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 Lua scripting initialization 4
9A-5.2.2 Portrayal Context Parameters ..... **Error! Bookmark not defined.**
9A-5.2.3 Generating a Portrayal 5
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 5
9A-12 Symbol Definitions 5
9A-13 The portrayal library 5
9A-13.1 FileFormat 5
9A-14 Lua Portrayal Functions 6
9A-14.1 Portrayal Script Functions 6
9A-14.2 Portrayal Host Functions 11

9A-1 Scope

This part of the standard defines the additions and changes to S-100 Part 9 necessary to implement portrayal using the Lua scripting mechanism defined in S-100 Part 50. Products which specify use of a Lua portrayal catalogue 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 5.3 Reference Manual, https://www.lua.org/manual/5.3/

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 1 illustrates the general portrayal model.

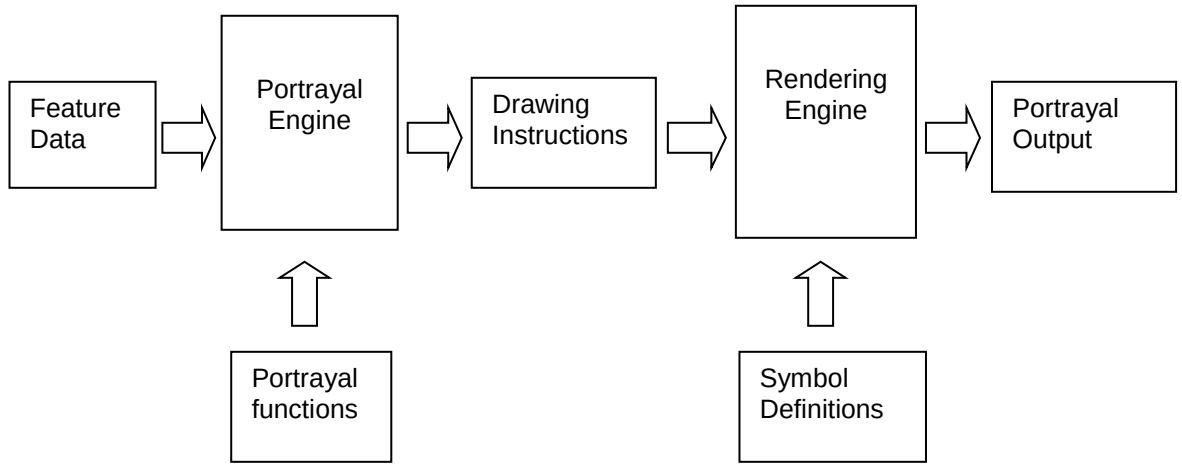

Figure 1 - General portrayal model

9A-5.1 The Portrayal Process

A Lua portrayal requires the following changes to the portrayal process described in 9-5.1:

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 callback functions.
Drawing instructions are returned to the host as an XML document, which is the result of the transformation applied to the input feature data.	Drawing instructions are returned to the host via host callback function *HostPortrayalEmit*

Table 1 - Changes to the portrayal process

These changes are illustrated in Figure 2:

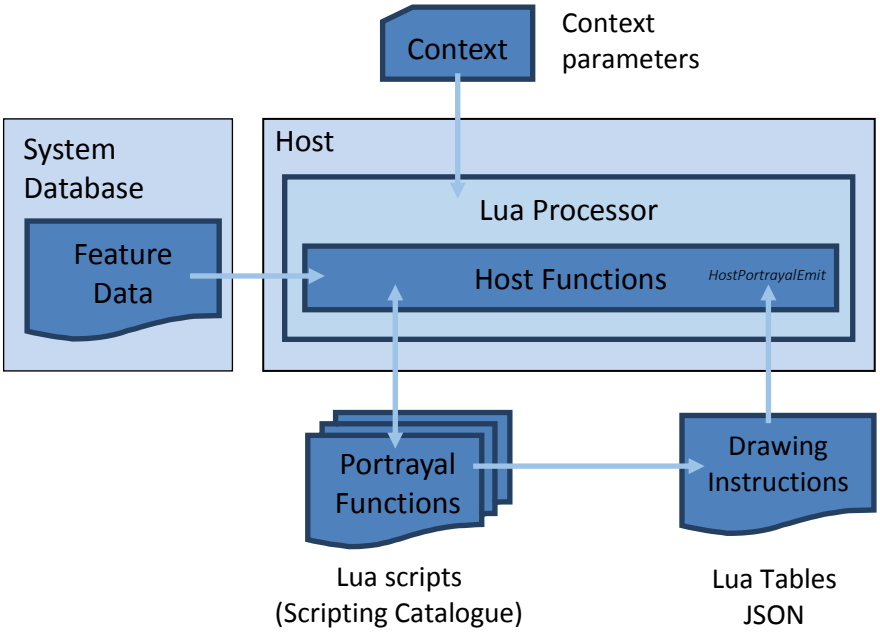

Figure 2 - Portrayal process

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 data structures and functions which enable the portrayal process are detailed in 9A-14.

The Lua portrayal process is shown in Figure 3.

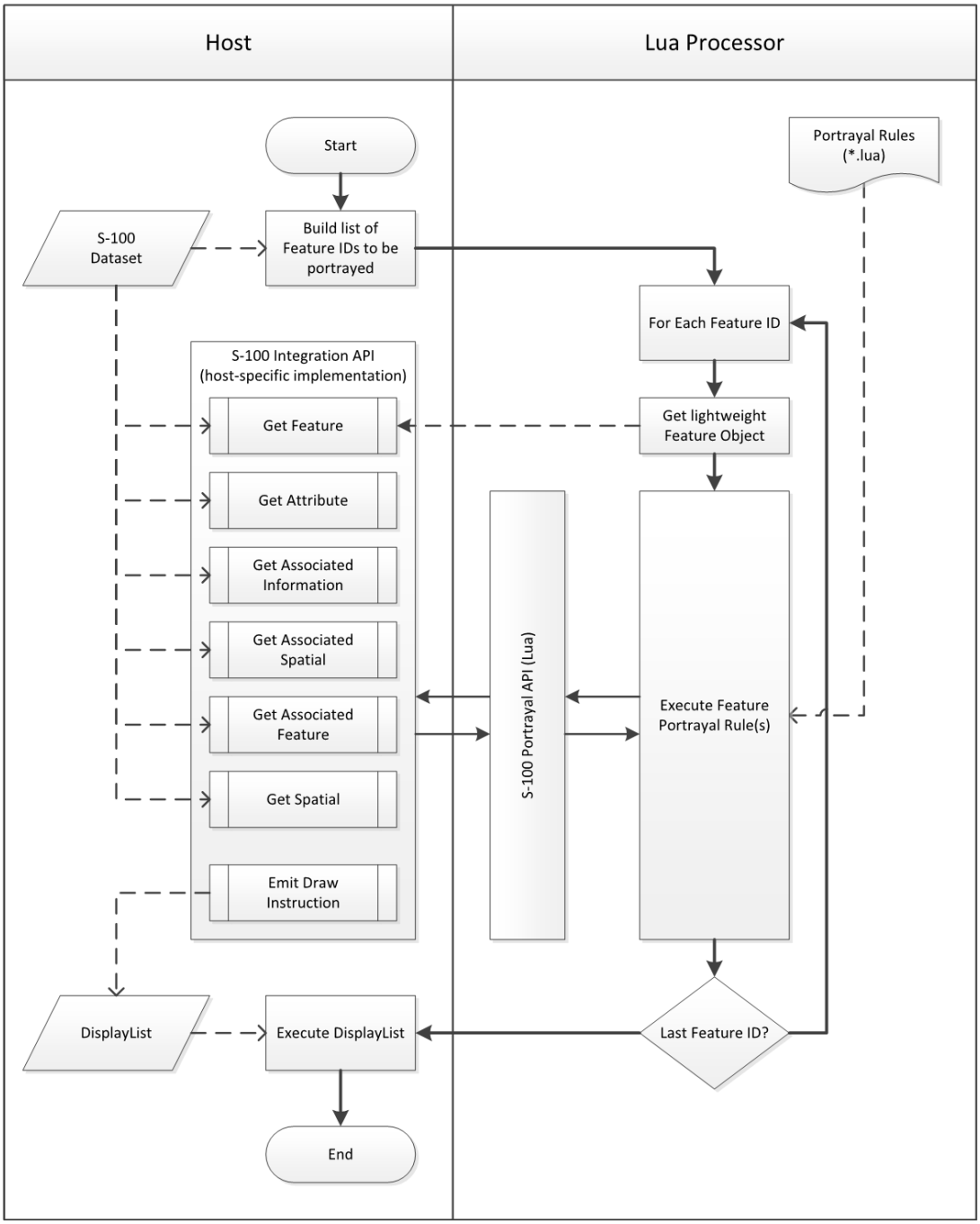

Figure 3. Lua Portrayal Process

9A-5.2.1 Portrayal Initialization

Prior to calling any Lua portrayal functions, the host must register the domain specific scripting catalogue functions by loading a Lua script file defined as a TopLevelTemplate. 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

To generate drawing instructions the host calls *PortrayalMain*, which starts the processing for a dataset. The portrayal will use the standard host functions described in Part 50 to obtain a list describing the feature instances within the dataset, and subsequeuently will iterate over each feature instance to generate drawing instructions.

As each feature instance is processed, the portrayal engine will call standard host functions to request attribute, spatial, or other information as needed. As processing for each feature instance is completed, the portrayal engine will call *HostPortrayalEmit* and provide the drawing instructions for the feature instance to the host application.

The portrayal for a given 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*.

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. Data is passed between a Part 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

There is no change to the drawing instructions described in 9-11.

9A-12 Symbol Definitions

There is no change to the symbol definitions described in 9-12.

9A-13 The portrayal library

There is no change to the organization of the portrayal library. The contents of the “Rules” folder, which 9-13.2 indicates contains XSLT templates stored in separate files, instead contains 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 conjuction 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.

boolean **PortrayalMain**(string *datasetID*, boolean *enableDrawInstructionCaching*)

Return Value

true

Portrayal completed successfully

false

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

Parameters

datasetID: string

Used by the host to uniquely identify a dataset.

enableDrawInstructionCaching: boolean

Controls whether the portrayal will generate drawing instructions for all feature instances in the dataset.

If set to true, only feature instances whose drawing instructions have changed since the most recent call to *PortrayalMain* will trigger calls to *HostPortrayalEmit*.

If set to false, a call to *HostPortrayalEmit* will be made for every feature instance 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 in the dataset.

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

void **PortrayalInitializeContextParameters**(string *datasetID*, ContextParameter[] *contextParameters*)

Return Value

void

Parameters

datasetID: string

Used by the host to uniquely identify a dataset.

contextParameters: ContextParameter[]

An array of ContextParameter objects. *PortrayalCreateContextParameter* should be used to create each entry.

Remarks

Provides the portrayal scripts with the default value for each portrayal context parameter defined within the portrayal catalogue. The host is responsible for retrieving the portrayal context parameters from the portrayal catalogue.

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.

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 functions described on the following pages in order to support portrayal. These functions are called from the portrayal domain specific catalogue functions, and augment the standard host functions described in Part 50.

boolean **HostPortrayalEmit**(string *featureID*, DrawingInstruction[] *drawingInstructions*)

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: DrawingInstruction[]

All of the drawing instructions generated for the feature instance identified by *featureID*.

Remarks

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

If the host finds it easier to parse the *drawingInstructions* in JSON format, the standard catalogue function *ConvertToJSON* can be used to convert from the Lua table representation to a JSON representation.

If *PortrayalMain* was called with *enableDrawInstructionCaching* enabled, it is the hosts responsibility to cache the returned drawing instructions; *HostPortrayalEmit* will only be called for feature instances whose drawing instructions have changed since a previous call to *PortrayalMain*.

Alternatively, the host may force re-generation of all drawing instructions by re-instantiating the Lua runtime, or by setting parameter *enableDrawInstructionCaching* to false when calling *PortrayalMain*.