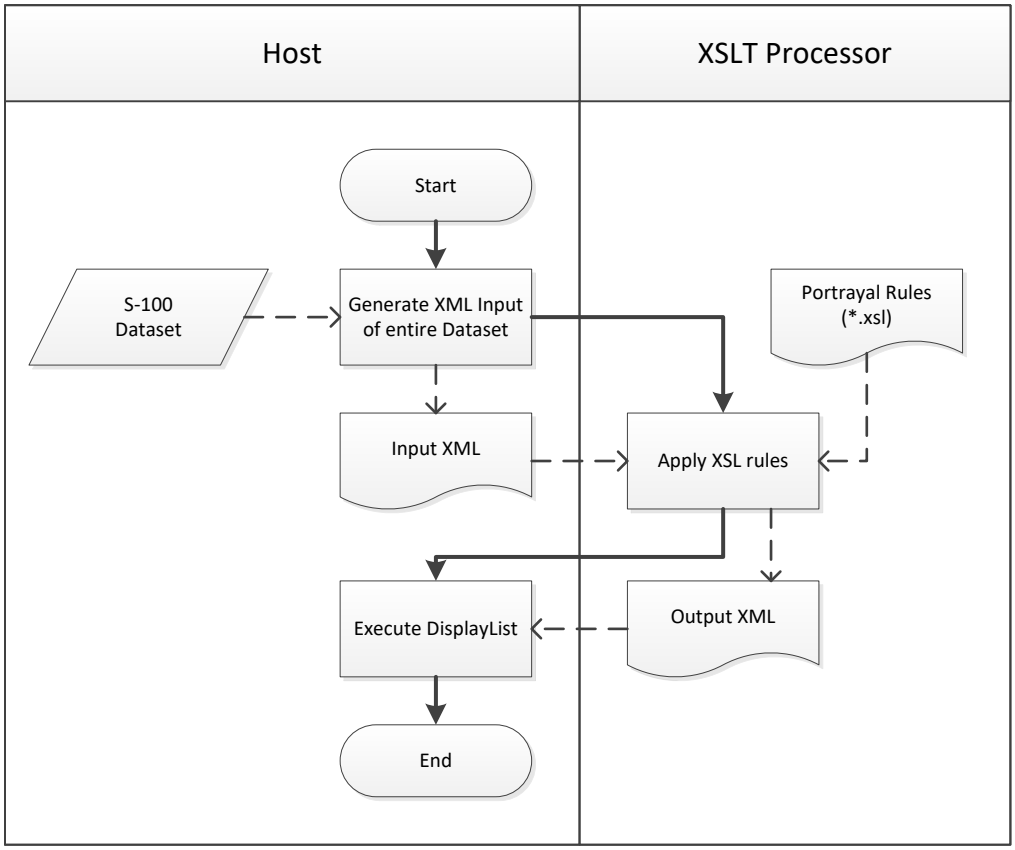
As an example of the difference between a Lua-based rule and an XSLT-based rule, we provide the rule scripts for the AnchorageArea feature type, as shown in Appendix C. In both cases, the scripts have been auto-generated from the S-52 lookup tables, so are longer than they might be if hand-generated. Even so, the scripts accurately illustrate the differences between the two approaches.

This contrast between the two approaches becomes more evident when Conditional Symbology Procedures (CSPs) are involved. In this case, the procedural format of the Lua language is a natural fit for the implementation of CSPs. For examples of this, examine the CSPs in the Lua version of the Portrayal Catalogue (available on www.basecamp.com).

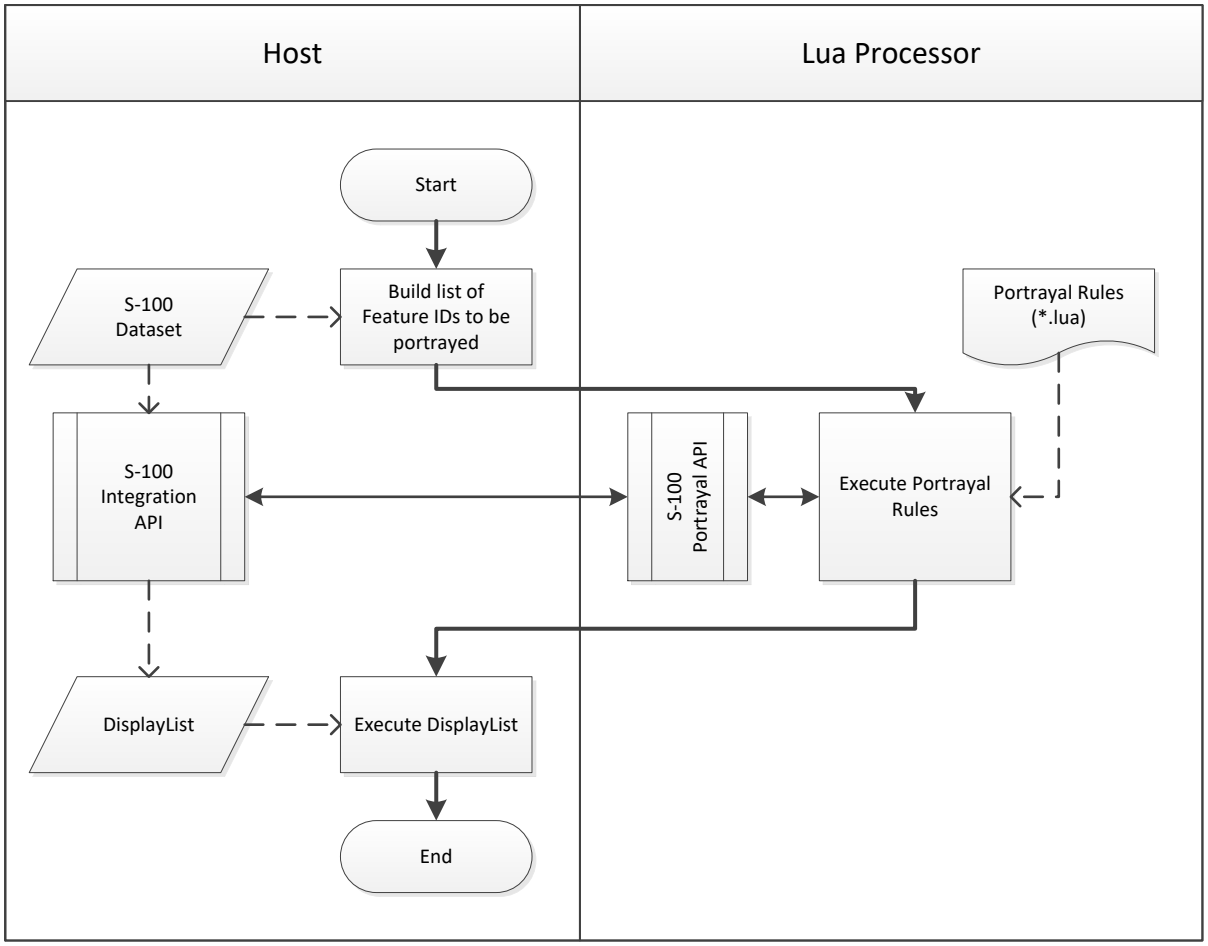
3. Comparison Between Lua and XSLT Rule Processing

The approach to processing the Lua and XSL rules are different.

For XSLT, the host serializes the entire dataset into an input XML document. From there, the input XML file is fed to the XSLT engine to process using the XSLT rules. The XSLT rules query information out of the input XML file and emit draw instructions. The entire set of generated draw instructions is returned to the host as an output XML document. The host then parses the output XML document and executes the draw instructions.



For Lua, the host generates a list of feature IDs. From there, the list is fed to the Lua engine to process using the Lua rules. Each feature ID is used to generate a Lua feature object (according to the Portrayal API), which is then used by the rule scripts to access additional information. The rule script, in turn, emits a set of draw instructions for each feature. These draw instructions are returned to the host one at a time through the Host Integration API. The host gathers up the draw instructions for later execution.



One subtle difference between this and the XSLT version is that, with the Lua approach, *only* the queried information requires serialization from the host to the portrayal environment. Whereas, with XSLT, it is necessary to serialize *all* of the data, because the host cannot know in advance which data the rule scripts are going to need access to. While it may be possible in some host environments to allow the XSLT to access a "virtual input document", thereby allowing similar behaviour to that of Lua, such capabilities are language-dependent. In comparison, the Lua approach works identically on all host environments, regardless of programming language. For a more detailed version of the above flow diagram, see Appendix B.

Another difference is that the XSLT portrayal is a black box which cannot be modified or optimized by the application. The Lua portrayal interacts with the application, and allows for application specific optimizations through the Host Integration API.

4. Current Capabilities and Limitations

The current Lua implementation is on par with the current XSLT implementation. In other words, it is possible to access all of the same data and emit all of the same drawing instructions.

The Lua implementation uses the S-52 CSP rules to implement DEPARE03 instead of the simplified implementation of the XSLT version. This was done to ensure all of the nuances of the CSP are accounted for (in particular the behaviour of safety contours). If it is decided that XSLT shall be used for portrayal, the Lua implementation of the CSP can still be used to provide assurance that the XSLT simplified version is correct.

Date dependant feature portrayal is not implemented in this version of the Lua portrayal. It is also not implemented in the XSLT version.

All items will be available for download from the basecamp S-100 Test Bed website at www.basecamp.com.

Recommendations

1. Replace the use of XSLT within the S-100 portrayal with Lua as described.
2. Update S-100 Part 9, including removal of the portrayal input schema.
3. Add a new part for S-100 Lua Scripting support, supporting development of Alarms and Indications, Interoperability, and other script based extensions to the main functionality described in S-100.

Action Required of the S-100 Working Group

The group are invited to:

a. Note the paper

b. Compare the portrayals generated by the two versions of the S-100 Viewer application

c. Evaluate the provided Lua S-101 portrayal catalogue

d. Discuss and take action on the recommendations

Appendix A: Host Integration API

The host application (ECDIS) must implement the following functions. These functions are called from the Lua rule files (scripts). A sample implementation for each function will be provided.

Host_Dataset_GetFeatures

This function returns the list of features in the dataset.

Host_GetSpatial

This function returns the requested spatial element.

Host_Spatial_GetInformationAssociations

This function returns the information associations for the given spatial.

Host_Feature_GetAttribute

This function returns the requested attribute for the given feature.

Host_Feature_GetInformationAssociations

This function returns the information associations for the given feature.

Host_Feature_GetFeatureAssociations

This function returns the feature associations for the given feature.

Host_Feature_GetSpatialAssociations

This function returns the spatial associations for the given feature.

Host_Information_GetAttribute

This function returns the requested attribute for the given information type.

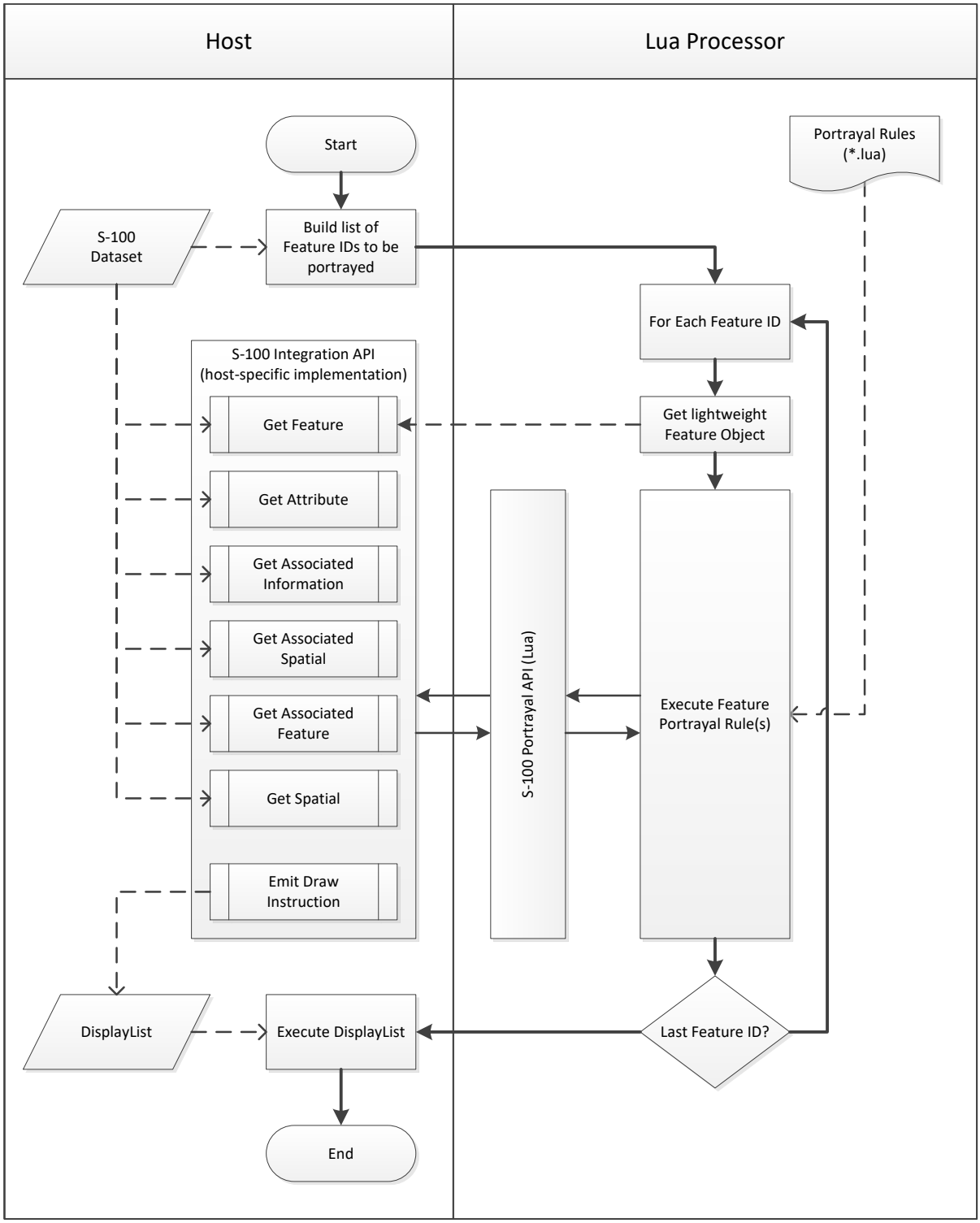
Host_FeaturePortrayal_Emit

This function is called by the rules engine as the drawing instructions are generated for each feature.

Host_DebuggerEntry

Optional. This function allows the host to provide a debugging environment for the rules engine.

Appendix B: Detailed Lua Portrayal Flowchart



APPENDIX C: Sample XSLT and Lua Rules

AnchorageArea_COMMON.xsl

```
<?xml version="1.0" encoding="UTF-8"?>

<xsl:transform xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
   <xsl:output method="xml" encoding="UTF-8" indent="yes"/>
   <xsl:template match="AnchorageArea[@primitive='Point']" priority="1">
      <xsl:if test="featureName!= ''">
         <textInstruction>
            <featureReference>
               <xsl:value-of select="@id"/>
            </featureReference>
            <viewingGroup>26</viewingGroup>
            <displayPlane>OVERRADAR</displayPlane>
            <drawingPriority>18</drawingPriority>
            <textPoint>
               <element>
                  <text>
                     <xsl:apply-templates select="featureName" mode="text"/>
                  </text>
                  <xsl:call-template name="textStyle">
                     <xsl:with-param name="style">default</xsl:with-param>
                  </xsl:call-template>
               </element>
               <offset>
                  <x>-3.51</x>
                  <y>-7.02</y>
               </offset>
            </textPoint>
         </textInstruction>
      </xsl:if>
      <pointInstruction>
         <featureReference>
            <xsl:value-of select="@id"/>
         </featureReference>
         <viewingGroup>26220</viewingGroup>
         <displayPlane>OVERRADAR</displayPlane>
```

```
         <drawingPriority>18</drawingPriority>
         <symbol reference="ACHARE02"/>
      </pointInstruction>
   </xsl:template>
</xsl:transform>
```

AnchorageArea_PLAIN_BOUNDARIES.xsl

```
<?xml version="1.0" encoding="UTF-8"?>

<xsl:transform xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
   <xsl:output method="xml" encoding="UTF-8" indent="yes"/>
   <xsl:template match="AnchorageArea[@primitive='Surface']" priority="1">
      <pointInstruction>
         <featureReference>
            <xsl:value-of select="@id"/>
         </featureReference>
         <viewingGroup>26220</viewingGroup>
         <displayPlane>UNDERRADAR</displayPlane>
         <drawingPriority>9</drawingPriority>
         <symbol reference="ACHARE51">
            <areaPlacement placementMode="VisibleParts"/>
         </symbol>
      </pointInstruction>
      <xsl:if test="featureName!= ''">
         <textInstruction>
            <featureReference>
               <xsl:value-of select="@id"/>
            </featureReference>
            <viewingGroup>26</viewingGroup>
            <displayPlane>UNDERRADAR</displayPlane>
            <drawingPriority>9</drawingPriority>
            <textPoint horizontalAlignment="End">
               <element>
                  <text>
                     <xsl:apply-templates select="featureName" mode="text"/>
                  </text>
                  <xsl:call-template name="textStyle">
```

```
                  <xsl:with-param name="style">default</xsl:with-param>
               </xsl:call-template>
            </element>
            <offset>
               <x>-3.51</x>
               <y>-7.02</y>
            </offset>
            <areaPlacement placementMode="VisibleParts"/>
         </textPoint>
      </textInstruction>
   </xsl:if>
   <lineInstruction>
      <featureReference>
         <xsl:value-of select="@id"/>
      </featureReference>
      <viewingGroup>26220</viewingGroup>
      <displayPlane>UNDERRADAR</displayPlane>
      <drawingPriority>9</drawingPriority>
      <xsl:call-template name="simpleLineStyle">
         <xsl:with-param name="style">dash</xsl:with-param>
         <xsl:with-param name="width">0.64</xsl:with-param>
         <xsl:with-param name="colour">CHMGF</xsl:with-param>
      </xsl:call-template>
   </lineInstruction>
   <xsl:call-template name="RESCSP03">
      <xsl:with-param name="viewingGroup">26220</xsl:with-param>
      <xsl:with-param name="displayPlane">UNDERRADAR</xsl:with-param>
      <xsl:with-param name="drawingPriority">9</xsl:with-param>
   </xsl:call-template>
</xsl:template>
<xsl:template match="AnchorageArea[@primitive='Surface' and categoryOfAnchorage=8]" priority="2">
   <pointInstruction>
      <featureReference>
         <xsl:value-of select="@id"/>
      </featureReference>
      <viewingGroup>26220</viewingGroup>
      <displayPlane>UNDERRADAR</displayPlane>
      <drawingPriority>9</drawingPriority>
```

```
      <symbol reference="ACHARE02">
         <areaPlacement placementMode="VisibleParts"/>
      </symbol>
   </pointInstruction>
   <xsl:if test="featureName!= ''">
      <textInstruction>
         <featureReference>
            <xsl:value-of select="@id"/>
         </featureReference>
         <viewingGroup>26</viewingGroup>
         <displayPlane>UNDERRADAR</displayPlane>
         <drawingPriority>9</drawingPriority>
         <textPoint horizontalAlignment="End">
            <element>
               <text>
                  <xsl:apply-templates select="featureName" mode="text"/>
               </text>
               <xsl:call-template name="textStyle">
                  <xsl:with-param name="style">default</xsl:with-param>
               </xsl:call-template>
            </element>
            <offset>
               <x>-3.51</x>
               <y>-7.02</y>
            </offset>
            <areaPlacement placementMode="VisibleParts"/>
         </textPoint>
      </textInstruction>
   </xsl:if>
   <lineInstruction>
      <featureReference>
         <xsl:value-of select="@id"/>
      </featureReference>
      <viewingGroup>26220</viewingGroup>
      <displayPlane>UNDERRADAR</displayPlane>
      <drawingPriority>9</drawingPriority>
      <xsl:call-template name="simpleLineStyle">
         <xsl:with-param name="style">dash</xsl:with-param>
```

```
            <xsl:with-param name="width">0.64</xsl:with-param>
            <xsl:with-param name="colour">CHMGF</xsl:with-param>
         </xsl:call-template>
      </lineInstruction>
      <xsl:call-template name="RESCSP03">
         <xsl:with-param name="viewingGroup">26220</xsl:with-param>
         <xsl:with-param name="displayPlane">UNDERRADAR</xsl:with-param>
         <xsl:with-param name="drawingPriority">9</xsl:with-param>
      </xsl:call-template>
   </xsl:template>
</xsl:transform>
```

AnchorageArea_SYMBOLIZED_BOUNDARIES.xsl

```
<?xml version="1.0" encoding="UTF-8"?>

<xsl:transform xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
   <xsl:output method="xml" encoding="UTF-8" indent="yes"/>
   <xsl:template match="AnchorageArea[@primitive='Surface']" priority="1">
      <pointInstruction>
         <featureReference>
            <xsl:value-of select="@id"/>
         </featureReference>
         <viewingGroup>26220</viewingGroup>
         <displayPlane>UNDERRADAR</displayPlane>
         <drawingPriority>9</drawingPriority>
         <symbol reference="ACHARE51">
            <areaPlacement placementMode="VisibleParts"/>
         </symbol>
      </pointInstruction>
      <xsl:if test="featureName!= ''">
         <textInstruction>
            <featureReference>
               <xsl:value-of select="@id"/>
            </featureReference>
            <viewingGroup>26</viewingGroup>
            <displayPlane>UNDERRADAR</displayPlane>
            <drawingPriority>9</drawingPriority>
```

```
         <textPoint horizontalAlignment="End">
            <element>
               <text>
                  <xsl:apply-templates select="featureName" mode="text"/>
               </text>
               <xsl:call-template name="textStyle">
                  <xsl:with-param name="style">default</xsl:with-param>
               </xsl:call-template>
            </element>
            <offset>
               <x>-3.51</x>
               <y>-7.02</y>
            </offset>
            <areaPlacement placementMode="VisibleParts"/>
         </textPoint>
      </textInstruction>
   </xsl:if>
   <lineInstruction>
      <featureReference>
         <xsl:value-of select="@id"/>
      </featureReference>
      <viewingGroup>26220</viewingGroup>
      <displayPlane>UNDERRADAR</displayPlane>
      <drawingPriority>9</drawingPriority>
      <lineStyleReference reference="ACHARE51"/>
   </lineInstruction>
   <xsl:call-template name="RESCSP03">
      <xsl:with-param name="viewingGroup">26220</xsl:with-param>
      <xsl:with-param name="displayPlane">UNDERRADAR</xsl:with-param>
      <xsl:with-param name="drawingPriority">9</xsl:with-param>
   </xsl:call-template>
</xsl:template>
<xsl:template match="AnchorageArea[@primitive='Surface' and categoryOfAnchorage=8]" priority="2">
   <pointInstruction>
      <featureReference>
         <xsl:value-of select="@id"/>
      </featureReference>
      <viewingGroup>26220</viewingGroup>
```

```
      <displayPlane>UNDERRADAR</displayPlane>
      <drawingPriority>9</drawingPriority>
      <symbol reference="ACHARE02">
         <areaPlacement placementMode="VisibleParts"/>
      </symbol>
   </pointInstruction>
   <xsl:if test="featureName!= ''">
      <textInstruction>
         <featureReference>
            <xsl:value-of select="@id"/>
         </featureReference>
         <viewingGroup>26</viewingGroup>
         <displayPlane>UNDERRADAR</displayPlane>
         <drawingPriority>9</drawingPriority>
         <textPoint horizontalAlignment="End">
            <element>
               <text>
                  <xsl:apply-templates select="featureName" mode="text"/>
               </text>
               <xsl:call-template name="textStyle">
                  <xsl:with-param name="style">default</xsl:with-param>
               </xsl:call-template>
            </element>
            <offset>
               <x>-3.51</x>
               <y>-7.02</y>
            </offset>
            <areaPlacement placementMode="VisibleParts"/>
         </textPoint>
      </textInstruction>
   </xsl:if>
   <lineInstruction>
      <featureReference>
         <xsl:value-of select="@id"/>
      </featureReference>
      <viewingGroup>26220</viewingGroup>
      <displayPlane>UNDERRADAR</displayPlane>
      <drawingPriority>9</drawingPriority>
```

```
         <lineStyleReference reference="ACHARE51"/>
      </lineInstruction>
      <xsl:call-template name="RESCSP03">
         <xsl:with-param name="viewingGroup">26220</xsl:with-param>
         <xsl:with-param name="displayPlane">UNDERRADAR</xsl:with-param>
         <xsl:with-param name="drawingPriority">9</xsl:with-param>
      </xsl:call-template>
   </xsl:template>
</xsl:transform>
```

AnchorageArea.lua

```lua
-- Referenced portrayal rules.
require 'RESTRN01'

function AnchorageArea(portrayalContext, featurePortrayal)
    local feature = featurePortrayal.Feature
    local drawingInstructions = featurePortrayal.DrawingInstructions

    if feature.PrimitiveType == PrimitiveType.Point then
        -- Simplified and paper chart points use the same symbolization
        featurePortrayal:SetDisplayParameters(26220, 6, nil, nil, 'OverRADAR')
        drawingInstructions:AddPointInstruction('ACHARE02')
        if feature.featureName and feature.featureName.name then
            drawingInstructions:AddTextInstruction(
                Text.CreateTextPoint(
                    {Text.CreateTextElement(feature.featureName.name, defaultFontCharacteristics, 10, 'CHBLK')},
                    Graphics.CreateVector(-3.51, -7.02)),
                nil,
                PortrayalModel.CreateDisplayParameters(26))
        end
    elseif feature.PrimitiveType == PrimitiveType.Surface and portrayalContext.ContextParameters.PLAIN_BOUNDARIES then
        if feature.categoryOfAnchorage == 8 then
            featurePortrayal:SetDisplayParameters(26220, 3, nil, nil, 'UnderRADAR')
            drawingInstructions:AddPointInstruction('ACHARE02')
            if feature.featureName and feature.featureName.name then
                drawingInstructions:AddTextInstruction(
                    Text.CreateTextPoint(
                        {Text.CreateTextElement(feature.featureName.name, defaultFontCharacteristics, 10, 'CHBLK')},
                        Graphics.CreateVector(-3.51, 7.02),
                        nil,
                        nil,
                        Text.HorizontalAlignment.End, Text.VerticalAlignment.Bottom),
                    nil,
                    PortrayalModel.CreateDisplayParameters(26))
            end
            drawingInstructions:AddLineInstruction(LineStyles.CreateStandardLineStyleDash(0.64, 'CHMGF'))
            RESTRN01(portrayalContext, featurePortrayal)
```

```
    else
        featurePortrayal:SetDisplayParameters(26220, 3, nil, nil, 'UnderRADAR')
        drawingInstructions:AddPointInstruction('ACHARE51')
        if feature.featureName and feature.featureName.name then
            drawingInstructions:AddTextInstruction(
                Text.CreateTextPoint(
                    {Text.CreateTextElement(feature.featureName.name, defaultFontCharacteristics, 10, 'CHBLK')},
                    Graphics.CreateVector(-3.51, -7.02),
                    nil,
                    nil,
                    Text.HorizontalAlignment.End,
                    Text.VerticalAlignment.Bottom),
                nil,
                PortrayalModel.CreateDisplayParameters(26))
        end
        drawingInstructions:AddLineInstruction(LineStyles.CreateStandardLineStyleDash(0.64, 'CHMGF'))
        RESTRN01(portrayalContext, featurePortrayal)
    end
elseif feature.PrimitiveType == PrimitiveType.Surface then
    if feature.categoryOfAnchorage == 8 then
        featurePortrayal:SetDisplayParameters(26220, 3, nil, nil, 'UnderRADAR')
        drawingInstructions:AddPointInstruction('ACHARE02')
        if feature.featureName and feature.featureName.name then
            drawingInstructions:AddTextInstruction(
                Text.CreateTextPoint(
                    {Text.CreateTextElement(feature.featureName.name, defaultFontCharacteristics, 10, 'CHBLK')},
                    Graphics.CreateVector(-3.51, 7.02)),
                nil,
                PortrayalModel.CreateDisplayParameters(26))
        end
        drawingInstructions:AddLineInstruction(LineStyles.CreateStandardLineStyleDash(0.64, 'CHMGF'))
        RESTRN01(portrayalContext, featurePortrayal)
    else
        featurePortrayal:SetDisplayParameters(26220, 3, nil, nil, 'UnderRADAR')
        drawingInstructions:AddPointInstruction('ACHARE51')
        if feature.featureName and feature.featureName.name then
            drawingInstructions:AddTextInstruction(
                Text.CreateTextPoint(
```

```
                    {Text.CreateTextElement(feature.featureName.name, defaultFontCharacteristics, 10, 'CHBLK')},
                    Graphics.CreateVector(-3.51, 7.02)),
                nil,
                PortrayalModel.CreateDisplayParameters(26))
        end
        drawingInstructions:AddLineInstruction('ACHARE51')
        RESTRN01(portrayalContext, featurePortrayal)
    end
else
    error('Invalid primitive type or mariner settings passed to portrayal')
end
end
```