OGC Testbed-13

Portrayal Engineering Report

Table of Contents

Publication Date: 2018-03-05

Approval Date: 2018-03-02

Posted Date: 2018-01-31

Reference number of this document: OGC 17-045

Reference URL for this document: http://www.opengis.net/doc/PER/t13-NG008

Category: Public Engineering Report

Editor: Stephane Fellah

Title: OGC Testbed-13: Portrayal Engineering Report

indicate compliance with any LICENSOR standards or specifications.

This Agreement is governed by the laws of the Commonwealth of Massachusetts. The application to this Agreement of the United Nations Convention on Contracts for the International Sale of Goods is hereby expressly excluded. In the event any provision of this Agreement shall be deemed unenforceable, void or invalid, such provision shall be modified so as to make it valid and enforceable, and as so modified the entire Agreement shall remain in full force and effect. No decision, action or inaction by LICENSOR shall be construed to be a waiver of any rights or remedies available to it.

None of the Intellectual Property or underlying information or technology may be downloaded or otherwise exported or reexported in violation of U.S. export laws and regulations. In addition, you are responsible for complying with any local laws in your jurisdiction which may impact your right to import, export or use the Intellectual Property, and you represent that you have complied with any regulations or registration procedures required by applicable law to make this license enforceable.

Chapter 1. Summary

Portrayal of geospatial information plays a crucial role in situation awareness, analysis and decision-making. Visualizing geospatial information often requires one to portray the information using symbology or cartographic presentation rules from a community or organization. For example, among those in the law enforcement, fire and rescue community, various local, national and international agencies use different symbols and terminology for the same event, location and building, employing syntactic, structural-based and document-centric data models (e.g., eXtensible Markup Language (XML) schemas and Style Layer Descriptors (SLD)). With this approach, interoperability does not extend to the semantic level, which makes it difficult to share, reuse and mediate unambiguous portrayal information between agencies.

This Engineering Report (ER) captures the requirements, solutions, models and implementations of the Testbed 13 Portrayal Package. This effort leverages the work on Portrayal Ontology development and Semantic Portrayal Service conducted during Testbed 10, 11 and 12. The objective of this Testbed 13 is to identify and complete the gaps in the latest version of the portrayal ontology defined in Testbed 12, complete the implementation of the Semantic Portrayal Service by adding rendering capabilities and performing a demonstration of the portrayal service that showcases the benefits of the proposed semantic-based approach.

1.1. Requirements

The Testbed 12 initiative defined and implemented a set of portrayal ontologies and a RESTful API for a Semantic Portrayal Service. Due to time limitations, the API implementation didn't address the rendering aspect of the service and didn't fully test the round-trip conversion of Style Layer Descriptor (SLD) documents with the portrayal ontologies. The work presented in this Engineering Report addresses the following requirements:

- Identify gaps with SLD and prior microtheories developed during Testbed 12
- Define the renderer Application Programming Interface (API) and output in image formats Portable Network Graphic (PNG).
- Define workflow of the demonstration scenario

1.2. Key Findings and Prior-After Comparison

OGC has not previously explored an approach for representing portrayal information using semantic-based technologies. Current OGC standards such Style Layer Description (SLD) and Symbol Encoding (SE) are document-centric and assume that the data models are based on Geography Markup Language (GML) encodings making it hard to share and reuse portrayal information that is based on other data models.

1.3. What does this ER mean for the Working Group and OGC in general

This Engineering Report (ER) is relevant to the GeoSemantics Domain Working Group (DWG) and a possible future Portrayal DWG. Both working groups should share the objectives of enabling semantic interoperability of geospatial-related information. The portrayal ontologies produced by this testbed define a reusable, extensible, machine-tractable model that allow the sharing of layer and portrayal information. The portrayal ontology provides an extensible framework to represent style for layers working on different data models and formats such as Extensible Markup Language (XML), JavaScript Object Notation (JSON), and others based on Linked Data. The Semantic Portrayal API showcases how semantic information can be shared using JSON for Linked Data (JSON-LD) and a hypermedia Representational State Transfer (REST) API combining semantic information and hypermedia controls.

1.4. Document contributor contact points

All questions regarding this document should be directed to the editor or the contributors:

Table 1. Contacts

Name	Organization
Stephane Fellah	Image Matters LLC
Emily Mitchell	Image Matters LLC
Simone Giannecchini	Geo-Solutions

1.5. Future Work

1.5.1. Map and Layer Profile

Layers and Maps of geospatial data are very commonplace, but there is no consistent and standard way to describe their metadata. While Layer and Map entities are derived from a Dataset entity, they have their own specific metadata. We propose for the next testbed to investigate a profile for Layer and Map concepts that extends the Registry Item and relates to Datasets, Services and Portrayal Information developed for the Semantic Registry and Semantic Portrayal Service.

1.5.2. Coverage Portrayal

So far, the emphasis on developing the portrayal ontologies has been on modeling and representing portrayal information for feature data. The proposal is that the next testbed focuses on addressing portrayal information for coverage data (in particular grid coverage). This will close the gap of expressiveness of the portrayal ontology with SLD and SE standards.

1.5.3. Composite Symbology Semantic Portrayal Service.

For the next Testbed, we propose to extend the ontology to accommodate more complex symbols such as composite symbols and symbol templates to describe more advanced symbology standards such as the family of MIL-STD-2525D symbols. It is proposed to also extend the portrayal ontology to represent composite symbols and symbol templates. Investigation should also include other renderer outputs such as JSON encoding of the portrayal information, so they can be handled on the client side in HTML5 Canvas or other rendering libraries such as D3.js.

1.6. Foreword

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. The Open Geospatial Consortium shall not be held responsible for identifying any or all such patent rights.

Recipients of this document are requested to submit, with their comments, notification of any relevant patent claims or other intellectual property rights of which they may be aware that might be infringed by any implementation of the standard set forth in this document, and to provide supporting documentation.

Chapter 2. References

The following documents are referenced in this document. For dated references, subsequent amendments to, or revisions of, any of these publications do not apply. For undated references, the latest edition of the normative document referred to applies.

- OGC 16-062 - OGC® Testbed-12 Catalogue and SPARQL Engineering Report
- OGC 15-058 - OGC® Testbed-11 Symbology Mediation Engineering Report
- OGC 15-054 - OGC® Testbed-11 Implementing Linked Data and Semantically Enabling OGC Services Engineering Report
- OGC 13-084r2, OGC I15 (ISO19115 Metadata) Extension Package of CS-W ebRIM Profile .0, 2014
- OGC 12-168r6, OGC® Catalogue Services 3.0 - General Model, 2016
- OGC 11-052r4, OGC GeoSPARQL- A Geographic Query Language for RDF Data, 2011
- OGC 09-026r2, OGC Filter Encoding 2.0 Encoding Standard - With Corrigendum
- OGC 08-125r1, KML Standard Development Practices, Version 0.6, 2009.
- OGC 07-147r2, KML Version 2.2.0.2008
- OGC 07-110r4, CSW-ebRIM Registry Service ebRIM profile of CSW (.0.1), 2009
- OGC 07-045, OGC Catalogue Services Specification 2.0.2 - ISO Metadata Application Profile (.0.0), 2007
- OGC 07-006r1, OpenGIS Catalogue Service Implementation Specification 2.0.2, 2007
- OGC 06-129r1, FGDC CSDGM Application Profile for CSW 2.0 (0.0.12), 2006
- OGC 06-121r9, OGC® Web Services Common Standard
- OGC 06-121r3, OpenGIS® Web Services Common Specification, version 1.1.0 with Corrigendum 1 2006
- OGC 05-078r4, OpenGIS Styled Layer Descriptor Profile of the Web Map Service Implementation Specification, Version 1.1.0, 2006
- OGC 05-077r4, OpenGIS® Symbology Encoding Implementation Specification, Version 1.1.0, 2006.
- ISO/TS 19139:2007, Geographic information — Metadata — XML schema implementation
- ISO 19119:2005, Geographic information — Services
- ISO 19117:2012, Geographic information — Portrayal
- ISO 19115:2003, Geographic information — Metadata
- ISO 19115:2003/Cor 1:2006, Geographic information — Metadata
- ISO 19115-1:2014, Geographic information — Metadata — Part 1: Fundamentals
- Dublin Core Metadata Initiative, last visited 12-09-2016, available from http://dublincore.org/
- NSG Metadata Foundation (NMF) – Part 1: Core, version 2.2, 23 September 2014 https://nsgreg.nga.mil/doc/view?i=4123
- DGIWG 114, DGIWG Metadata Foundation (DMF),last visited 12-09-2016, available from

https://portal.dgiwg.org/files/?artifact_id=9189&format=pdf

- DoD Discovery Metadata Specification (DDMS),last visited 12-09-2016, available from https://metadata.ces.mil/dse-help/DDMS/index.htm
- SPARQL Protocol and RDF Query Language (SPARQL),last visited 12-09-2016, available from https://www.w3.org/TR/rdf-sparql-query
- DCAT, last visited 12-09-2016, available from https://www.w3.org/TR/vocab-dcat/
- National System for Geospatial Intelligence Metadata Implementation Specification (NMIS) – Part 2: XML Exchange Schema
- Project Open Data Metadata Schema v1.1 https://project-open-data.cio.gov/v1.1/schema/
- Asset Description Metadata Schema (ADMS) https://www.w3.org/TR/vocab-adms/
- JSON-LD 1.0 https://www.w3.org/TR/json-ld/
- OWL-S: Semantic Markup for Web Services https://www.w3.org/Submission/OWL-S/

Chapter 3. Terms and definitions

For the purposes of this report, the definitions specified in Clause 4 of the OWS Common Implementation Standard [OGC 06-121r9] shall apply. In addition, the following terms and definitions apply.

3.1. feature

```
representation of some real world object or phenomenon
```

3.2. interoperability

```
capability to communicate, execute programs, or transfer data among various functional
units in a manner that requires the user to have little or no knowledge of the unique
characteristics of those units [ISO 19119]
```

3.3. layer

```
basic unit of geographic information that may be requested as a map from a server
```

3.4. linked data

```
a pattern for hyperlinking machine-readable data sets to each other using Semantic Web
techniques, especially via the use of RDF and URIs. Enables distributed SPARQL queries
of the data sets and a browsing or discovery approach to finding information (as
compared to a search strategy). Linked Data is intended for access by both humans and
machines. Linked Data uses the RDF family of standards for data interchange (e.g.,
RDF/XML, RDFa, Turtle) and query (SPARQL). If Linked Data is published on the public
Web, it is generally called Linked Open Data.
```

3.5. map

```
pictorial representation of geographic data
```

3.6. model

```
abstraction of some aspects of a universe of discourse [ISO 19109]
```

3.7. ontology

```
a formal specification of concrete or abstract things, and the relationships among
them, in a prescribed domain of knowledge [ISO/IEC 19763]
```

3.8. portrayal

```
portrayal presentation of information to humans [ISO 19117]
```

3.9. semantic interoperability

```
the aspect of interoperability that assures that the content is understood in the same
way in both systems, including by those humans interacting with the systems in a given
context
```

3.10. semantic mediation

```
transformation from one or more datasets into a dataset based on a different
conceptual  model.
```

3.11. symbol

```
a bitmap or vector image that is used to indicate an object or a particular property
on a map.
```

3.12. symbology encoding

```
style description to apply to the digital features being rendered
```

3.13. syntactic interoperability

```
the aspect of interoperability that assures that there is a technical connection, i.e.
that the data can be transferred between systems
```

Chapter 4. Conventions

4.1. Abbreviated terms

- API Application Programming Interface
- CRS Coordinate Reference System
- CSV Comma Separated Values
- CSW Catalog Services for the Web
- DCAT Data Catalog Vocabulary
- DCAT-AP DCAT Application Profile for Data Portals in Europe
- DCMI Dublin Core Metadata Initiative
- EARL Evaluation and Report Language EU European Union
- EuroVoc Multilingual Thesaurus of the European Union
- GEMET GEneral Multilingual Environmental Thesaurus
- GML Geography Markup Language
- GeoDCAT-AP Geographical extension of DCAT-AP
- GeoJSON Geospatial JavaScript Object Notation
- IANA Internet Assigned Numbers Authority
- INSPIRE Infrastructure for Spatial Information in the European Community
- ISO International Organization for Standardization
- JSON JavaScript Object Notation
- JRC European Commission - Joint Research Centre MDR Metadata Registry
- N3 Notation 3 format
- NAL Named Authority Lists
- OGC Open Geospatial Consortium
- OWL Web Ontology Language
- RDF Resource Description Framework
- RFC Request for Comments
- SE Symbology Encoding
- SLD Style Layer Descriptor
- SKOS Simple Knowledge Organization System
- SPARQL SPARQL Protocol and RDF Query URI Uniform Resource Identifier
- SVG Scalable Vector Graphics
- TTL Turtle Format
- URI Uniform Resource Identifier

- URL Uniform Resource Locator
- URN Uniform Resource Name
- W3C World Wide Web Consortium
- WG Working Group
- WKT Well Known Text
- XML eXtensible Markup Language
- XSLT eXtensible Stylesheet Language Transformations

Chapter 5. Overview

This ER is broken down into three sections. The first section is related to the Portrayal ontology modeling. It documents the changes needed to better align with SLD and SE standards and lessons learned from the implementations. The second section is related to the Semantic Portrayal Service Application Protocol Interface (API). It documents the changes related to the API, in particular the endpoints related to the rendering of geospatial data and legends. The last section focuses on documenting the workflow of the Portrayal Demonstration, challenges and issues found during the Technology Integration Experiments (TIE). The ER also has two appendices: the first documents the portrayal ontology, and the second documents the Representational State Transfer (REST) API of the Semantic Portrayal Service.

Chapter 6. Portrayal Ontologies

This section summarizes the findings, design approaches and changes in the portrayal ontologies.

6.1. Background

The formalization of portrayal ontologies started in OGC Testbed 10, where the focus was on representing point-based symbologies related to Disaster and Emergency Management.

An Incident Ontology and Taxonomy for Natural Events and Emergency Incidents was developed and used to represent incidents that could be represented in the Homeland Security Working Group Symbology (HSWG) for incidents.

The initial implementation of the Semantic Portrayal Service during the OGC Testbed 11 focused on defining the styles, portrayal rules, point-based symbols and graphics to enable a Web Processing Service (WPS) to produce an SLD document. The initial ontology was heavily based on ISO 19117 Geographic Information-Portrayal standard.

However, during the implementation of style renderers and development of the graphic ontology during Testbed 12, it was concluded the ISO 19117 was mostly designed for runtime implementation (for example use of portrayal function) rather than adapted for a declarative approach.

It was found that the OGC SE standard provides a declarative approach based on XML encoding that is better aligned with modern renderer API approaches such as Java Canvas, Hypertext Markup Language (HTML) Canvas, Scalable Vector Graphics (SVG), MapCSS, ESRI Map Renderer, etc. An update of the portrayal ontologies was done by introducing a symbolizer microtheory aligned with SE and the graphic ontology based on SVG constructs. The scope of the portrayal ontologies was limited to vector-based (feature-based) representation.

6.2. Goals for Testbed 13

The objective of Testbed 13 in terms of Portrayal Ontology development is to identify the gaps between SLD/SE standards and the ontologies developed during Testbed 12. The scope of this analysis is limited to vector data only. Further work is needed for coverage data (raster data in particular) in future testbeds. To conduct this analysis, a round trip conversion from SLD to Linked Data Representation and vice versa was performed. The goal is to have the portrayal ontologies being, at least, as expressive as SLD/SE and able to support rendering tasks. The second objective is to test the ability of the portrayal ontology to work on models different from XML, by testing its application to Linked Data representation.

6.3. Findings

6.3.1. Tight Coupling of SLD/SE with XML Model

The SLD/SE standards are tightly coupled with the OGC Feature Model and its XML encoding in GML. The implementation of the standards in an OGC Web Map Service (WMS) assumes typically that vector data is provided by OGC Web Feature Service (WFS). Some short notation based on CQL has been introduced to try to bridge the gap, but it is mostly used as syntactic sugar. There are many formats that are not based on XML such as GeoJSON, Linked Data formats (Turtle, JSON-LD, NT), and Comma Separated Values (CSV). Each of these standards uses different schema language (JSON Schema, OWL, RDFS, CSV schema). When it comes to enable semantic interoperability of portrayal information with a feature model, there needs to be a way to represent the feature model semantically and map it to the different schemas encoding existing for each of the format. There is also a need to have an extensible addressing framework that can accommodate different data models.

6.3.2. Identifications

One of the main challenges with the SLD/SE standards is the lack of a global unique identifier for representing the different portrayal information expressed by SLD/SE. The identifiers are identified within the scope of the SLD document and make it hard to reuse and link to other artifacts expressed in other SLDs documents or knowledge base such as feature dictionary.

Another challenge is the ability to identify feature types in unified way that is independent of the data model used. Most of the OGC standards use XML schema and GML to represent feature type definitions. In Linked Data (such as in GeoSPARQL), Feature types are represented through an OWL class Uniform Resource Identifier (URI). Property Binding in SLD uses XPath to represent paths in XML structure. In Linked Data, properties are typically defined globally and defined as URI. SPARQL Protocol and RDF Query Language (SPARQL) and the Shapes Constraint Language (SHACL) provide a mechanism to define RDF Path. A unified approach is needed to define property paths on different data models (JSON, XML, Linked Data, CSV,..).

6.3.3. Expression Bindings

The OGC SE standard uses OGC Filter Encoding standard [OGC 09-026r2] to express portrayal rules conditions and binding expressions to symbolizer attributes. The OGC Filter Encoding standard describes an XML and Key-Value Pair (KVP) encoding of a system-neutral syntax for expressing the projection, selection and sorting clauses of a query expression. The intent is that this neutral representation can be easily validated, parsed and then translated into some target query language such as SPARQL or SQL for processing. The OGC SE standard extends the expression model with some pre-built functions commonly used in Portrayal (categorization, formatting functions). While the goal of the OGC Filter Encoding standard is to define a system-neutral syntax, it suffers of many drawbacks.

- The standard requires to implement converters from OGC Filter to a target native query language (ex. SPARQL, SQL), which are not always trivial to implement against specific target data models.
- Verbosity: The XML encoding of simple expression can be very verbose compared to other standards query language (CQL, SPARQL).
- Lack of a standard mechanism to define and share functions.
- Assumption that the feature model is mappable to XML and can be represented in XML Schema. This is not always the case, always available (ex. JSON, RDF, CSV) or even feasible. Other schema languages can be used such as JSON Schema, OWL, RDF Schema or CSV Schema.

- Use of XML QNames: The QName to URI Mapping is broken when trying to map to RDF ontology. Fundamentally, using “QNames” as abbreviations for URIs is a bad idea. QNames have a number of restrictions on them because they are built to be legal XML Names: the kinds of things that one can call elements and attributes. URIs don’t have these restrictions: it’s perfectly possible for the last part of a URI to consist purely of numbers, or to have a slash at the end, or even to have request parameters. Fair enough that meaningful QNames can be used for some URIs, but if one cannot use them properly for all URIs, then there has to be a better way.

All the parameter attribute values for the symbolizers defined in the OGC SE standard XML encoding require a OGC expression based on the OGC Filter specification. This makes the encoding very verbose in case a user wants to assign a simple value such as stroke-width to a number. It also makes it difficult to validate the expression as the actual types of parameter attributes are not strongly typed.

6.3.4. Feature Type modeling

SLD/SE defines two properties to refer to FeatureType information:

- The **FeatureTypeName** identifies the specific feature type that the feature-type style is for. It can be optional but only if a feature type is in-context or if it is intended for usage for a number of feature types using **SemanticTypeIdentifier**.
- The **SemanticTypeIdentifier** is experimental and is intended to be used to identify what the feature style (or coverages in case of usage inside a CoverageStyle) is suitable to be used for using community-controlled name(s). For example, a single style may be suitable to use with many different feature types. The syntax of the **SemanticTypeIdentifier** string is undefined, but the strings “generic:line”, “generic:polygon”, “generic:point”, “generic:text”, “generic:raster”, and “generic:any” are reserved to indicate that a FeatureTypeStyle may be used with any feature type with the corresponding default geometry type (i.e., no feature properties are referenced in the feature style).

These properties are not well adapted to accommodate different schema languages and do provide mechanisms to indicate where to get additional information about the feature types. For example, OpenStreet Feature could be encoded in GML, JSON or XML and would require one to define a different SLD encoding for each of these schemas. It would be useful to define the feature type at the conceptual (semantic) level and then provide links to different encodings and distributions of the schema. Having a global unique identifier for each feature type will enable the integration/linking of the feature type dictionary with styling information an minimize duplication.

6.3.5. Layer

The OGC SLD standard defines the concept of “layer” as a collection of features that can be potentially of various mixed feature types. A named layer is a layer that can be accessed from an OGC Web Server using a well-known name. For example, the WMS interface uses the LAYER parameter to reference named layers as in the example parameter:

```
LAYERS=Rivers,Roads,Houses
```

The SLD standard defines the concept of **NamedLayer** to represent map layers that can be referred to by name by a different service. The name that is defined locally to the document is not a global identifier. This is an issue when one wants to refer a layer to other concepts that are defined outside the document, such as a dataset used by the layer (which could be defined by a DCAT Dataset instance).

The NamedLayer element in the SLD standard is defined by the following XML-Schema fragment:

```
<xsd:element name="NamedLayer">
  <xsd:complexType>
        <xsd:sequence>
            <xsd:element ref="se:Name"/>
            <xsd:element ref="se:Description" minOccurs="0"/>
            <xsd:element ref="sld:LayerFeatureConstraints" minOccurs="0"/>
            <xsd:choice minOccurs="0" maxOccurs="unbounded">
                <xsd:element ref="sld:NamedStyle"/>
                <xsd:element ref="sld:UserStyle"/>
            </xsd:choice>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
```

The **LayerFeatureConstraints** element is optional in a NamedLayer and allows the user to specify constraints on what features of what feature types are to be selected by the named-layer reference. It uses OGC Filter as the filter language, which is mostly designed for data that are mappable to XML.

A named styled layer can include any number of named styles and user-defined styles, including zero, mixed in any order. If zero styles are specified, then the default styling for the specified named layer is to be used. A named style, similar to a named layer, is referenced by a well-known name. A particular named style only has meaning when used in conjunction with a particular named layer. One of the issues with this approach is that the name of the layer is defined locally. It does not define a globally unique identifier that is referenceable, so it can be reused by other layers defined outside the document.

A **UserLayer** is defined as a subclass of **NamedLayer** for representing a user-defined layer to be built from WFS and WCS data only, and inline features encoded as GML FeatureCollection only. This is too restrictive as it does not allow **UserLayer** leveraging other data sources such as GeoJSON, Linked Data sources accessible from a SPARQL endpoint or other popular formats such as CSV or Shapefile. UserLayer is defined by the following XML-Schema fragment:

```
  <xsd:element name="UserLayer">
    <xsd:annotation>
      <xsd:documentation>
        A UserLayer allows a user-defined layer to be built from WFS and
        WCS data.
      </xsd:documentation>
    </xsd:annotation>
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="sld:Name" minOccurs="0"/>
        <xsd:element ref="sld:RemoteOWS" minOccurs="0"/>
        <xsd:element ref="sld:LayerFeatureConstraints"/>
        <xsd:element ref="sld:UserStyle" maxOccurs="unbounded"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
```

A user-defined style allows map styling to be defined externally from a system and to be passed around in an interoperable format. The XML-Schema fragment for the UserStyle SLD element is defined as follows:

```
<xsd:element name="UserStyle">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element ref="se:Name" minOccurs="0"/>
            <xsd:element ref="se:Description" minOccurs="0"/>
            <xsd:element ref="sld:IsDefault" minOccurs="0"/>
            <xsd:choice maxOccurs="unbounded">
                <xsd:element ref="se:FeatureTypeStyle"/>
                <xsd:element ref="se:CoverageStyle"/>
                <xsd:element ref="se:OnlineResource"/>
            </xsd:choice>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
<xsd:element name="IsDefault" type="xsd:boolean"/>
```

A **UserStyle** can contain one or more **FeatureTypeStyles** or **CoverageStyles** which allow the rendering of features of specific types. These are described in OGC Symbology Encoding. These styles can either be provided inline within the SLD document or they can be referenced using an **OnlineResource** containing a OGC SE document with a **FeatureTypeStyle** or **CoverageStyle** root element. This organization allows the more convenient use of feature-style libraries.

There is no standard that allows the management of user-defined layers and enables global referencing of layers to enable the reuse of the layers in different maps. There is a need for an API and global referencing of layers, as well as the need to create and describe the concept of a layer in multiple data models and APIs.

6.3.6. Legend

Legend plays a central role in the interpretation of map. Legends are typically included with maps to indicate to the user how various features are represented in the map or on the layer. It is therefore important to be able to produce a legend on a map display client. Generating a legend on the client side may involve a significant amount of processing. The client will need to examine the selected style and determine which rules apply at the currently used map scale. While this would save some interactions between the client and server and would allow the viewer client to present consistent sample shapes (across remote map servers from different vendors), the legend graphics might look different from the graphics actually rendered in the map since the viewer and server may have different rendering engines and different graphical capabilities. A better approach is either to associate the legend with a given rule or style or to provide API endpoints to generate legend for a style and legend item glyphs for a given rule or symbolizer.

There are currently three OGC specifications that are relevant to represent Legends: OGC Symbol Encoding (SE), OGC Style Layer Descriptor 1.0 and 1.1, and OGC Web Map Service (WMS) specifications.

In the OGC SE specification, a **Rule** can be associated with a **LegendGraphic** element referring to a **Graphic Symbolizer**. The Graphic Symbolizer can refer to an external graphic using a URL or defined as a **Mark**. **LegendGraphic** only has a limited role in building legends. For vector types, a map server would normally render a standard vector geometry (such as a box) with the given symbolization for a rule. But for some layers, such as for Digital Elevation Model (DEM) data, there is not really a “standard” geometry that can be rendered to get a good representative image. This is what the LegendGraphic SE element is intended for, to provide a substitute representative image for a Rule. For example, it might reference a remote URL for a DEM layer called “GTOPO30”:

```
http://www.vendor.com/sld/icons/COLORMAP_GTOPO30.png
```

The OGC SLD specification defines the operation GetLegendGraphic to generate a legend as an image from an SLD style, rule and feature type with flexible legend options to render the layout and labels of the legend on the server side. An SLD-WMS operation request for GetLegendGraphic can look like this encoded in KVP:

```
http://www.vendor.com/wms.cgi?
 VERSION=1.1.0&
 REQUEST=GetLegendGraphic&
 LAYER=ROADL_1M%3Alocal_data&
 STYLE=my_style&
 RULE=highways&
 SLD=http%3A%2F%2Fwww.sld.com%2Fstyles%2Fkpp01.xml&
 WIDTH=16&
 HEIGHT=16&
 FORMAT=image%2Fgif&
```

This would produce a 16x16 icon for the Rule named “highways” defined within layer “ROADL_1M:local_data” in the SLD. The list of available formats for legend graphics and exceptions can be assumed to be the same as are available for a map in the WMS GetMap request.

In OGC Web Map Service 1.3 specification, a **Style** may contain a **LegendURL** that provides the location of an image of a map legend appropriate to the enclosing style. A **Format** element in **LegendURL** indicates the MIME type of the legend image, and the optional attributes width and height state the size of the image in pixels. Servers should provide the width and height attributes if known at the time of processing the GetCapabilities request. The legend image should clearly represent the symbols, lines and colors used in the map portrayal. The legend image should not contain text that duplicates the Title of the layer, because that information is known to the client and may be shown to the user by other means.

One of the challenges encountered today by web developers if the display of legends for map. In many instances, legends are returned as bulk images containing multiple items for different symbolizers. Most of the modern web applications today are using Responsive Web design, an approach that suggests that design and development should respond to the user's behavior and environment based on screen size, platform and orientation. This implies that the client needs to have the ability to layout legend items in a flexible way with positioning of their labels depending of the screen formats. The use of an image for the whole legend is well adapted for responsive design. A more flexible mechanism is needed to convey the whole legend of a map by decomposing legend items that can be customized by a REST endpoint. Very often a legend image is also encoded inline using base64 encoding instead of referring to remote URL, which can add overhead when fetching each individual item.

The Portrayal Service should also accommodate the custom rendering of legend items (glyphs) for symbol, symbolizers and rules to provide visual cues when searching these items in the portrayal service. To address this challenge, an ontology was designed and encoded to describe Legend and Legend Item that can be used in portable way. The full model is documented in Appendix A.

6.4. Design

6.4.1. Expression Ontology

To address the issues related to the encoding of expressions in the previous section, an extensible approach was adopted. It leverages the heterogeneity of the data models, schema languages and query language standards, instead of attempting homogenizing the query language. There are a number of well-defined standards that are used for different data models. In the case of Linked Data, the standard SPARQL and its OGC extension GeoSPARQL are well-established. For XML, XQuery is the W3 standard. Each of these languages has standard mechanisms for defining functions. The advantage of this approach is that the full expressiveness of each language and existing query engines available can be leveraged without requiring a conversion process.

The key idea of the adopted approach is to treat expressions as literals and associate a standard URI for the query language of the expression. A similar approach has been used in OWL-S for representing conditions and parameter bindings in workflow of web services.

A standalone ontology for expression was defined, anticipating that it could be reused in other standards that require query expression in different languages.

The core concept of the ontology is Expression. It has only two properties: **expressionBody** that captures the expression as a literal and **expressionLanguage**, which refers to the URI of standard query language. The ontology introduces two convenience subclasses: OGC Expression and SPARQLExpression which have a fixed value to a query language (see Figure 1)

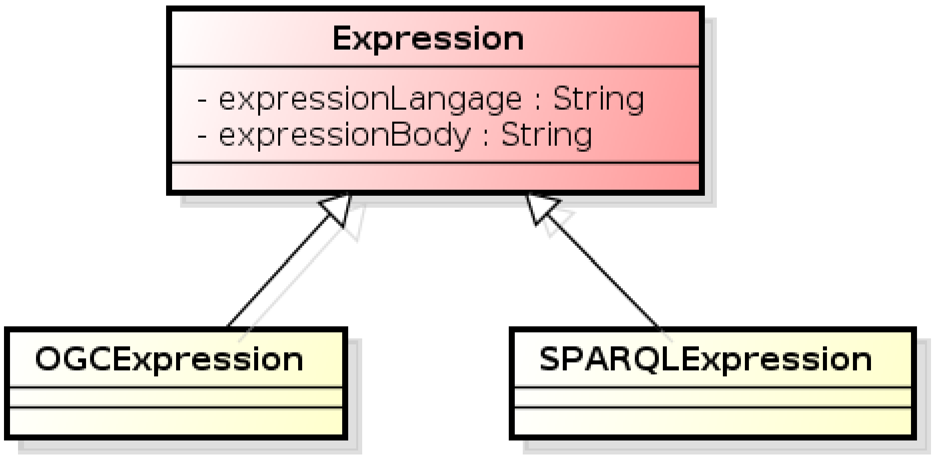


Figure 1. Expression Model

The following are URLs for the query languages that have been used:

Language	URL
SPARQL Query 1.0	http://www.w3.org/ns/sparql-service-description#SPARQL10Query
SPARQL Query 1.1	http://www.w3.org/ns/sparql-service-description#SPARQL11Query
OGC Filter	

This ontology was used successfully to convert SLD portrayal rules expression, but not fully tested for the rendering of the feature data. The Expression ontology is documented in Annex A.

6.4.2. Binding Ontology

In the symbolizer and graphic ontologies, the graphic properties have a range that corresponds to the type of their values. Using expressions directly as values, will invalidate the model against the ontology rules (i.e. if a range for graphic property is defined as a xsd:integer, assigning an expression object will be invalid in OWL). To address the issue of validation of parameter values (**SVGParameter**) in SE Encoding, a lightweight Binding ontology was introduced. Symbolizer and PortrayalRule are defined as subclasses of **Parameterizable**. A **Binding** can be attached to any Parameterizable Object (see Figure 2). It assigns an expression to a property of the **Parameterizable** instance.

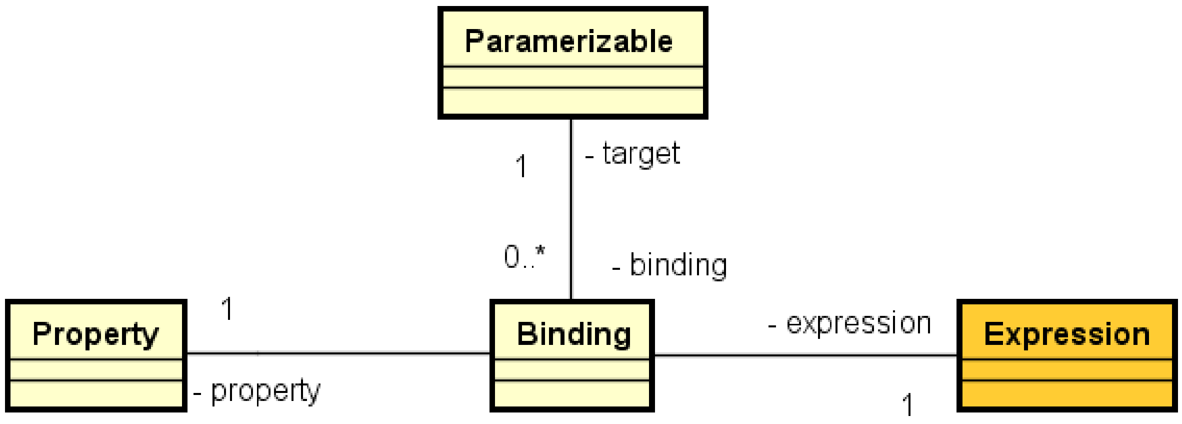


Figure 2. Binding Model

This ontology was used successfully to convert SLD portrayal rules expression bindings, however was not tested for the rendering of the feature data.

The Binding ontology is documented in Annex A.

6.4.3. Legend Ontology

A **Legend** plays a central role in the understanding of the meaning of a map or layer. It associates symbols used in a **style** with its intended denotation (meaning). This meaning is often associated with a human readable label but could also be associated with a machine-processable concept. A layer can have multiple layer styles. For each style, there is a legend associated with it.

A formal model for a legend was designed to enforce best practices that can make it easier for a client to layout legends in a variety of screen sizes automatically. To address this, the legend was broken down into a set of individual items that can be laid out. The model can still represent a legend with different symbolizers as one image using only one item, however it would be preferable to decompose each symbolizer/rule as one legend item to obtain more flexibility in the layout of the legend.

The Legend ontology is documented in Appendix A.

6.4.4. Layer Model

For the Testbed 13, an initial model was developed to represent a map layer that addresses some of the findings described in the previous section. This model was introduced around the end of the implementation phase in order to support the creation, update, cloning and search of map layers. The model may need some refinement in future testbeds.

The layer model is slightly different from the one defined in current OGC services, in the sense that it can accommodate multiple data sources using different models and formats (XML, JSON, CSV, Shapefile, WFS, GML). These data sources are typically accessible from a URL but could be set inline to capture annotations or small datasets (as demonstrated in Testbed with GeoJSON). Each data source can have several parameters that are represented as a set of key value pairs. The description of the parameters should be provided in the capabilities document of the portrayal service or a dedicated endpoint. This aspect has only been partially addressed in this testbed due to lack of time.

The focus of the layer modeling within the Portrayal Service was to capture the information necessary to perform the rendering of the layer. The design was not focused on the capture of metadata to enable search and discovery of layers in a semantic registry, which was investigated in another Testbed 13 Thread related to the Semantic Registry Information Model (SRIM) [11]. While there is some overlap between both models, the focus was to capture the essential metadata needed to render the layer.

Future testbeds will need to investigate the reconciliation of the SRIM profile for layer and map concepts and the layer description of the portrayal service needed to perform its rendering. The role of each service should also be clarified, for example, which service should capture metadata about layers. The definition of Data source needs also to be reconciled with the Distribution defined in DCAT. Coordinating efforts with W3C Spatial Data On The Web Working Group will help to resolve some of these issues.

Chapter 7. Semantic Portrayal Service

This section summarizes the findings, design approaches, and changes for the Semantic Portrayal Service.

7.1. Findings

During the OGC Testbed 10 and 11, the initial set of portrayal ontologies have been developed to represent point-based symbols. In OGC Testbed 12, the model has been extended to support symbolizers for line and polygons. A design and implementation of a REST-based Semantic Portrayal Service was accomplished to manage portrayal information using the Semantic Registry Service based on the Semantic Registry Information model (SRIM) developed during the Testbed 12. REST CRUD (create, read, update, and delete) operations were implemented to manage and search styles, symbolizers, symbols using dedicated endpoints for each. Due to the lack of time, the rendering endpoints for layers, symbolizers and legends were not implemented. The service also didn't have the ability to manage layers, perform faceted search on portrayal items and perform bulk import of portrayal information either encoded as SLD documents or Linked Data Format.

The goal of OGC Testbed 13 was to implement the rendering endpoints needed to demonstrate the capabilities of the Semantic Portrayal Service. Another goal was to import and export SLD documents and identify and address the gaps between the ontological model and the SLD model.

During the investigation of these gaps, it was identified that map layer management was also needed. Users want to create layers with customized styles that can be saved, so the layers can be easily accessible through a REST API. None of the current OGC service specifications support the ability to manage and search layers with user-defined styles.

The need of the definition of a profile for the Semantic Registry for layers and maps metadata was identified. This task was addressed by another thread in Testbed 13. The portrayal effort of this thread will need to be reconciled with the results of the activities of the Semantic Registry Thread in future testbeds.

7.2. Design

7.2.1. REST API Design

The Semantic Portrayal Service implementation is accessible through a hypermedia-driven and Linked Data REST-based API to access layer and portrayal information (styles, rules, symbolizers, symbols) from the service. The Semantic Portrayal implements REST API Level 3 and Level 2 of the Richardson Maturity Model (see Testbed 12 ER) and Linked Data API.

The style information is encoded in the following representations: RDF/XML, Turtle, N-Triples, JSON-LD and HAL-JSON. The Semantic Portrayal Service REST API is described in more detail in Annex B.

7.2.2. Importing Portrayal Information

To import portrayal information, a pluggable, extensible design approach was adopted, so it can accommodate a variety of formats, as standards evolve in the industry. Each importer type provides a list of parameters with name, description, type and cardinality. This information is used to build User Interface (UI) forms to capture value bindings for each parameter. For the OGC Testbed 13 initiative, importers for SLD 1.0, SLD 1.1 and Linked Data formats (RDF, Turtle, NTriples), based on the portrayal ontologies, were implemented. The importers can work with remote URL or file attachment.

7.2.3. Import SLD

The OGC SLD importer was designed to import SLD documents from either a URL or a file attachment. The imported document was parsed and mapped to the updated portrayal ontology and registered in the Semantic Registry implementing the Portrayal Information. A complete mapping of all related feature style information to the portrayal ontology was accomplished successfully by updating the ontology model with expression and bindings micro-ontology. It was found that the mapping was complete and feasible. However, the ontology had more expressiveness than OGC SLD as it could accommodate multiple data models (JSON, XML, RDF) and expression languages.

7.2.4. Linked Data Import

The Linked Data model provides a powerful integration mechanism to represent any objects, properties and relationships that can be understood unambiguously by machine (to perform inferences for example). The OGC Testbed 12 implementation of the Semantic Portrayal Service provided the ability to perform transactions of portrayal items in a very granular way by using dedicated endpoints for each item type (symbols, styles, symbolizers). While this was a valid approach, the performance to upload a large set of portrayal information was poor. It also limits one of the main benefits of using Linked Data: linking objects of different types in a semantic graph.

For Testbed 13, an importer for Linked Data formats (RDF/XML, Turtle, N-Triples) was implemented to upload any portrayal concepts managed by the portrayal services and expressed by the portrayal ontologies used by the service. The Linked Data Model was processed on the server side by iterating on all instances of resource types supported by the service and indexing them in the semantic registry. URIs of the resources were used to generate internal identifiers consistently. It was observed that significant improvement in bulk upload time compared to the more granular approach taken in the previous Testbed.

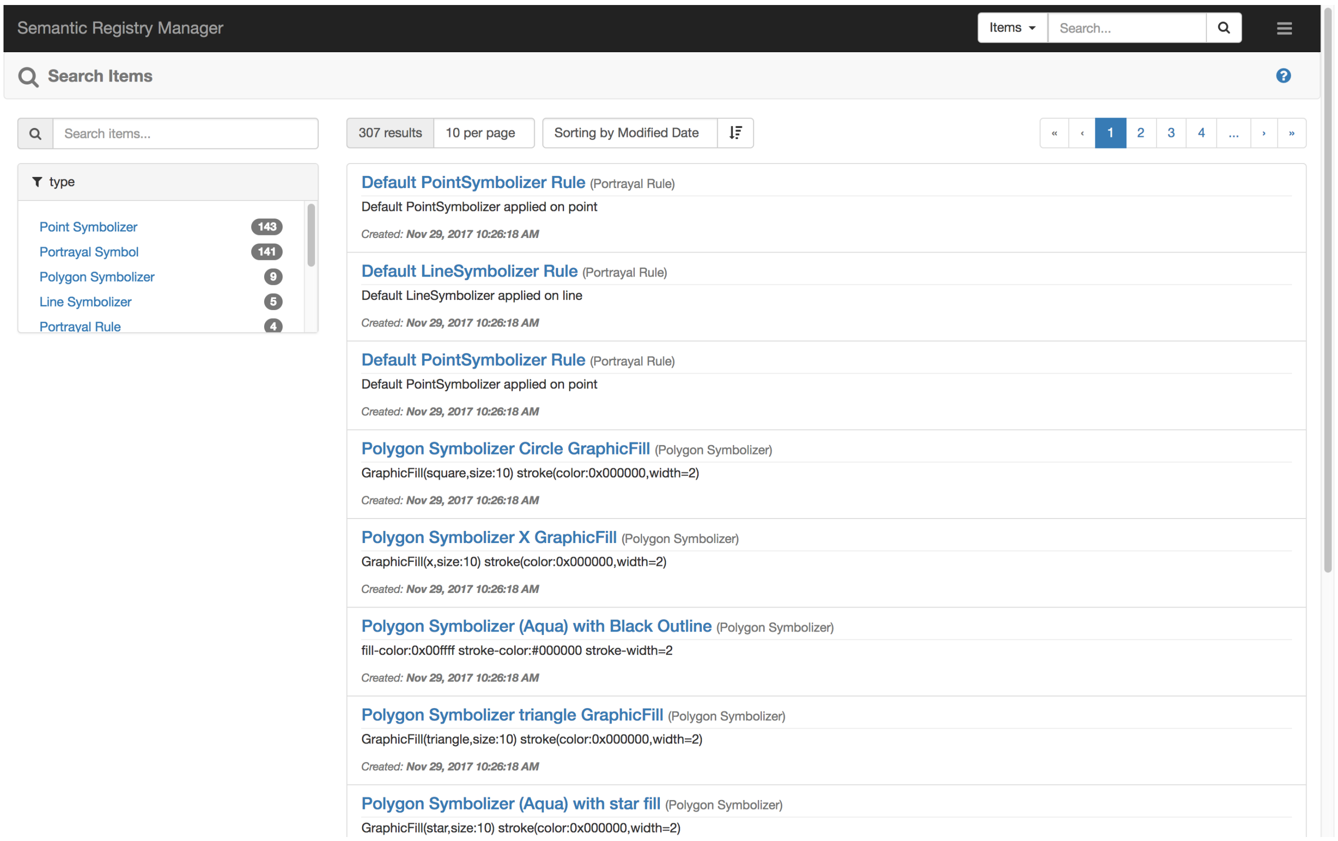
A future improvement, that can be addressed in future OGC Testbeds for this endpoint, is to integrate Shape Constraint Language (SHACL) validator to validate the shapes of the objects to make sure they contain the required properties needed by the service to perform their tasks. An investigatigation of the use of Linked Data for exporting all portrayal information managed by the service should also be performed.

7.2.5. Export SLD

Exporting SLD XML from the portrayal service was done through the `/export` endpoint. Using the portrayal ontologies, the convertion of FeatureTypeStyle ontological concept to a full SLD document with loss was done without loss. However, the export of symbolizers alone was not possible without breaking the XML validation against the SLD document schema.

7.2.6. Integration with Portrayal Registry

The Semantic Portrayal Service was using the Portrayal Registry to store style items.The SRIM profile was updated to accommodate the portrayal items by using the portrayal ontologies. An endpoint to the portrayal registry was also added to import Linked Data for portrayal information. The import endpoint of the Portrayal Service was interfaced with the Semantic Registry import by forwarding the request to the registry. The search of portrayal items in the Semantic Portrayal Service was delegated to the Semantic Registry. The following client Figure 3 shows the portrayal information stored in the semantic registry.



Point Symbolizer	143
Portrayal Symbol	141
Polygon Symbolizer	9
Line Symbolizer	5
Portrayal Rule	4

Default PointSymbolizer Rule (Portrayal Rule)
Default PointSymbolizer applied on point
Created: Nov 29, 2017 10:26:18 AM

Default LineSymbolizer Rule (Portrayal Rule)
Default LineSymbolizer applied on line
Created: Nov 29, 2017 10:26:18 AM

Default PointSymbolizer Rule (Portrayal Rule)
Default PointSymbolizer applied on point
Created: Nov 29, 2017 10:26:18 AM

Polygon Symbolizer Circle GraphicFill (Polygon Symbolizer)
GraphicFill(square,size:10) stroke(color:0x000000,width=2)
Created: Nov 29, 2017 10:26:18 AM

Polygon Symbolizer X GraphicFill (Polygon Symbolizer)
GraphicFill(x,size:10) stroke(color:0x000000,width=2)
Created: Nov 29, 2017 10:26:18 AM

Polygon Symbolizer (Aqua) with Black Outline (Polygon Symbolizer)
fill-color:0x00ffff stroke-color:#000000 stroke-width=2
Created: Nov 29, 2017 10:26:18 AM

Polygon Symbolizer triangle GraphicFill (Polygon Symbolizer)
GraphicFill(triangle,size:10) stroke(color:0x000000,width=2)
Created: Nov 29, 2017 10:26:18 AM

Polygon Symbolizer (Aqua) with star fill (Polygon Symbolizer)
GraphicFill(star,size:10) stroke(color:0x000000,width=2)

Figure 3. Portrayal Registry Client

7.2.7. Rendering

For the Testbed 13 initiative, the different types of renderers for the portrayal service were identified:

- **Layer Renderer**: render layers managed by the service but also transient layers (layer specification submitted to the renderer but not managed by the service).
- **Map Renderer**: Render a map from multiple layers managed by a portrayal service. It was chosen to follow the WMS GetMap operation protocol to facilitate the integration with existing web map client such as Leaflet, OpenLayers or MapBox. These APIs do not require a full implementation of the WMS specification as most of them only use the GetMap operation and ignore the GetCapabilities operation.
- **Glyph Renderer**: Render a symbolizer/symbols into an image that can be used as a legend item

or preview of symbolizers (or symbols). This is useful when symbolizers are defined by users and clients need a way to generate a legend for the symbolizer.

- **Legend Renderer**: Generate a legend object composed of multiple legend items (each one representing a symbolizer corresponding to a portrayal rule. A Legend ontology was developed to describe the legend and legend items in JSON and Linked Data form. This would facilitate the interoperability of sharing legend information but also provide flexibility to clients to perform customized layout of the legend for specific targeted device display. Due to lack of time, the implementation of the renderer was not implemented.

The details of each renderer design are as follows:

Layer Renderer

While developing the client for the portrayal service, the following use cases for rendering layers were identified:

- Render a layer by using its identifier with its associated styles.
- Render a layer by using its identifier and overwrite its styling information using different bindings for its associated styles or replacing the associated styles with new styles.
- Render a transient layer by passing its data source information and styles information explicitly

To address these use cases, different endpoints for layer rendering were introduced:

To address the first use case, which renders a layer by referring its internal id, a convenience rendering endpoint was provided using REST principles following the following pattern:

```
/layers/{id}/renderer
```

This endpoint renders the layer based on its definition using its associated source and styles. When this endpoint is used without parameter, the default extent of the data sources is used, a default size proportional to the bounding box and the CRS of the dataset. By making all the parameters optional, this provides a quick way to get a rendering of the layer by using a simple URL address. However, the rendering endpoint also accepts parameters such as CRS, BBOX, Width, Height, Style or Symbolizer identifiers to provide custom rendering of the layer within a given bounding box in a given CRS and rendered in a given image size.

When URI of layers are used to reference a layer, a second endpoint was defined. It works in a more flexible way to render the layer using the following URL pattern:

```
/renderer/layer
```

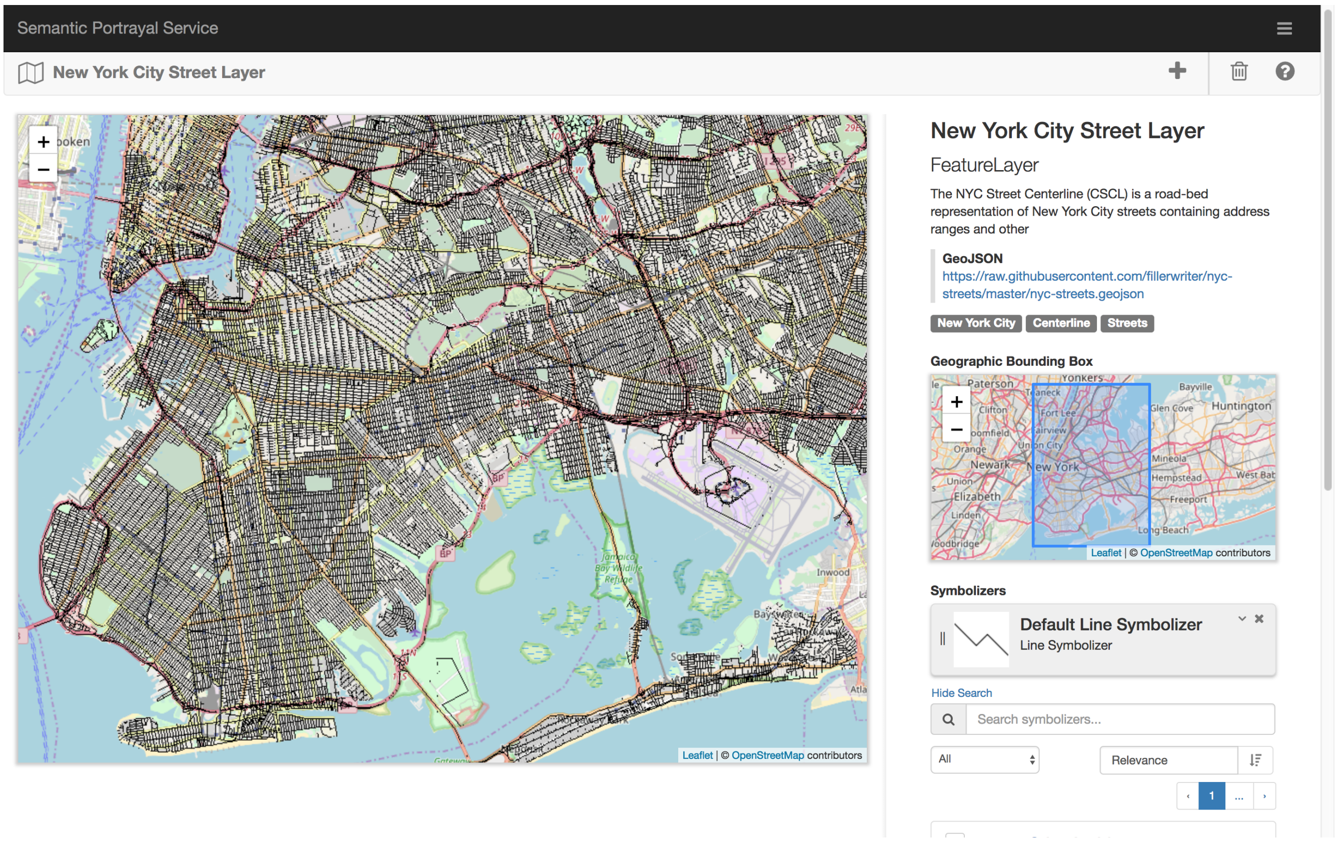
This endpoint requires either a layer internal id or a URI to refer to a layer and uses the same parameters as the former endpoint. This endpoint allows the rendering of a layer that can be accessible remotely using a resolvable URL. It was suggested to investigate distributed rendering of layers in future testbeds.

Details of the layer renderer endpoints are provided in Appendix B.

Map Renderer

To facilitate the integration of the Portrayal Service with existing clients, a map renderer endpoint was implemented using the WMS GetMap protocol parameters of the HTTP Get operation such as BBOX, CRS, WIDTH, HEIGHT, STYLE, FORMAT parameters. The Map Renderer endpoint is documented in Annex B.

Most of the Web Map clients that support WMS protocol such as OpenLayer, MapBox, Leaflet, only use the GetMap operation endpoint of the standard. The GetCapabilities is typically used by developers to get the identifier of the layer. The approach taken by the client is compatible with this approach. The only difference is that the layers can be discovered with the layer search endpoint and that each layer is modeled as a REST resource that is addressable with a unique URL and id. To test the viability of the approach, the integration of the Portrayal Service with the popular Leaflet Map Client was successfully implemented, by getting a portrayal service layer overlapping OpenStreetMap base layer using reprojection and rendering of the layer with custom styles. For the demonstration, symbolizer identifiers were used to refer to the styles (see Figure 4).


Semantic Portrayal Service
New York City Street Layer
Leaflet | © OpenStreetMap contributors

New York City Street Layer

FeatureLayer

The NYC Street Centerline (CSCL) is a road-bed representation of New York City streets containing address ranges and other

GeoJSON
https://raw.githubusercontent.com/fillerwriter/nyc-streets/master/nyc-streets.geojson

Figure 4. Map Rendering using WMS GetMap protocol

In future testbeds, more testing needs to be done with the map renderer to use Symbols, FeatureTypeStyle, and CoverageStyle identifiers as style parameters. As more implementations of the Portrayal Service become available, an investigation should be conducted to use Layer resource URL instead of internal identifiers to access layers remotely and rendering the map using a cascading behavior similar to the cascading behavior of WMS.

Glyph Renderer

To support rendering of symbols and legends, dedicated renderer endpoints for symbolizers and symbols were defined. The endpoint accepts an id or the URI of the symbolizer and optionally width and height of the output formats. For the testbed, the Testbed 11 Canadian EMS icons (encoded in PNG) were used to render point glyphs. The rendering of well-known shapes for point symbolizers, and basic line and polygon symbolizers was demonstrated for displaying previews of symbols and symbolizers in the client. Future testbeds need to further investigate more advanced symbolizer specifications including the usage of some conditions and expression bindings in portrayal rules. The API of the endpoint is documented in Appendix B.

7.2.8. Layer Management API

By the end of the implementation phase of the testbed, the need of implementing a mechanism to persist layer specification was identified, so layers can be referenced easily to construct maps composed of multiple layers. The API and Layer Model design are not as mature as other endpoints developed during this testbed but it provides a good starting point for future investigations and refinements.

In order to make it easier to reference the layer and enable the sharing of layer information over the web, a RESTful approach was used to manage layers. Standard CRUD operations using POST, PUT, DELETE and GET were adopted. An initial model was designed to represent Layer information including Data sources, and Styling information (see previous section). The API is anticipated to be changed based on the enhancements needed to the layer model (in particular with the alignment of the SRIM model).

More detailed information of the REST API for Layer Management can be found in Appendix B.

Chapter 8. Portrayal Demonstration

This section summarizes the findings, design approaches, and workflows used for the Portrayal Thread Demonstration.

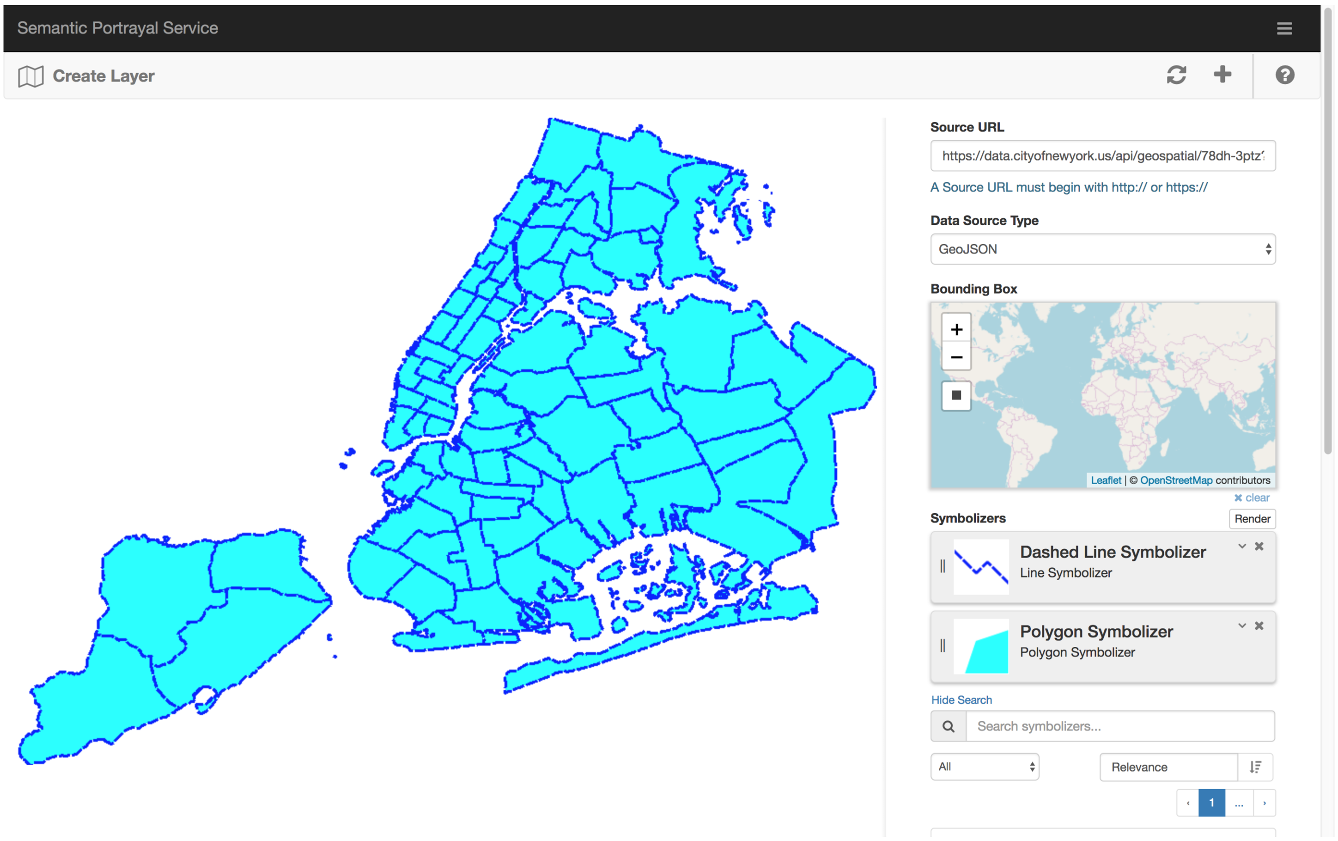
8.1. Datasets

To demonstrate the robustness and flexibility of the portrayal model to accommodate different data models and formats, the following formats and protocols to build map layers were implemented and tested:

8.1.1. GeoJSON

GeoJSON is a format for encoding a variety of geographic data structures. GeoJSON supports the following geometry types: Point, LineString, Polygon, MultiPoint, MultiLineString, and MultiPolygon. Geometric objects with additional properties are Feature objects. Sets of features are contained by FeatureCollection objects. In 2015, the Internet Engineering Task Force (IETF), in conjunction with the original specification authors, formed a GeoJSON Working Group (WG) to standardize GeoJSON. RFC 7946 was published in August 2016 and is the new standard specification of the GeoJSON format, replacing the 2008 GeoJSON specification.

For this testbed, GeoJSON as data sources for making layers was used in two ways. The first way was to access GeoJSON from a remote URL. For the demonstration, a Github repository of GeoJSON data for countries, states and New York City roads and Police Precincts (see Figure 5) was used.



Source URL

https://data.cityofnewyork.us/api/geospatial/78dh-3ptz

A Source URL must begin with http:// or https://

Data Source Type

GeoJSON

Bounding Box

Symbolizers

Dashed Line Symbolizer
Line Symbolizer

Polygon Symbolizer
Polygon Symbolizer

Figure 5. Remote GeoJSON Layer

The use of GeoJSON inline was also tested in the Layer descriptions as a way to capture annotations in web-based applications. To perform this task, two sets of layers were created: the first set retrieved data remotely using a URL, the second set was defined the same layer using inline GeoJSON.

8.1.2. Shapefile

The shapefile format is a popular geospatial vector data format for geographic information system (GIS) software. It is developed and regulated by Esri as a (mostly) open specification for data interoperability among Esri and other GIS software products. The shapefile format can spatially describe vector features: points, lines, and polygons, representing simple feature data. A Geospatial Dataset encoded in Shapefile format is composed of multiple files. However many services distribute shapefiles in zip file format, so they can be bound together as a coherent set of files. For this reason, the implementation of the portrayal service supports shapefiles that are packaged as a zip file that can be accessed remotely via a simple URL. For the demonstration, shapefiles describing airports of the world were used and portrayed with icon based symbolizers.

8.1.3. Web Feature Service

For the demonstration, a Technology Integration Experiments (TIE) was performed with the Web Feature Service 1.1.0 from GeoSolutions that provided data supporting a mass migration scenario. A significant amount of time was spent solving some issues related to the coordinate order of the WGS-84 coordinate reference system handled by the WFS Client library and GeoServer implementation. The notation EPSG:4326 and the URI form are still misunderstood in the industry causing issues with interoperability. For Testbed 13, the dynamic creation and custom rendering of layers based on WFS data returned in GML were successfully demonstrated.

8.1.4. GeoSPARQL endpoint

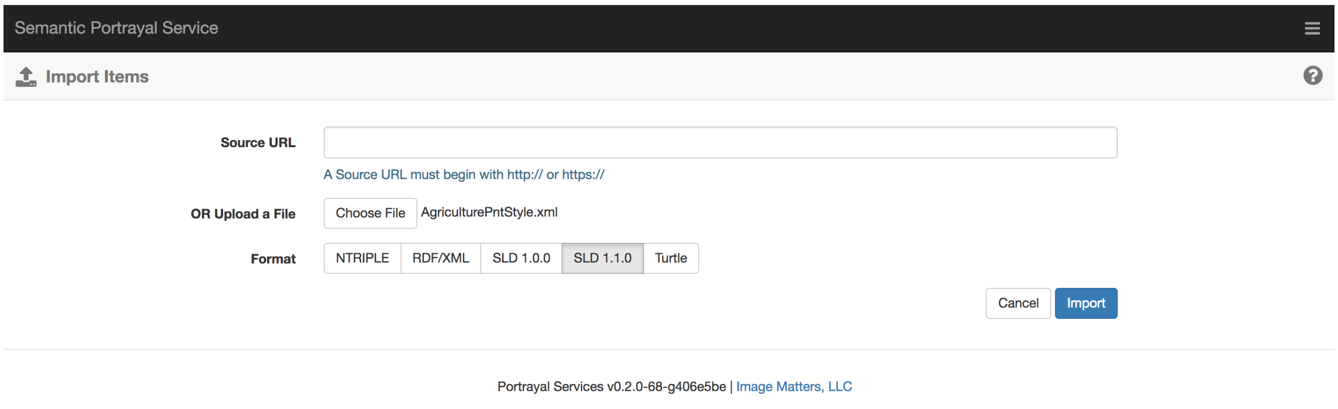
For this testbed, Image Matters deployed its GeoSPARQL-compliant Server populated with OpenStreetMap data produced in Testbed 12. Some development was started to implement an OGC Query/Filter converter to GeoSPARQL. Unfortunately, due to technical difficulties related to the WFS Client and CRS conventions and the last minute addition of a Layer Management REST API on the Portrayal Service, the full integration of GeoSPARQL with the Portrayal Service was not achieved. This aspect can be addressed in next testbeds, as it could provide a powerful mechanism to portray any Geospatial Linked data on the web.

8.2. Import/Export of Portrayal Information

The ability to upload OGC SLD 1.0 and SLD 1.1 documents to import portrayal information including FeatureTypeStyle, Rules and Symbolizers, was added to the client implementation. The documents were converted into an RDF encoding compliant with the portrayal ontologies and were imported into the Semantic Registry as a bulk operation to improve performance of ingesting a large number of items into the registry. One of the limitations of the conversion of SLD to Linked Data representation was the lack of descriptive information for some of the elements in the SLD documents, making it difficult to display in a user-friendly way so that a user can interpret the meaning of the portrayal information properly.

The client also provided the ability to directly import an RDF model using the portrayal ontology to represent feature types, symbolizers, symbols, rules, rule lists, graphic information using the bulk import operation in the Semantic Registry.

The resulting imports from SLD and Linked Data representation in the Semantic Registry were immediately available to the Semantic Registry Service and accessible via its REST API. The user interface allows the upload of SLD documents from a URL or a file attachment (see Figure 6).


Semantic Portrayal Service

Import Items

Source URL

A Source URL must begin with http:// or https://

OR Upload a File: Choose File AgriculturePntStyle.xml

Format: NTRIPLE | RDF/XML | SLD 1.0.0 | SLD 1.1.0 | Turtle

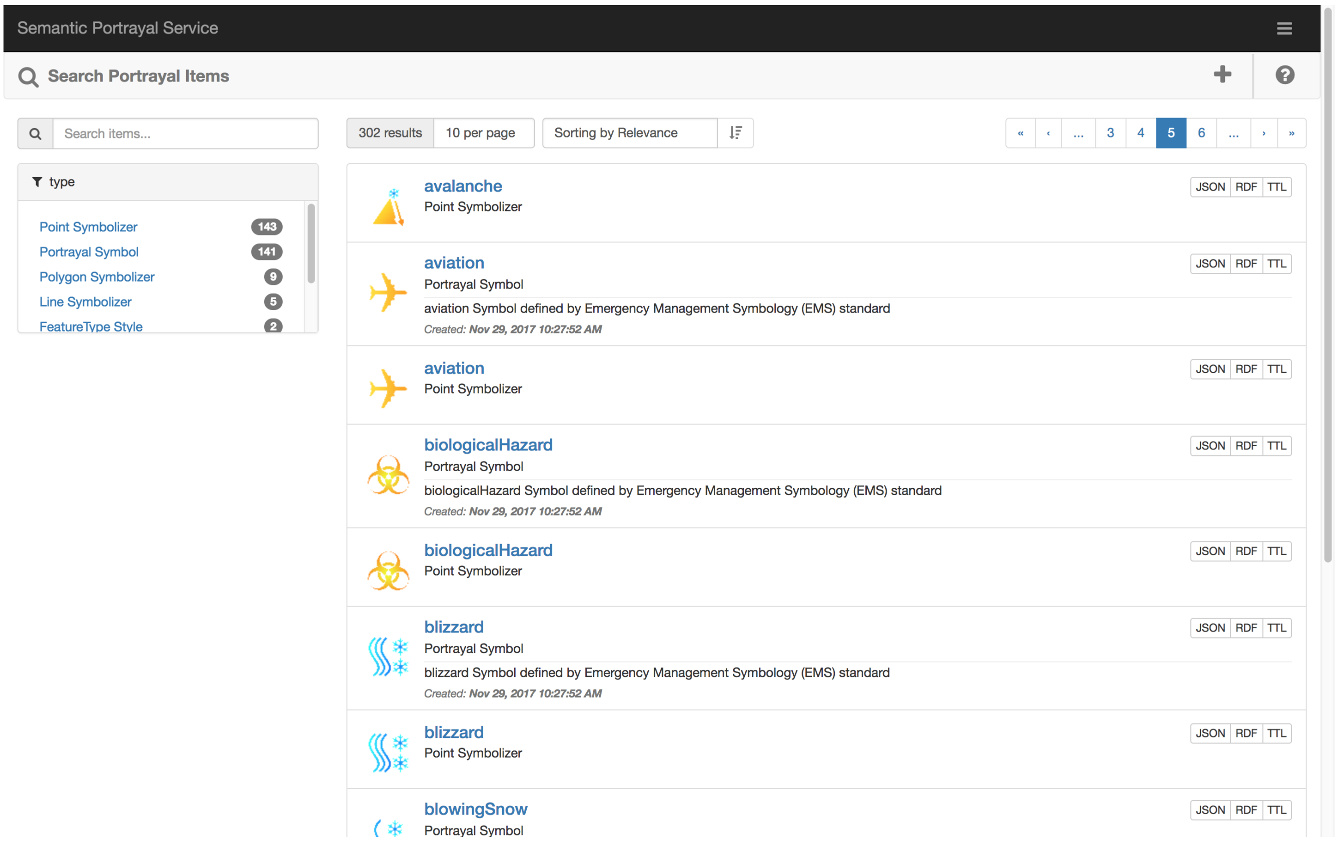
Cancel Import

Portrayal Services v0.2.0-68-g406e5be | Image Matters, LLC

Figure 6. SLD Import UI

8.3. Portrayal Information Search

To demonstrate the search capabilities of portrayal information, a faceted search UI has been built to search portrayal items by type or free text (see Figure 7). More facets may be investigated in the future.


Semantic Portrayal Service

Search Portrayal Items

Search items...

302 results | 10 per page | Sorting by Relevance

« ‹ ... 3 4 5 6 ... › »

type
- Point Symbolizer 143
- Portrayal Symbol 141
- Polygon Symbolizer 9
- Line Symbolizer 5
- FeatureType Style 2

avalanche
Point Symbolizer

aviation
Portrayal Symbol
aviation Symbol defined by Emergency Management Symbology (EMS) standard
Created: Nov 29, 2017 10:27:52 AM

aviation
Point Symbolizer

biologicalHazard
Portrayal Symbol
biologicalHazard Symbol defined by Emergency Management Symbology (EMS) standard
Created: Nov 29, 2017 10:27:52 AM

biologicalHazard
Point Symbolizer

blizzard
Portrayal Symbol
blizzard Symbol defined by Emergency Management Symbology (EMS) standard
Created: Nov 29, 2017 10:27:52 AM

blizzard
Point Symbolizer

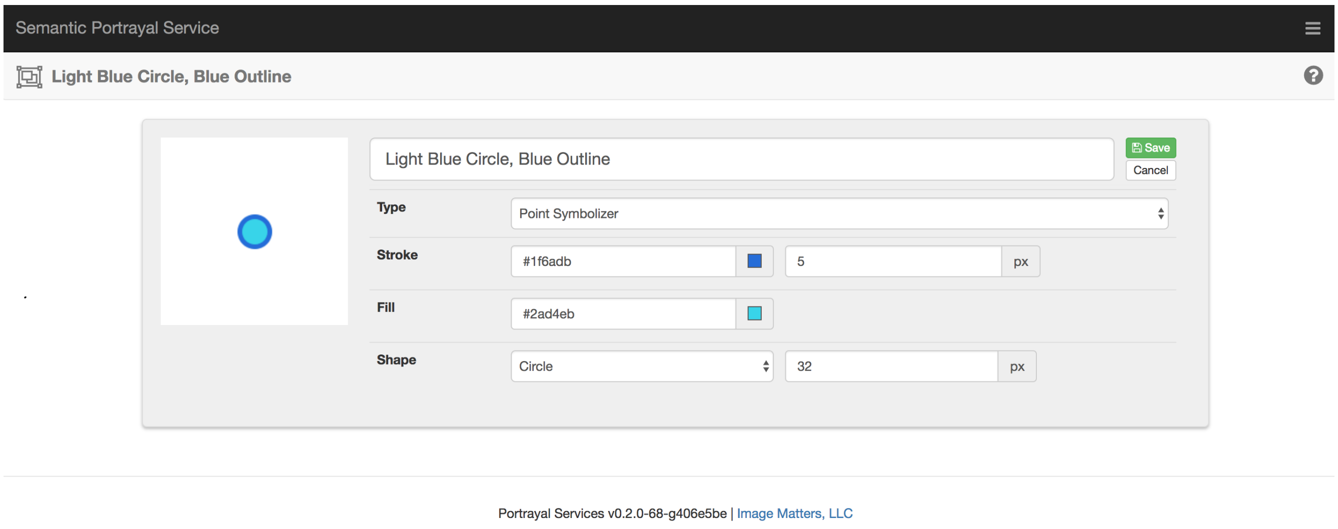
blowingSnow
Portrayal Symbol

JSON RDF TTL

Figure 7. Portrayal Item Search

8.4. Symbolizer Editor

To demonstrate the ability to create, share and apply customized symbolizers to different data sources in order to create a new layer or modify existing layers, a set of editors for creating and updating point, line and polygon symbolizers was implemented in the portrayal client. The initial implementation supports only static symbolizers with fixed values on graphic properties. Future improvements to be addressed in future testbeds include the completion of the symbolizer editors by supporting all the graphic attributes defined in the model and support bindings to complex expressions using feature data information. This would help to test the versatility of the binding model on different data source types (JSON, XML, Linked Data). Figure 8 illustrates the creation of symbolizers.


Semantic Portrayal Service
Save
Cancel
Portrayal Services v0.2.0-68-g406e5be | Image Matters, LLC

Light Blue Circle, Blue Outline

Light Blue Circle, Blue Outline

Type	Point Symbolizer	
Stroke	#1f6adb	5 px
Fill	#2ad4eb	
Shape	Circle	32 px

Figure 8. Symbolizer Editor UI

8.5. Layer Management

By the end of the implementation phase of Testbed 13, the need to create and save user-defined layers was identified. The Semantic Portrayal Service REST API was extended to support CRUD operations to create, update, delete and retrieve layers. The initial implementation saves the layers in the Semantic Portrayal Service store. Future work will need to harmonize the layer model with the SRIM Profile for Layers and Maps and synchronize the layer repository with the Semantic Registry, so layers can be discovered in the Registry to support search and discovery of layers in other applications.

8.5.1. Layer Creation

To demonstrate the REST API for Layer management, it was added in the portrayal web client the functionalities of creating new layers from different data sources such as Zipped Shapefile, GeoJSON, WFS and apply style information on this layer with interactive feedback in a map view. The order of application of symbolizers was controllable in the UI. The rendering of the layer in the map was delegated to the server by accessing the layer rendering endpoint with information about the view context. The rendering endpoint reprojected from the data to the viewer target Coordinate Reference System (CRS) as needed (see Figure 9).

Figure 9 shows the creation of a layer from a dataset of Police Precinct in New York City encoded in

GeoJSON accessible from a URL. Two static symbolizers (line and polygon) are applied to the data and rendered in Web Mercator Projection using the layer rendering endpoint of the Semantic Portrayal Service. The rendered image was returned as PNG format.

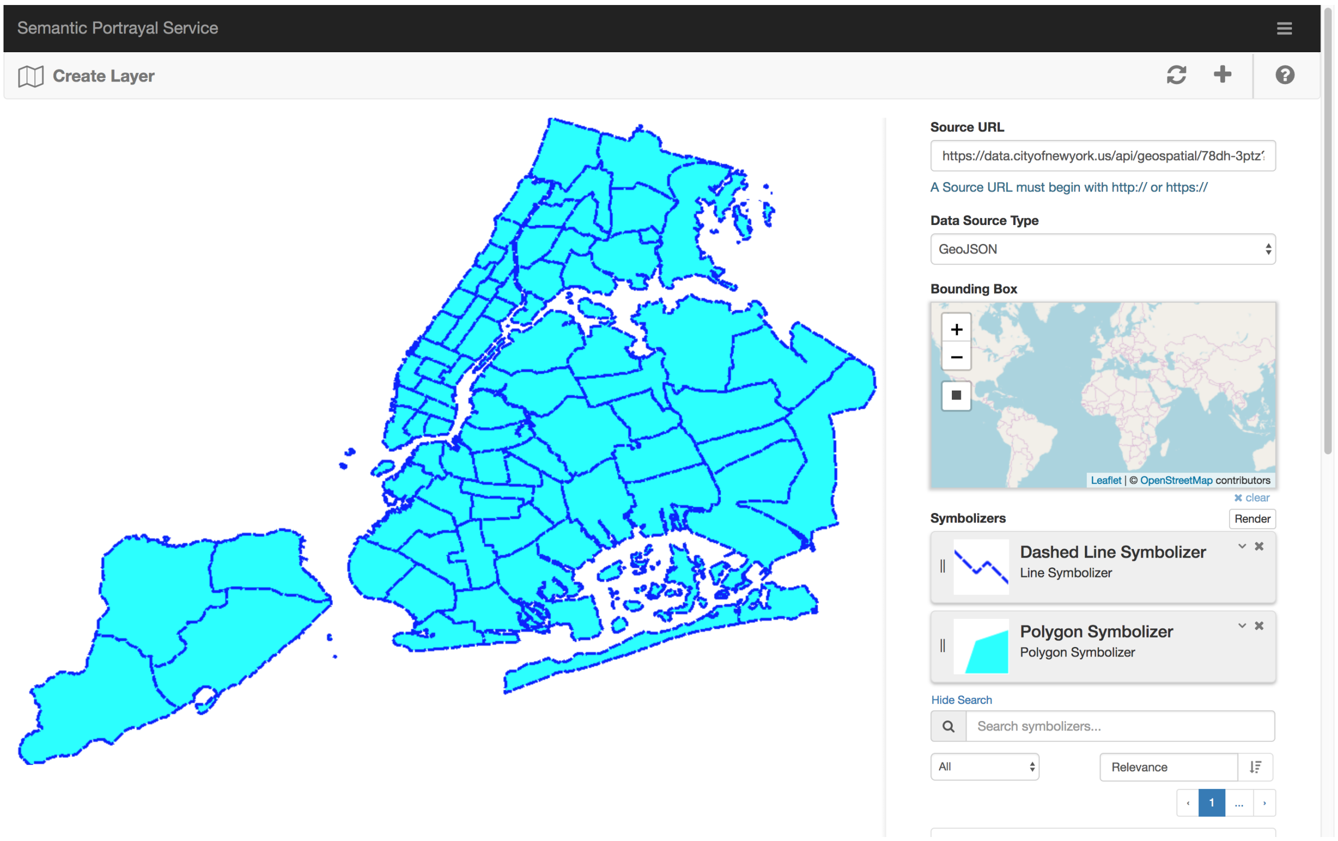
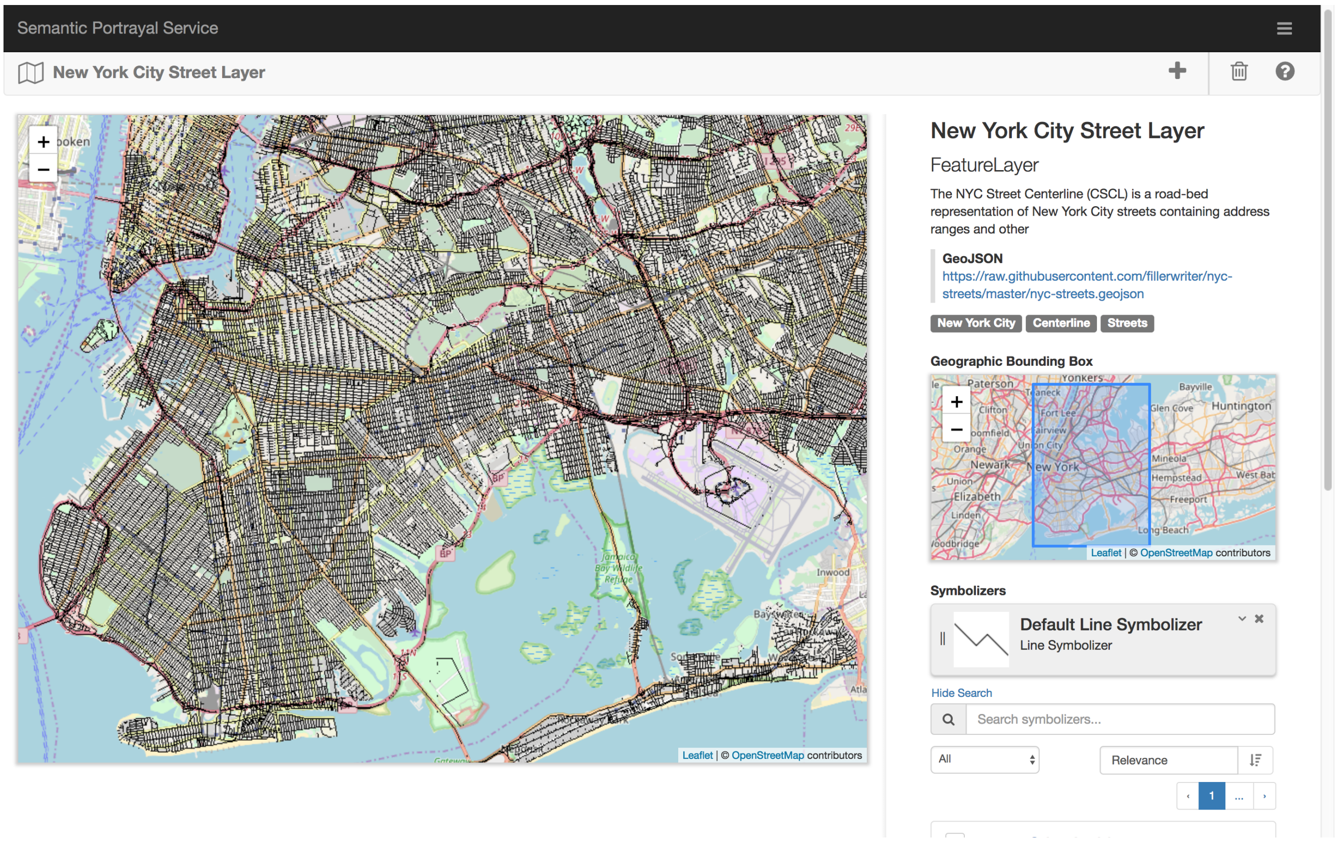


Figure 9. Layer Creation UI

8.5.2. Integration with Web Map Client

To integrate the layer managed by the portrayal service with existing web clients such as Leaflet, OpenLayers or MapBox,a rendering endpoint was implemented with the OGC GetMap operation. The integration with Leaflet was demonstrated (Figure 10).


Semantic Portrayal Service

New York City Street Layer

New York City Street Layer

FeatureLayer

The NYC Street Centerline (CSCL) is a road-bed representation of New York City streets containing address ranges and other

GeoJSON
https://raw.githubusercontent.com/fillerwriter/nyc-streets/master/nyc-streets.geojson

New York City | Centerline | Streets

Geographic Bounding Box

Symbolizers

Default Line Symbolizer
Line Symbolizer

Hide Search

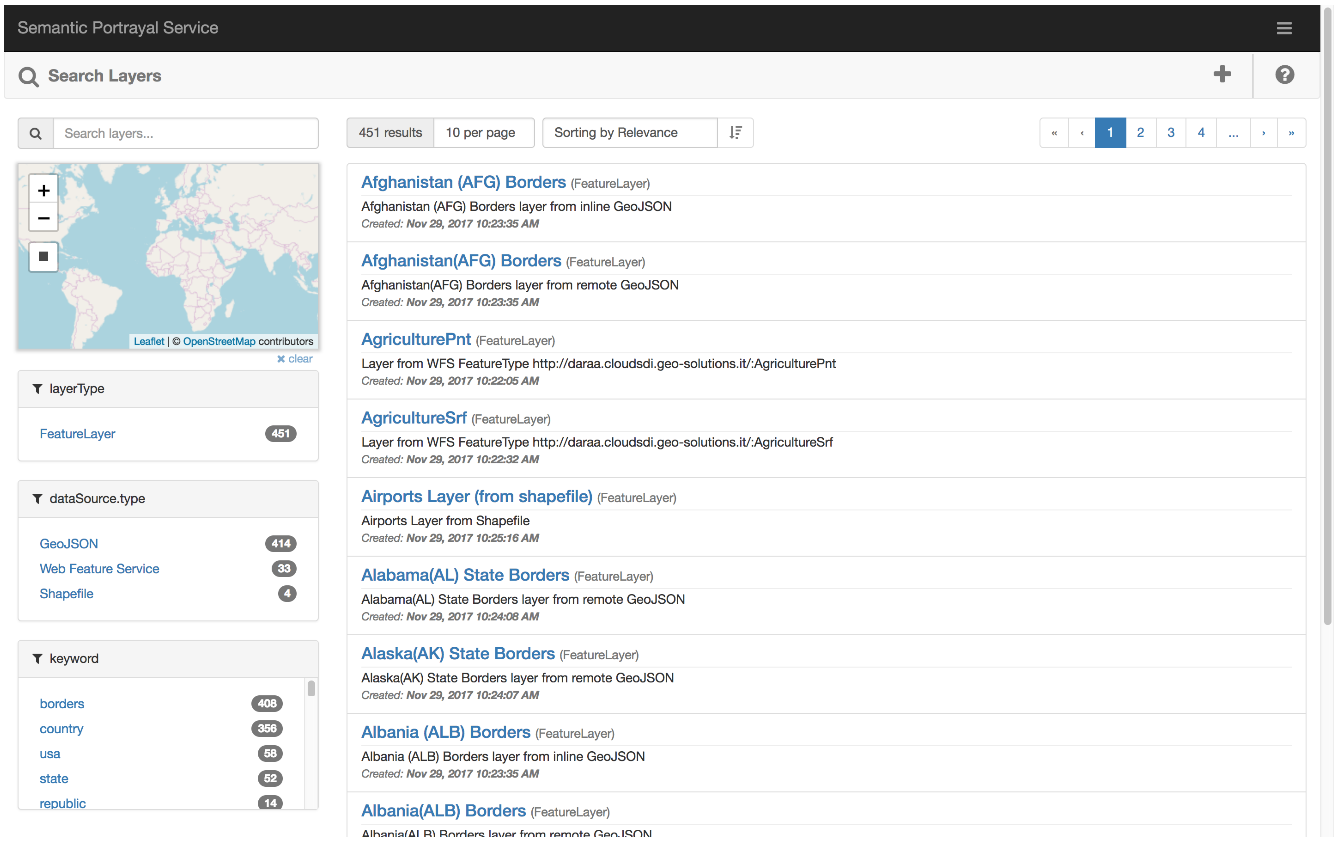
Search symbolizers...

All | Relevance

Figure 10. Leaflet Integration with Portrayal Service

8.5.3. Faceted Layer Search

To demonstrate the search capabilities for layers, a faceted search for layers was implemented in the portrayal web client. The client used the Layer Search REST API to perform search by free text, spatial bounding box, keyword, layer types, data sources types. The search results could be sorted by relevance, creation date, modified date or alphabetically in ascending or descending order. A summary of each layer was displayed with hyperlinks to the layer detail page (see Figure 11).


Semantic Portrayal Service
Search Layers
Search layers...
Leaflet | © OpenStreetMap contributors
✖ clear
« ‹ 1 2 3 4 ... › »

451 results | 10 per page | Sorting by Relevance

layerType

FeatureLayer	451

dataSource.type

GeoJSON	414
Web Feature Service	33
Shapefile	4

keyword

borders	408
country	356
usa	58
state	52
republic	14

Afghanistan (AFG) Borders (FeatureLayer)
Afghanistan (AFG) Borders layer from inline GeoJSON
*Created: **Nov 29, 2017 10:23:35 AM***

Afghanistan(AFG) Borders (FeatureLayer)
Afghanistan(AFG) Borders layer from remote GeoJSON
*Created: **Nov 29, 2017 10:23:35 AM***

AgriculturePnt (FeatureLayer)
Layer from WFS FeatureType http://daraa.cloudsdi.geo-solutions.it/:AgriculturePnt
*Created: **Nov 29, 2017 10:22:05 AM***

AgricultureSrf (FeatureLayer)
Layer from WFS FeatureType http://daraa.cloudsdi.geo-solutions.it/:AgricultureSrf
*Created: **Nov 29, 2017 10:22:32 AM***

Airports Layer (from shapefile) (FeatureLayer)
Airports Layer from Shapefile
*Created: **Nov 29, 2017 10:25:16 AM***

Alabama(AL) State Borders (FeatureLayer)
Alabama(AL) State Borders layer from remote GeoJSON
*Created: **Nov 29, 2017 10:24:08 AM***

Alaska(AK) State Borders (FeatureLayer)
Alaska(AK) State Borders layer from remote GeoJSON
*Created: **Nov 29, 2017 10:24:07 AM***

Albania (ALB) Borders (FeatureLayer)
Albania (ALB) Borders layer from inline GeoJSON
*Created: **Nov 29, 2017 10:23:35 AM***

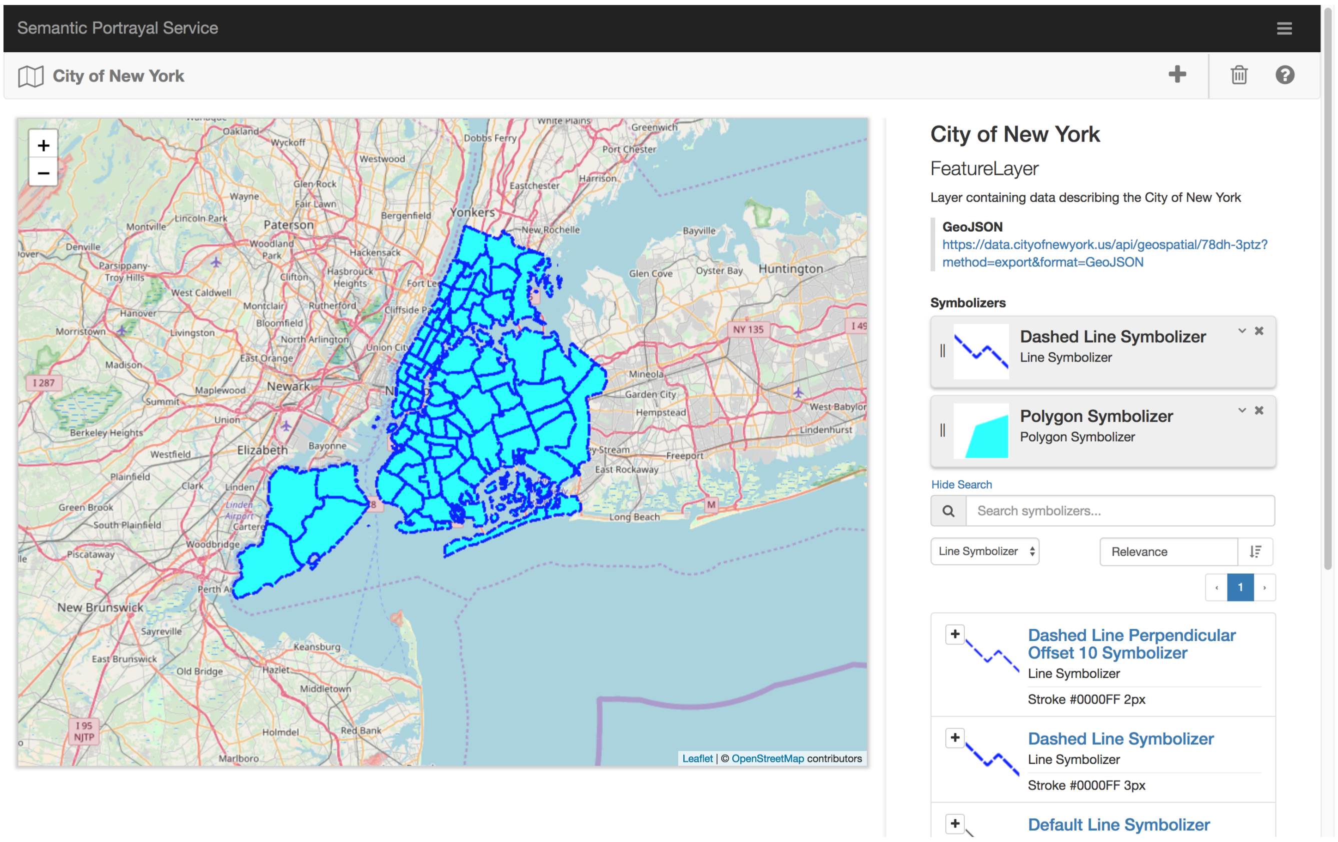
Albania(ALB) Borders (FeatureLayer)
Albania(ALB) Borders layer from remote GeoJSON

Figure 11. Faceted Layer Search

For future testbed efforts, it is suggested to work on the alignment with the SRIM Profile for Layer and Map used for search and discovery of layers in the Semantic Registry. Additional fields such as subject, theme, layer classification using semantic concepts may be useful to support better semantic search of layer information in the faceted search. There is also a need to investigate synchronization mechanisms between the layers managed by the Semantic Portrayal Service and the Semantic Registry.

8.5.4. Layer Detail Page

To demonstrate the rendering of layers and legend, a detailed page for a layer was implemented. It displays basic metadata about the layer, data source, the style information and legends. The graphic glyphs associated with each style were rendered on the server side using the symbolizer renderer endpoint. The layer display was reprojected and rendered using the server-side layer renderer endpoint implementing the WMS GetMap operation. Figure 12 illustrates the display of City of New York Police Precincts Layer with an OpenStreetMap base layer using Leaflet Web Map Client. The UI allows the user to modify the current style of the layer by providing a symbolizer search functionality and apply the symbolizer interactively using the server-side layer renderer. The customized layer could be cloned and saved by the client.


Semantic Portrayal Service
City of New York
+
−
Leaflet | © OpenStreetMap contributors

City of New York

FeatureLayer

Layer containing data describing the City of New York

GeoJSON
https://data.cityofnewyork.us/api/geospatial/78dh-3ptz?method=export&format=GeoJSON

Symbolizers

Dashed Line Symbolizer
Line Symbolizer

Polygon Symbolizer
Polygon Symbolizer

Hide Search

Search symbolizers...

Line Symbolizer | Relevance

Dashed Line Perpendicular Offset 10 Symbolizer
Line Symbolizer
Stroke #0000FF 2px

Dashed Line Symbolizer
Line Symbolizer
Stroke #0000FF 3px

Default Line Symbolizer

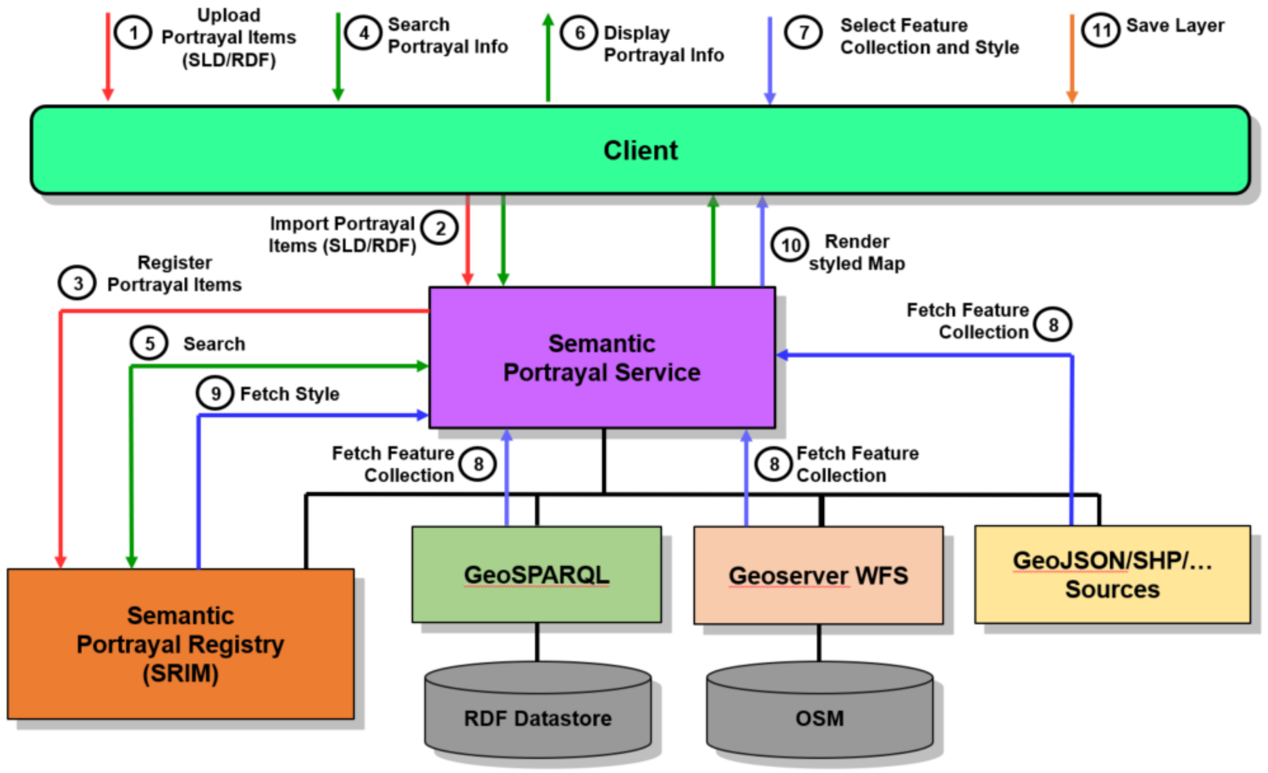
Figure 12. Layer Detail Page

More advanced styling capabilities will need to be investigated in future testbeds by testing the application of relevant portrayal rules that are based on feature data attributes. The handling of coverage/image layers and styles specific to coverage layers should also be investigated.

8.6. Demonstration Workflow

For the demonstration day, a workflow involving different data sources (WFS, Shapefile, GeoJSON, GeoSPARQL), Semantic Registry, Semantic Portrayal Service and portrayal web client, was defined to illustrate the different functionalities and interaction between the different services.

The workflow Figure 13 starts with the population of portrayal information by importing SLD documents and Linked Data representations of Portrayal Information (1). The Portrayal Service processes the portrayal information (2) and registers each portrayal artifact in the Semantic Registry implementing the Portrayal Profile (3) playing the role of a Portrayal Registry. Then the user performs a search of portrayal information (4) that is delegated to the semantic registry (5). Portrayal search results are presented to the user (6). Then the user creates a new layer by selecting a feature collection source and a style (7). The Semantic Portrayal Service retrieves the Feature collection from its source (8) and the style items from the registry (9) to be rendered as an image layer. The Portrayal Service performs the rendering of the layer by reprojecting and applying the styles to the feature data (10) and sends the rendered image to the client to display the layer to the user. The user then saves the layer in the portrayal Service for future retrieval.



1 Upload Portrayal Items (SLD/RDF)
4 Search Portrayal Info
6 Display Portrayal Info
7 Select Feature Collection and Style
11 Save Layer
Client
Import Portrayal Items (SLD/RDF) 2
10 Render styled Map
3 Register Portrayal Items
5 Search
9 Fetch Style
Semantic Portrayal Service
Fetch Feature Collection 8
Fetch Feature Collection 8
8 Fetch Feature Collection
Semantic Portrayal Registry (SRIM)
GeoSPARQL
Geoserver WFS
GeoJSON/SHP/… Sources
RDF Datastore
OSM

Figure 13. Demonstration Workflow Search

For the demonstration, sources encoded in GeoJSON and zipped Shapefiles were used, as well as the Web Feature Service of GeoSolutions used for the Mass Migration Thread. Due to time constraints, the integration of a GeoSPARQL endpoint was not tested. This should be addressed in future testbeds to exercise the Portrayal ontologies further and offer the opportunity to perform map rendering from a vast source of available geospatial Linked Data (such as DBPedia, Geonames).

Appendix A: Semantic Portrayal Ontologies

A.1 Overview

The Portrayal Ontologies specify a conceptual model for portrayal data, in particular symbols and portrayal rules. Portrayal rules associate features with symbols for the portrayal of the features on maps and other display media. These ontologies include classes, attributes and associations that provide a common conceptual framework that specifies the structure of and interrelationships between feature types, portrayal rules, symbols and symbolizers. It separates the content of the data from the portrayal of that data to allow the data to be portrayed in a manner independent of the dataset model. The graphic description used for symbolizers is intended to be format independent but convertible to any target formats (SVG, KML). The ontologies are derived from concepts found in existing portrayal specifications (ISO 19117, OGC Symbology Encoding, Styled Layer Descriptor Profile of WMS, SVG, KML, CartoCSS, MapCSS).

For the Testbed 13 initiative, four new ontology modules were introduced to represent legends, layers, expressions and bindings. Other ontologies defined in previous testbeds were refined to better align with OGC SLD and support round-trip conversion between the semantic representation of portrayal information and SLD.

To favor reusability, the Portrayal ontologies are decomposed into a number of reusable ontology modules (Figure 14), each one addressing an orthogonal aspect of portrayal:

- **Style ontology**: defines the concept of Style and portrayal rules.
- **Legend ontology (new)**: defines legend and legend item descriptions.
- **Symbol ontology**: defines the concept of SymbolSet and Symbol and structural definition of Symbol components.
- **Symbolizer ontology**: defines the concepts of Symbolizers defining instructions to render data as graphics.
- **Graphic ontology**: defines graphic elements including graphic objects and attributes.
- **Binding ontology (new)**: defines binding of data properties to expression associated to graphic properties
- **Expression ontology (new)**: defines an extensible ontology for representing expression for different data model and different expression languages such as SPARQL, OGC Filter and the Rule Interchange Format (RIF) for example.
- **Layer ontology (new)**: defines map layers including data sources and styling information (Initial version as developed at the end of the Testbed)

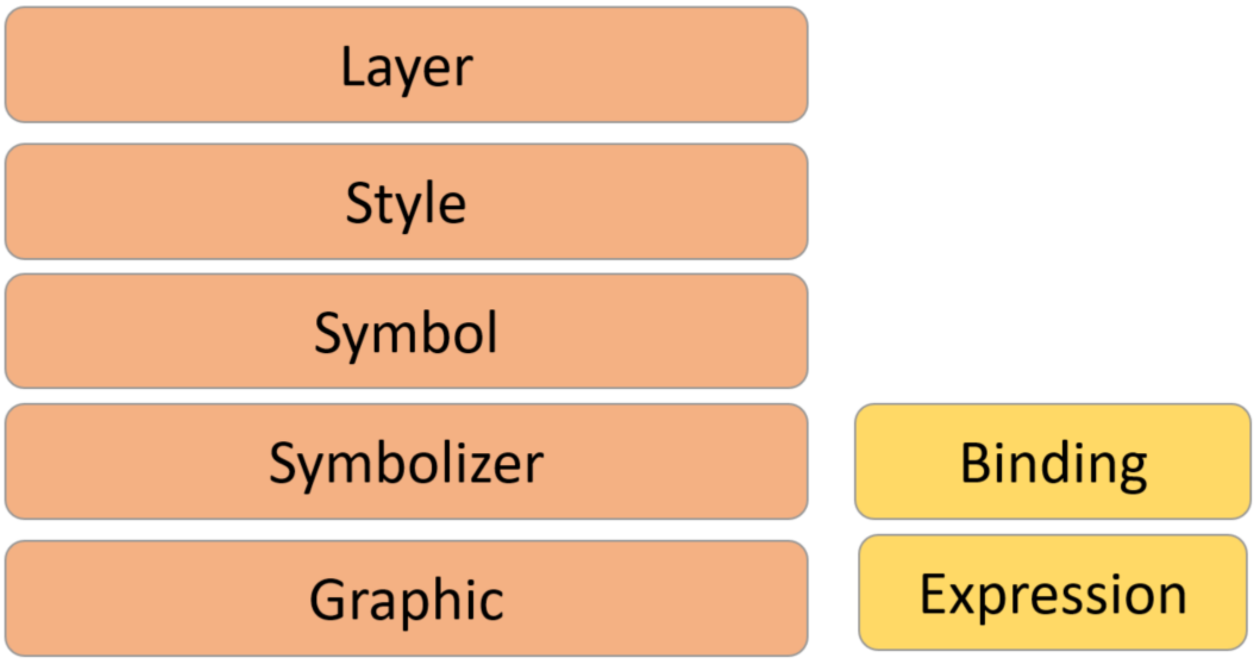


Figure 14. Portrayal ontology modules

A.1.1 Namespaces

The Portrayal ontology module specifications are based on a formal semantic model (ontology) that uses a number of well-established ontologies. Each ontology itself defines a minimal set of classes and properties of its own. The full set of namespaces and prefixes used in this document is shown in the table below.

Table 2. Table 3. Namespaces

Prefix	Namespace URI	Schema & Documentation
style	http://www.opengis.net/ont/portrayal/style#	N/A
symbol	http://www.opengis.net/ont/portrayal/symbol#	N/A
symbolizer	http://www.opengis.net/ont/portrayal/symbolizer#	N/A
graphic	http://www.opengis.net/ont/portrayal/graphic#	N/A
legend	http://www.opengis.net/ont/portrayal/legend/	N/A
layer	http://www.opengis.net/ont/portrayal/layer	N/A
expression	http://www.opengis.net/ont/expression/	N/A
binding	http://www.opengis.net/ont/binding#	N/A

dcat	http://www.w3.org/ns/dcat#	Data Catalog Vocabulary [http://www.w3.org/TR/2014/REC-vocab-dcat-20140116/]
dct	http://purl.org/dc/terms/	DCMI Metadata Terms [http://dublincore.org/documents/2012/06/14/dcmi-terms/]
extent	http://www.opengis.net/ont/spatial/extent	N/A
geosparql	http://www.opengis.net/ont/geosparql	GeoSPARQL [http://www.opengeospatial.org/standards/geosparql]
prov	http://www.w3.org/ns/prov#	PROV-O: The PROV Ontology [http://www.w3.org/TR/2013/REC-prov-o-20130430/]
rdf	http://www.w3.org/1999/02/22-rdf-syntax-ns#	Resource Description Framework (RDF): Concepts and Abstract Syntax [http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/]
rdfs	http://www.w3.org/2000/01/rdf-schema#	RDF Vocabulary Description Language 1.0: RDF Schema [http://www.w3.org/TR/2004/REC-rdf-schema-20040210/]
srim	http://www.geoplatform.gov/ont/srim#	
schema	http://schema.org/	Schema Vocabulary [http://schema.org/]
skos	http://www.w3.org/2004/02/skos/core#	SKOS Simple Knowledge Organization System - Reference [http://www.w3.org/TR/2009/REC-skos-reference-20090818/]
xsd	http://www.w3.org/2001/XMLSchema#	XML Schema Part 2: Datatypes Second Edition [http://www.w3.org/TR/2004/REC-xmlschema-2-20041028/]

A.2 Ontologies

A.2.1 Style Ontology

The Style Ontology defines concepts of Style, Portrayal Rule Set, Portrayal Rule, Rule Condition and PortrayalContext. [style_ontolology_image] below shows an overview of the model in UML.

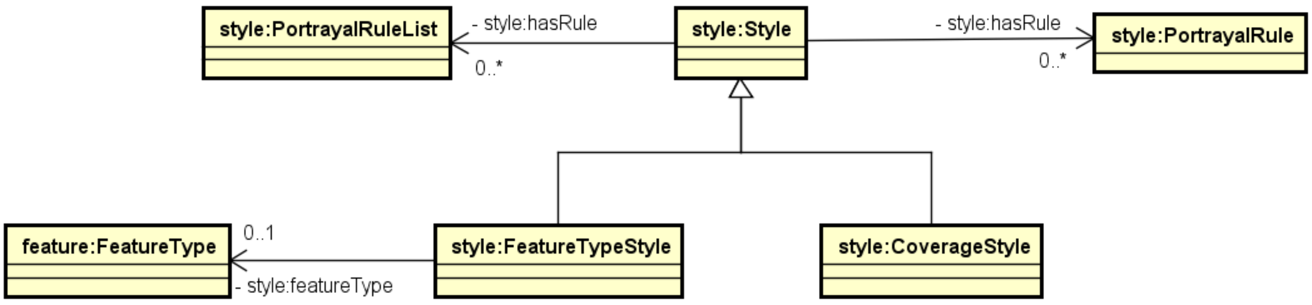


Figure 15. Style Model Overview

A.2.1.1 Style

The **Style** concept is the central concept of the Style ontology. It associates symbol sets with portrayal rule sets, which define the mapping of feature types to symbols. The Style also captures descriptive metadata and tradecraft information such as the audience for the style, scope of application and field of application.The following table summarizes the properties of the Style class.

Table 3. Properties of the Style Class

Property	Usage Note	Range	Multiplicity and use
name	Name identifier of the style.	String	One
dct:title	Multilingual human-readable title for the style		1..n (one per language)
dct:description	Multilingual human-readable description for the style	String	1..n (one per language)
hasRule	PortrayalRule or ordered PortrayalRuleList associated with the Style.	PortrayalRule or PortrayalRuleList	1..n
dct:audience	The intended audience of this style.	foaf:Group	0..n
scope	Descriptive definition of the scope of application of the style	String	0..1
language	Language associated with the style	String	0..n
style:symbolSet	SymbolSet associated with the style	SymbolSet	0..n

Property	Usage Note	Range	Multiplicity and use
fieldOfApplication	The field of application of this style, where values are defined as SKOS concept in a taxonomy.	skos:Concept	0..n

A.2.1.2 FeatureTypeStyle

A **FeatureTypeStyle** is a style that is applied for a specific **feature:FeatureType**. **FeatureTypeStyle** inherits all properties from **Style**.

Table 4. Properties of the Style Class

Property	Usage Note	Range	Multiplicity and use
name	Name identifier of the style.	String	One
dct:title	Multilingual human-readable title for the style	String	1..n (one per language)
dct:description	Multilingual human-readable description for the style	String	1..n (one per language)
featureType	FeatureType associated with the style	String	1..n
hasRule	PortrayalRule or ordered PortrayalRuleList associated with the Style.	PortrayalRule or PortrayalRuleList	1..n

The following example shows the definition of the EMS Style with audience information organized as a hierarchy.

EMS Style with Audience Information

```
@prefix :      <http://www.opengis.net/testbed/12/portrayal/ems/style#> .
@prefix feature: <http://www.opengis.net/ont/feature#> .
@prefix dct:   <http://purl.org/dc/terms/> .
@prefix xsd:   <http://www.w3.org/2001/XMLSchema#> .
@prefix rdfs:  <http://www.w3.org/2000/01/rdf-schema#> .
@prefix ems:   <http://www.opengis.net/testbed11/ont/incident/ems#> .
@prefix style: <http://www.opengis.net/ont/portrayal/style#> .
@prefix foaf:  <http://xmlns.com/foaf/0.1/> .
@prefix group: <http://www.socialml.org/ontologies/group#> .

:EMSIncidentStyle  a       style:FeatureTypeStyle ;
        dct:audience
<http://ows.usersmarts.com/ldapp/audiences/community/CanadianEmergencyAndDisasterManag
ement> ;
        dct:description    "Style defining the set of rules for mapping incident types
from EMS to symbology" ;
        dct:title          "EMS Incident Type Style" ;
        style:featureType  ems:EMSIncident ;
        style:hasRule      :ems.incident.hazardousMaterial.radiologicalHazard-
portrayal-rule , :ems.incident.temperature.windChill-portrayal-rule,
:ems.incident.health-portrayal-rule , :ems.incident.hazardousMaterial-portrayal-rule
.

ems:EMSIncident  a       feature:FeatureType ;
        rdfs:comment     "Incident defined for Canadian Emergency Management System" ;
        rdfs:label       "EMSIncident" ;
        feature:gmlName  "ems:EMSIncident" .

<http://ows.usersmarts.com/ldapp/audiences/community/EmergencyAndDisasterManagement>
        a           foaf:Group , group:Group ;
        rdfs:label  "Emergency and Disaster Management Community" ;
        foaf:name   "Emergency and Disaster Management Community" .
```

A.2.1.3 CoverageStyle

A **CoverageStyle** is a style that is applied for a Coverage. **CoverageStyle** inherits all properties from **Style**. This concept is introduced as a placeholder for future extension.

A.2.1.4 PortrayalRuleSet

A **portrayal rule set** describes a function set which maps the feature types of a feature catalog to a symbol. It is composed of one or more portrayal rules, which in turn maps an individual feature type to a symbol. The table below provides a summary of the **PortrayalRuleSet**.

Table 5. Properties of the PortrayalRuleSet Class

Property	Usage Note	Range	Multiplicity and use
dct:title	Multilingual human-readable name for the PortrayalRuleSet	string	0..n (one per language)
dct:description	Multilingual human-readable description for the PortrayalRuleSet	string	0..n (one per language)
hasRule	PortrayalRule member of this PortrayalRuleSet.	Portrayal Rule	0..n

The following is a sample of the PortrayalRuleSet defined for the EMS Style.

EMS PortrayalRuleSet Example

```
:EMSRuleSet  a           style:PortrayalRuleSet ;
        dct:description  "Set of rules for mapping incident types from EMS to
symbology" ;
        dct:title        "EMS Portrayal Rule Set" ;
        style:hasRule   :ems.incident.temperature.windChill-portrayal-rule ,
                        :ems.incident.roadway.hazardousRoadConditions-portrayal-rule ,
                    :ems.incident.civil-portrayal-rule ,
                    :ems.incident.roadway.trafficReport-portrayal-rule ,
                    :ems.incident.meteorological.waterspout-portrayal-rule ,
                    :ems.incident.geophysical.lahar-portrayal-rule ,
                    :ems.incident.meteorological-portrayal-rule ,
                    :ems.incident.meteorological.stormSurge-portrayal-rule ,
                    :ems.incident.aviation-portrayal-rule ,
                    :ems.incident.hazardousMaterial.radiologicalHazard-portrayal-rule
,
                    :ems.incident.crime.bomb-portrayal-rule .
```

A.2.1.5 PortrayalRule

A **PortrayalRule** defines a rule that associates a feature type (**feature:FeatureType**) to a symbol (**symbol:Symbol**) satisfying a certain condition (**PortrayalRuleCondition**) in a given context (**PortrayalContext**). The Table below summarizes its properties.

Table 6. Properties of the PortrayalRule

Property	Usage Note	Range	Multiplicity and use
name	Name of the PortrayalRule	string	0..1
dct:title	Multilingual human-readable title for the PortrayalRule	string	0..n (one per language)
dct:description	Multilingual human readable description for the PortrayalRule.	string	0..n (one per language)
hasCondition	The conditions that needs to be satisfied by the rule	RuleCondition	0..n

Property	Usage Note	Range	Multiplicity and use
maxScaleDenominator	The maximum scale denominator to which the rule applies	integer	0..1
minScaleDenominator	The minimum scale denominator to which the rule applies	integer	0..1
portrayalContext	The context of application of the PortrayalRule (placeholder for future extension)	Portrayal Context	0..n
symbol	The symbol associated with the rule	symbol:Symbol	1

The listing below shows an example of **PortrayalRule** for Windchill. The PortrayalRule applied on the ems:EMSIncident featureType and associates the EMS symbol for Windchill defined by the URL: http://www.opengis.net/testbed/11/cci/ems/symbols#ems.incident.temperature.windChill-symbol [http://www.opengis.net/testbed/11/cci/ems/symbols#ems.incident.temperature.windChill-symbol]

EMS PortrayalRule Example

```
:ems.incident.temperature.windChill-portrayal-rule
        a     style:PortrayalRule ;
        dct:description  "Portrayal rule for incident type
ems.incident.temperature.windChill" ;
        dct:title  "Wind Chill incident portrayal rule" ;
        style:hasRuleCondition :ems.incident.temperature.windChill-portrayal-rule-
condition;
        style:maxScaleDenominator  "100000.0"^^xsd:double ;
        style:minScaleDenominator  "0.0"^^xsd:double ;
        style:symbol
<http://www.opengis.net/testbed/12/ems/symbols#ems.incident.temperature.windChill-
symbol> .
```

A.2.1.6 RuleCondition

The **RuleCondition** defines the condition in which a portrayal rule applies for a given feature. The **RuleCondition** can be encoded using multiple encodings. The Table below summarizes the properties of PortrayalRuleCondition.

Table 7. Properties of the RuleCondition

Property	Usage Note	Range	Multiplicity and use
dct:title	Multilingual human-readable name for the RuleCondition	string	0..n (one per language)
dct:description	Multilingual human-readable description for the RuleCondition	string	0..n (one per language)

Property	Usage Note	Range	Multiplicity and use
hasConstraint	The encoding of the constraint defining the condition	Constraint	0..n

To provide an extensible mechanism to express constraints for the condition, we modified the previous ontology in Testbed 11, which uses specialized properties (sparqlCondition, rifCondition,ogcFiterCondition)

Table 8. Properties of the Constraint

Property	Usage Note	Range	Multiplicity and use
constraintLanguage	The constraint language used to encode the constraint. The following constraint languages URLs are currently recommended: • SPARQL: http://www.w3.org/ns/sparql-service-description#SPARQL11Query • OGC Filter: http://schemas.opengis.net/filter • RIF : http://www.w3.org/2007/rif • CQL : http://www.opengis.net/specs/cql	URL	1
body	The constraint body expressed in the constraint language	String	1

The following example demonstrates the encoding of the portrayal rule condition for the Portrayal rule for the symbol Windchill. The condition applies on the feature property ems:incidentType. If the value of this property is equals to http://www.opengis.net/taxonomy/ems#ems.incident.temperature.windChill [http://www.opengis.net/taxonomy/ems#ems.incident.temperature.windChill] then the rule is applicable.

EMS RuleCondition Example

```
:ems.incident.temperature.windChill-portrayal-rule-condition
    a style:RuleCondition ;
    style:hasConstraint
    [   style:Constraint ;
        style:body  "PREFIX ems:
<http://www.opengis.net/testbed11/ont/incident/ems#>\nASK \nWHERE { ?this
ems:incidentType
<http://www.opengis.net/taxonomy/ems#ems.incident.temperature.windChill>.\n}" ;
        style:constraintLanguage  <http://www.w3.org/ns/sparql-service-
description#SPARQL11Query>
    ] ;
    style:hasConstraint
    [   a   style:Constraint ;
        style:body
"<ogc:Filter><ogc:PropertyIsEqualTo><ogc:PropertyName>incidentType</ogc:PropertyName><
ogc:Literal>http://www.opengis.net/taxonomy/ems#ems.incident.temperature.windChill</og
c:Literal></ogc:PropertyIsEqualTo></ogc:Filter>" ;
        style:constraintLanguage  <http://schemas.opengis.net/filter>
    ] ;
    style:hasConstraint
    [   a   style:Constraint ;
        style:body  "Prefix(ems
<http://www.opengis.net/testbed11/ont/incident/ems#>)\nExists ?this (
ems:incidentType(?this
<http://www.opengis.net/taxonomy/ems#ems.incident.temperature.windChill>) )" ;
        style:constraintLanguage  <http://www.w3.org/2007/rif>
    ]
```

The rule is expressed in three different encodings: OGC Filter, SPARQL and RIF.

The SPARQL query is formulated as an ASK query which returns a boolean value. The variable ?this is bound to the current instance of featureType that is being tested.

```
PREFIX ems: <http://www.opengis.net/testbed11/ont/incident/ems#>
ASK
WHERE {
      ?this ems:incidentType

<http://www.opengis.net/taxonomy/ems#ems.incident.temperature.windChill>.
}
```

The equivalent RIF condition is expressed as:

```
Prefix(ems <http://www.opengis.net/testbed11/ont/incident/ems#>)
Exists ?this
 ( ems:incidentType(?this
<http://www.opengis.net/taxonomy/ems#ems.incident.temperature.windChill>) )
```

The Constraints expressed in SPARQL and RIF can be used by a semantic portrayal rule engine that consumes feature data represented as Linked Data (recommendation for next testbed). We foresee that the portrayal catalog containing these rules can be extended with a rendering service that will apply the rules on a set of linked data compatible with the style and generates the portrayal rendering to a number of target formats (SVG, KML etc..). Future testbeds should experiment with this capability.

To perform the bridge between the Linked Data representation of the FeatureType and GML representation we annotated the FeatureType and FeatureProperty with the attribute gmlName to indicate what is the mapping between the conceptual definition and the GML syntactic definition based on XML schema.

The following example shows the feature type definition for EMSIncident with the gmlName annotations.

```
ems:EMSIncident  a       feature:FeatureType ;
        rdfs:comment     "Incident defined for Canadian Emergency Management System" ;
        rdfs:label       "EMSIncident" ;
        feature:gmlName  "ems:EMSIncident" .

ems:incidentType  a      feature:FeatureProperty ;
        rdfs:label       "incidentType" ;
        feature:gmlName  "ems:incidentType" .
```

The following example shows the feature type definition for HSWGIncident with the gmlName annotations.

```
hswg:HSWGIncident  a     feature:FeatureType ;
        rdfs:comment     "Incident defined for Homeland Security Working Group" ;
        rdfs:label       "HSWGIncident" ;
        feature:gmlName  "hswg:HSWGIncident" .

hswg:incidentType  a     feature:FeatureProperty ;
        rdfs:label       "incidentType" ;
        feature:gmlName  "hswg:incidentType" .
```

A.2.1.7 PortrayalRuleList and RuleItem

In some instance, rules need to be executed in a given order with support of fallback rules.The **PortrayalRuleList** defines an ordered list of **RuleItem** instances. The PortrayalRuleList is implemented by an **rdf:List**.

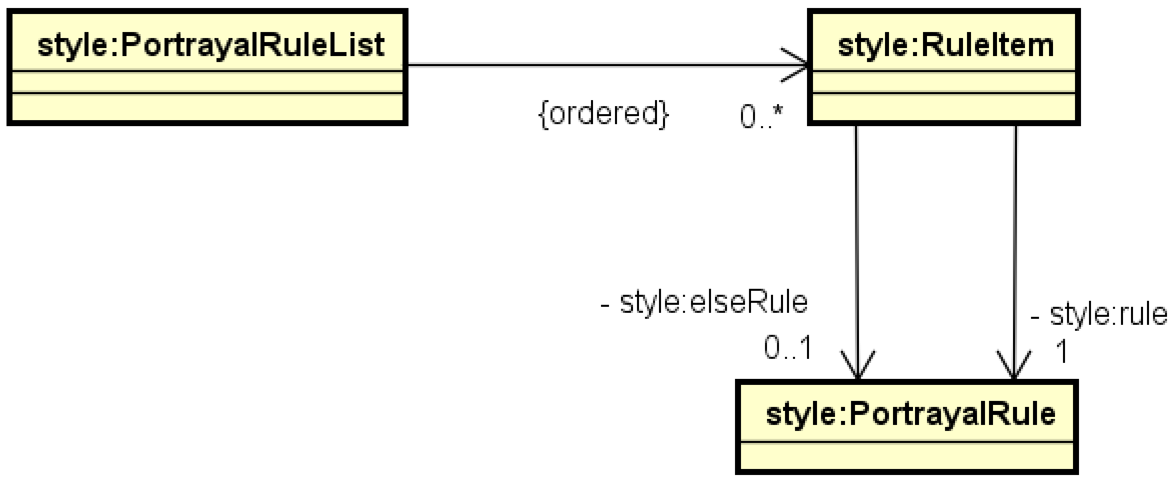


Figure 16. PortrayalRuleList

The **RuleItem** defines a placeholder referring indirectly to a portrayal rule. However it can also provide a reference to fallback portrayal rules (elseRule) if the rule is not applicable. The RuleItem is used when the order of application of the rules is important. The RuleItem instances are used as elements of **PortrayalRuleList**.

*Table 9. Properties of the RuleItem***

Property	**Usage Note**	**Range**	**Multiplicity and use**
rule	The rule associated with the rule item	Portrayal Rule	1

A.2.2 Legend Ontology

The Legend Ontology defines concepts of Legend and LegendItem. The following figure shows an overview of the model in UML (we use the namespace prefix olegend for Open Legend).

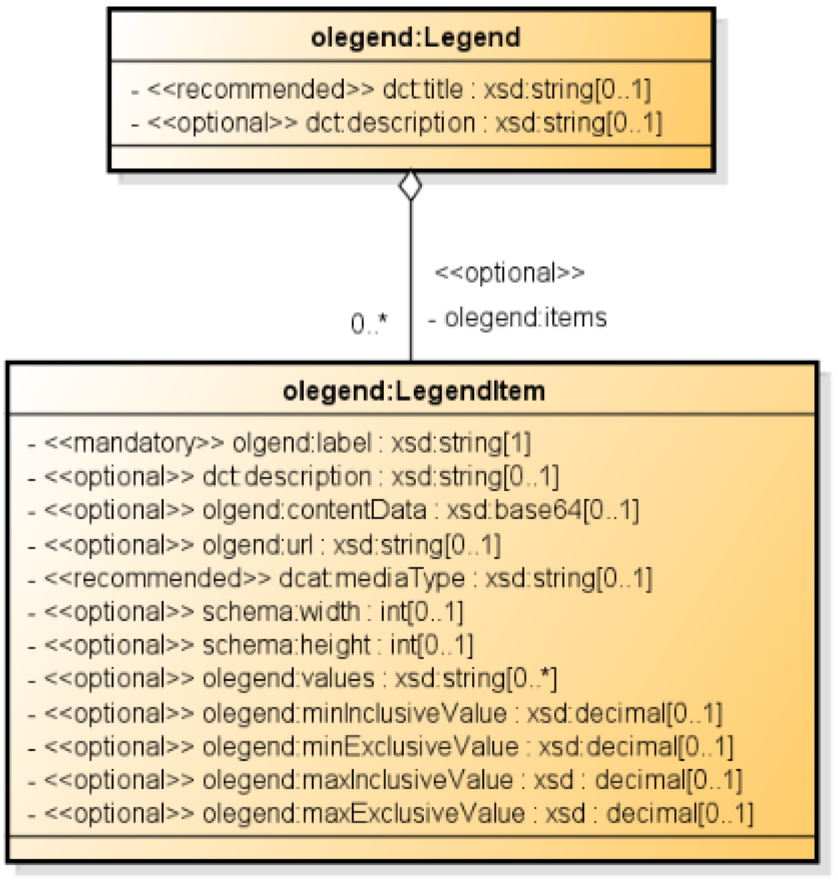

olegend:Legend
- <<recommended>> dct:title : xsd:string[0..1]
- <<optional>> dct:description : xsd:string[0..1]
<<optional>>
- olegend:items
0..*
olegend:LegendItem
- <<mandatory>> olgend:label : xsd:string[1]
- <<optional>> dct:description : xsd:string[0..1]
- <<optional>> olgend:contentData : xsd:base64[0..1]
- <<optional>> olgend:url : xsd:string[0..1]
- <<recommended>> dcat:mediaType : xsd:string[0..1]
- <<optional>> schema:width : int[0..1]
- <<optional>> schema:height : int[0..1]
- <<optional>> olegend:values : xsd:string[0..*]
- <<optional>> olegend:minInclusiveValue : xsd:decimal[0..1]
- <<optional>> olegend:minExclusiveValue : xsd:decimal[0..1]
- <<optional>> olegend:maxInclusiveValue : xsd : decimal[0..1]
- <<optional>> olegend:maxExclusiveValue : xsd : decimal[0..1]

Figure 17. Legend Model Overview

A **Legend** plays a central role in the understanding of the meaning of a map or layer. It associates symbols used in a **style** with its intended denotation (meaning). This meaning is often associated with a human readable label but could also be associated with a machine processable concept. A layer can have multiple layer styles. For each style, there is a legend associated with it.

NOTE We choose to define the Legend Object in a separate micro-ontology (a.k.a. microtheory) as it is a reusable concept that could be used to associated a legend to a map, layer or a style.

A.2.2.1 Legend Object

The primary objective of the Legend (**legend:Legend**) object is to be presented and be read by end users. Most of the time, legends are encoded as image that could be retrieved from a remote endpoint or inline using base64 encoding. When legends are very rich in symbols, they tend to produce image that are disproportionally long to display on a screen. To work around this problem, legends can be broken down into individual items (**LegendItem**) that typically associate one symbol with one item. The client has then the flexibility to layout the legend the best way using the space constraints of the target device screen.

Term	Mapping	Definition	Range	Card.
title	dct:title [http://dublincore.org/documents/dcmi-terms/#terms-title]	The title of a legend	string	0..1
description	dct:description [http://dublincore.org/documents/dcmi-terms/#terms-description]	The description of a legend	string	0..1
items	legend:items	A legend can be encoded as one item or be decomposed into more fined grained items (typically mapping one symbol per item)	legend:LegendItem	0..n

A.2.2.2 Legend Item

The legend item (legend:LegendItem) can be used to decompose the items of a given layer legend, or could also be used as the place holder for the legend for the whole layer or even a set of layers (in the case of map). The drawback of the second approach is the client has less control of the layout of the legend and a large legend tends to be difficult to be managed for display.

Term	Mapping	description	Range	Card.
label	rdfs:label [https://www.w3.org/TR/rdf-schema/#ch_label]	The label of a legend item	string	0..1
description	dct:description [http://dublincore.org/documents/dcmi-terms/#terms-description]	The description of a legend item	string	1
url	legend:url	the remote resource url to access the legend content	url	0..1
contentData	legend:contentData	Base64 encoding of the content of the legend	base64 string	0..1

Term	Mapping	description	Range	Card.
mediaType	dcat:mediaType [https://www.w3.org/TR/vocab-dcat/#Property:distribution_media_type]	the media type of the content of the legend item	string	0..1
values	legend:value	data values associated with the legend	string	0..n
minInclusiveValue	legend:minInclusiveValue	min inclusive classbreak value associated with the legend item	decimal	0..1
minExclusiveValue	legend:minExclusiveValue	min exclusive classbreak value associated with the legend item	decimal	0..1
maxInclusiveValue	legend:maxInclusiveValue	max inclusive classbreak value associated with the legend item	decimal	0..1
maxExclusiveValue	legend:maxExclusiveValue	max exclusive classbreak value associated with the legend item	decimal	0..1

A.2.3 Symbology Ontology

The Symbology Ontology is a microtheory that defines the conceptual model for defining **SymbolSet** and **Symbol** with their structural components called **Symbolizer**.

The figure below shows an overview of the model.

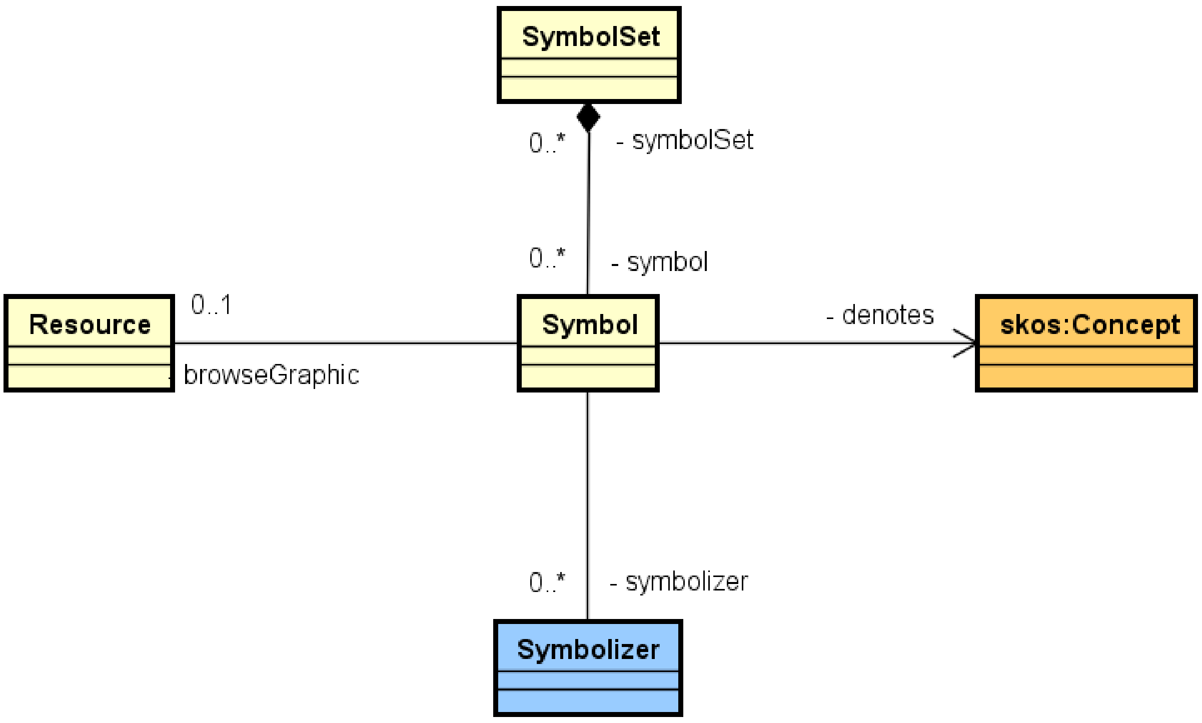


Figure 18. Symbology Model Overview

A.2.3.1 SymbolSet

SymbolSet collects symbols into sets of symbols that are used together. Symbols can be shared among symbol sets. A Symbol set can be equated with a legend of a map. The Table below summarizes the **SymbolSet** properties.

Table 10. Properties of the SymbolSet Class

Property	Usage Note	Range	Multiplicity and use
dct:identifier	Unique identifier for the symbol set mainly used by machine.	string	One
dct:title	Multilingual human-readable title for the SymbolSet	string	0..n (one per language)
dct:description	Multilingual human-readable description for the SymbolSet	string	0..n (one per language)
specification	Cites the specification standard for the SymbolSet	Resource	0..1
symbol	Symbol member of this SymbolSet	Symbol	0..n

The following example shows a sample of the EMS SymbolSet.

EMS SymbolSet Example

```
@prefix :       <http://www.opengis.net/testbed/11/cci/ems/symbols#> .
@prefix rdfs:  <http://www.w3.org/2000/01/rdf-schema#> .
@prefix graphic: <http://www.opengis.net/ont/portrayal/graphic#> .
@prefix symbol: <http://www.opengis.net/ont/portrayal/symbol#> .
@prefix dct:   <http://purl.org/dc/terms/> .
@prefix skos:  <http://www.w3.org/2004/02/skos/core#> .

:EMSSymbolSet  a              symbol:SymbolSet ;
        dct:description       "Standard Canadian Emergency Mapping Symbology (EMS)
SymbolSet version 1.0" ;
        dct:title             "Canadian Emergency Mapping Symbology (EMS) SymbolSet
(version 1.0)" ;
        symbol:symbol      :ems.incident.roadway.roadwayClosure-symbol ,
:ems.incident.temperature.windChill-symbol , :ems.incident.temperature.heatWave-symbol
, :ems.incident.civil.looting-symbol , :ems.incident.civil.dignitaryVisit-symbol ,
:ems.incident.civil.displacedPopulations-symbol , :ems.incident.publicService-symbol ;
   symbol:specification  <https://cms.masas-
x.ca.s3.amazonaws.com/EMS_Symbology_v1.0.pdf> .
```

A.2.3.2 Symbol

A Symbol is the type used to define symbol classes. Symbols are collected into symbol sets. A symbol has one machine readable identifier. It is described by a title, description and can refer to a formal specification document. A symbol can denote a concept defined in a SKOS taxonomy. The table below summarizes the Symbol properties.

Table 11. Properties of the Symbol Class

Property	Usage Note	Range	Multiplicity and use
dct:identifier	Machine readable name for the symbol. The identifier should be unique	string	1
dct:title	Multilingual human-readable title for the Symbol	string	0..n (one per language)
dct:description	Multilingual human-readable description for the Symbol	string	0..n (one per language)
specification	Reference to the full details of the portrayal symbol	Resource	0..1
browseGraphic	Reference a graphic representing the symbol	Resource	0..1
denotes	Concept that is denoted by the symbol	skos:Concept	0..n
symbolizer:symbolizer	Defines the symbolizer rendering the symbol	symbolizer:Symbolizer	0..n

Property	Usage Note	Range	Multiplicity and use
symbolSet	The SymbolSet which this symbol belongs to.	SymbolSet	0..n
skos:notation	Notation used to refer the symbol as defined in a notation system. Use a custom datatype if multiple notations are used	string	0..n

The following listing shows the encoding in Turtle for the WindChill symbol belonging to the EMS SymbolSet.

EMS Symbol Example

```
:ems.incident.roadway.roadwayClosure-symbol
        a                       symbol:Symbol ;
        dct:title               "roadwayClosure" ;
        symbol:browseGraphic
<http://ows.usersmarts.com/ems/icons/tier1/Base/ems.incident.roadway.roadwayClosure.pn
g> ;
        symbol:denotes
<http://www.opengis.net/taxonomy/ems#ems.incident.roadway.roadwayClosure> ;
        symbol:specification   <https://cms.masas-
x.ca.s3.amazonaws.com/EMS_Symbology_v1.0.pdf> ;
        symbol:symbolName      "ems.incident.roadway.roadwayClosure" ;
        symbol:symbolSet       :EMSSymbolSet ;
        symbolizer:symbolizer  :ems.incident.roadway.roadwayClosure-pointSymbolizer .

:ems.incident.roadway.roadwayClosure-pointSymbolizer
        a                       symbolizer:PointSymbolizer ;
        dct:title               "PointSymbolizer for roadwayClosure symbol" ;
        graphic:graphicSymbol  [ a                        graphic:GraphicSymbol ;
                                  graphic:externalGraphic
<http://ows.usersmarts.com/ems/icons/tier1/Base/ems.incident.roadway.roadwayClosure.pn
g>
                                ] .

<http://ows.usersmarts.com/ems/icons/tier1/Base/ems.incident.roadway.roadwayClosure.pn
g>
        a                        graphic:ExternalGraphic ;
        dct:description          "icon for ems.incident.roadway.roadwayClosure" ;
        dct:title                "ems.incident.roadway.roadwayClosure icon" ;
        graphic:format           "image/png" ;
        graphic:onlineResource
<http://ows.usersmarts.com/ems/icons/tier1/Base/ems.incident.roadway.roadwayClosure.pn
g> .
```

A.2.4 Symbolizer Microtheory

The Symbolizer ontology defines a set of symbolizers (also called renderers) that defines instructions for rendering data to graphic representation. The ontology is built on top of the Graphic Ontology which defines graphic objects and properties. Symbolizers are referred by **symbol:Symbol** to describe how conceptual symbol are rendered into graphics. This section describes an overview of the microtheory, examples and rationale of some of the design decisions.

The OGC Symbology Encoding defines a number of symbolizers that uses SVG parameters defines a key value pair. While this mechanism provides extensibility by accommodating future key value pairs, there is no schema allowing knowledge of which keys are valid for a given symbolizer. Starting with Testbed 12, we decided to move away from the approach used in the previous Testbed 11 which was based on the ISO 19117 model which introduces symbol definition, symbol components and graphics. We found out that the ISO 19117 was not adequate to be represented as a semantic descriptive model. The ISO 19117 is more geared toward an implementation which can use functions to calculate positioning of symbol components. We decided to align our model more toward main stream map renderer and descriptive standards such as SVG.

A.2.4.1 Symbolizer Hierarchy

The top concept of the Symbolizer ontology is **Symbolizer**. A **Symbolizer** describes how a feature is to appear on a map. The Symbolizer describes not just the shape that should appear but also how visual variables are defined using graphical properties such as color and opacity. A Symbolizer is obtained by specifying one of the subclasses of Symbolizer and then supplying parameters to override its default behavior. The hierarchy of symbolizer is illustrated in the following figure.

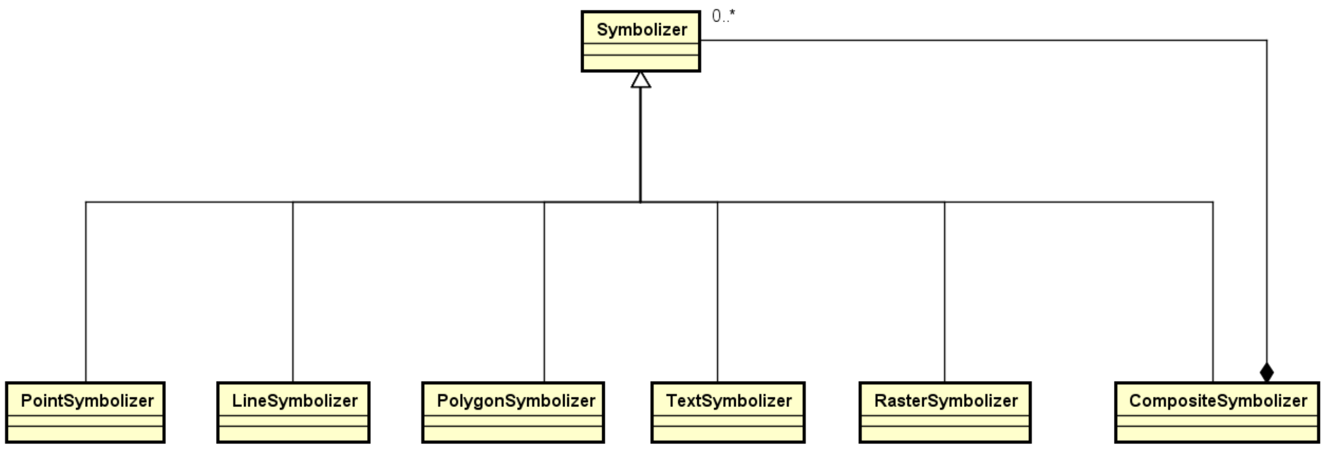


Figure 19. Symbolizer Hierarchy

Note that the hierarchy uses similar terminology than the OGC SE specification, except that it introduces an additional class called **CompositeSymbolizer** that defines a composition of symbolizers applied in a given order and using a composition operation (default is source over), as well as a **CustomSymbolizer** for modifying preexisting Symbolizers. This version of the ontology mostly focused on rendering of feature (vector) data, and is not addressing in depth the Raster Symbolizer, which is introduced as a placeholder for future extension. Other extensions of symbolizer may be defined down the road for more specialized use-cases such as complex symbologies that are difficult to express with graphic descriptors (example MIL 2525 symbology).

A.2.4.2 Symbolizer

Symbolizer is the top class of all the symbolizer and provides properties that are inherited by all subclasses.

Table 12. Properties of the Symbolizer Class

Property	Usage Note	Range	Multiplicity and use
name	symbolizer name	xsd:string	0..1
dct:title	title of the symbolizer	xsd:string	0..1
dct:description	description of the symbolizer	xsd:string	0..1
:uom	Unit of measure that applies to all elements included inside a Symbolizer such as stroke-width, Size, fontsize, Gap, InitialGap, Displacement and PerpendicularOffset. If no uom is set inside of Symbolizer, all units are measured in pixel.	URI	0..1
binding:binding	List of Bindings to apply to the Symbolizer or objects it contains	binding:Binding	0..*

A.2.4.3 Point Symbolizer

A **PointSymbolizer** is defined as a Symbolizer that is used to draw a “graphic” at a point. If the geometry associated with the **PointSymbolizer** is not a point, such as a line, polygon, raster, a representative point of the geometry (typically centroid) should be used.

Table 13. Properties of the PointSymbolizer Class

Property	Usage Note	Range	Multiplicity and use
graphic:graphicSymbol	graphic symbol	graphic:GraphicSymbol	0..1
:geometryProperty	Geometric Property associated with the point symbolizer	:Property	0..n

A.2.4.4 Line Symbolizer

A **LineSymbolizer** is a **Symbolizer** that is used to render a "stroke" along a linear geometry type such as string of line segments. Geometry types other than inherently linear types can also be used. If a point geometry is used, it should be interpreted as a line of “epsilon” (arbitrarily small) length with a horizontal orientation centered on the point, and should be rendered with two end caps. If a "area" geometry (for example polygon) is used, then its closed outline is used as the line string (with no end caps). If a raster geometry is used, its coverage-area outline is used for the line, rendered with no end caps.

Table 14. Properties of the LineSymbolizer Class

Property	Usage Note	Range	Multiplicity and use
:geometryProperty	Geometric Property associated with the symbolizer	:Property	0..n
graphic:stroke	Stroke associated with the line symbolizer	graphic:Stroke	0..1
graphic:perpendicularOffset	Perpendicular offset	xsd:decimal	0..1

A.2.4.5 Polygon Symbolizer

A **PolygonSymbolizer** is a **Symbolizer** that is used to render the area enclosed by a polygon. If a polygon has “holes,” then they are not filled, but the borders around the holes are stroked in the usual way. “Islands” within holes are filled and stroked, and so on. If a point geometry is referenced instead of a polygon, then a small, square, ortho-normal polygon should be constructed for rendering. If a line is referenced, then the line (string) is closed for filling (only) by connecting its end point to its start point, any line crossings are corrected in some way, and only the original line is stroked. If a raster geometry is used, then the raster-coverage area is used as the polygon.

Table 15. Properties of the PolygonSymbolizer Class

Property	Usage Note	Range	Multiplicity and use
:geometryProperty	Geometric Property associated with the symbolizer	:Property	0..n
graphic:stroke	Stroke associated with the polygon symbolizer	graphic:Stroke	0..1
graphic:fill	Fill associated with the polygon symbolizer	graphic:Fill	0..1
graphic:perpendicularOffset	Perpendicular offset	xsd:decimal	0..1
graphic:displacement	Displacement	graphic:Translation	0..1

A.2.4.6 Text Symbolizer

A **TextSymbolizer** is a **Symbolizer** that is used to render text.

Table 16. Properties of the TextSymbolizer Class

Property	Usage Note	Range	Multiplicity and use
:geometryProperty	Geometric Property associated with the symbolizer	:Property	0..n

Property	Usage Note	Range	Multiplicity and use
graphic:font	Font associated with the text symbolizer	graphic:Font	0..1
graphic:fill	Fill associated with the text symbolizer	graphic:Fill	0..1
graphic:halo	Halo associated with the text symbolizer	graphic:Halo	0..1
graphic:textLabel	Text label associated with the text symbolizer	String	1
graphic:labelPlacement	LabelPlacement associated with the text symbolizer (could be PointPlacment or LinePlacement for example)	graphic:LabelPlacement	0..1

A.2.4.7 Raster Symbolizer

A **RasterSymbolizer** is a **Symbolizer** that is used to render raster. This symbolizer has been introduced as a placeholder for future extensions, as this testbed was primarely focused on symbolization of vector data.

A.2.4.8 Composite Symbolizer

A **CompositeSymbolizer** is a symbolizer composed of a list of Symbolizers that are applied in a given order using a given composition operation. The default composition operation is **src-over** composition.

Table 17. Properties of the CompositeSymbolizer Class

Property	Usage Note	Range	Multiplicity and use
:comp-op	Composition operator valid values are " "src-over", "dest-out", "dest-over	xsd:string	0..1
:items	List of Symbolizer rendered in the order of the list	rdf:List of Symbolizers	1

A.2.4.9 Custom Symbolizer

A **CustomSymbolizer** is used to modify a previously existing Symbolizer with new Bindings and/or a new GeometryProperty.

Table 18. Properties of the CustomSymbolizer Class

Property	Usage Note	Range	Multiplicity and use
:symbolizerId	Registry identifier of the Symbolizer to modify	xsd:string	1

Property	Usage Note	Range	Multiplicity and use
:geometry Property	Property to set as the new Symbolizer’s GeometryProperty	:Property	0..1

A.2.5 Graphics Microtheory

A.2.5.1 Context

This document is prepared in the context of the Semantic Portrayal Service - a true semantic-enabled service that demonstrates semantic integration and interoperability across disparate portrayal catalogues.

A.2.5.2 Scope

The objective of this ontology is to define vocabulary terms describing graphic elements (objects and properties) using Linked Data encoding. The use of ontology languages such as OWL provides a powerful mechanism to express the semantic, classification, relationships and constraints of the graphic elements. The Linked data encoding for the graphics enables to build a "Web of Graphics" that could be referenced by other resources and the decentralized distribution of graphics information on the web, favoring reusability of graphics and linkage to application that can go beyond styling application. In the case of map portrayal application, the graphic ontology is used by symbolizers to instruct renderers the way to render features on a map.

To model the graphic elements of the ontology, we try to use a model that is close to the way graphic properties are defined in CSS and SVG. For example to encode color, the preferred way is to use literal with a **CSSColorLiteral** datatype that uses the same syntax as CSS and SVG. Similarly length can be expressed using different unit of measures (percent, cm, mm). Instead of introducing a concept of Length and Unit of Measure that could be very verbose, we choose to use introduce a datatype **LengthType** that is based on the SVG and CSS syntax.

A.2.5.3 Terminology used in the Graphics Ontology

The Graphics Ontology reuses terms from various existing specifications. The default namespace used in the documentation of the ontology is **http://www.opengis.net/ont/portrayal/graphic#**. The preferred prefix for this namespace is **graphic**. Classes and properties in the next sections that have been taken from the following namespaces.

Table 19. Namespace Prefixes

Prefix	Namespace
dct	http://purl.org/dc/terms/
geosparql	http://www.opengis.net/ont/geosparql#
owl	http://www.w3.org/2002/07/owl#
rdf	http://www.w3.org/1999/02/22-rdf-syntax-ns#
rdfs	http://www.w3.org/2000/01/rdf-schema#

Prefix	Namespace
xsd	http://www.w3.org/2001/XMLSchema#

A.2.5.4 Graphics Ontology Classes

This figure shows a UML Diagram of all classes and properties included in the Graphics Ontology.

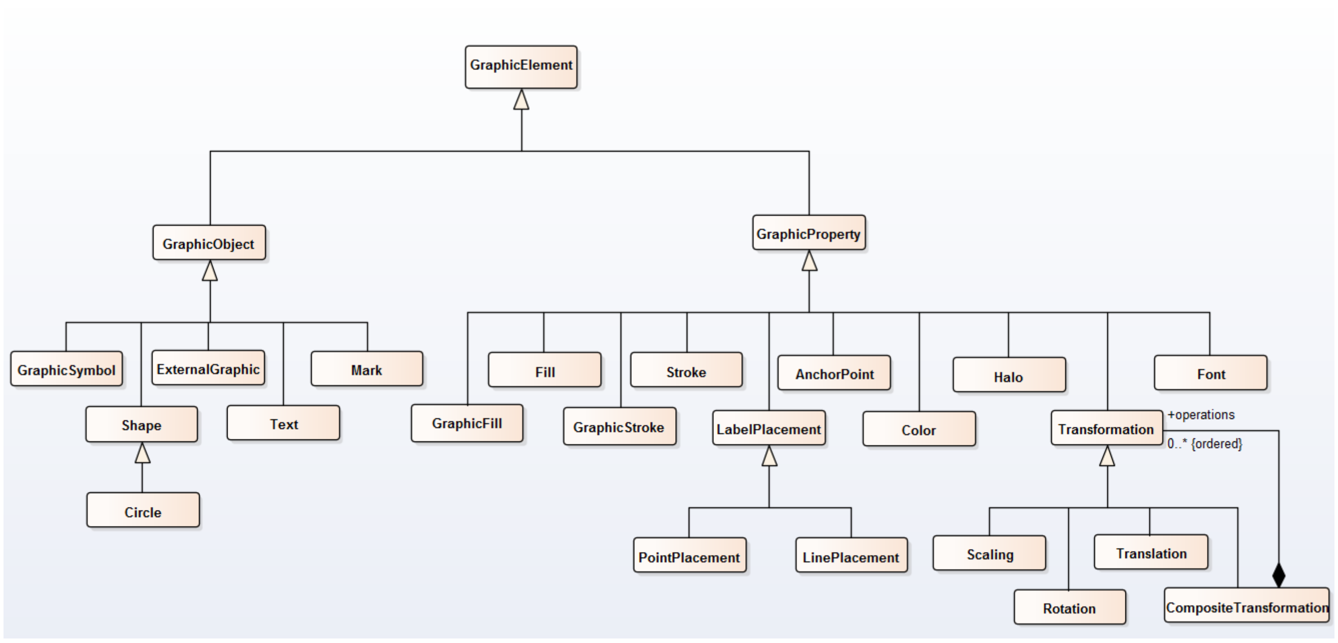


Figure 20. Graphics Ontology

Graphic Element

The **:GraphicElement** concept is the top concept of the Graphic Ontology. It is aligned with the definition introduced in ISO 19117. **:GraphicElement** is the abstract root for the graphic elements, as defined in a graphic specification language (such as SVG or OGC SE), that defines a symbol. The graphic elements may be **graphic objects**, such as shapes, texts or **graphic properties** such as color, stroke, fill, font. A **:GraphicElement** has two subclasses: **:GraphicObject** and **:GraphicProperty**.

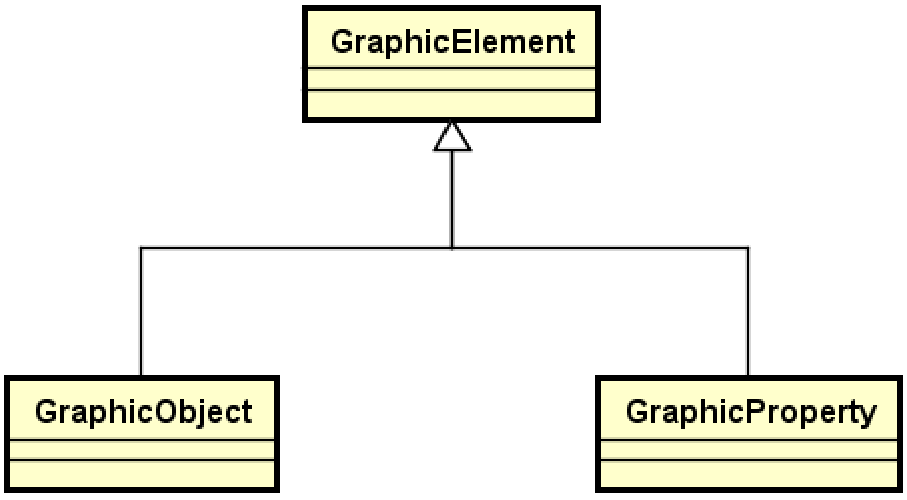


Figure 21. Graphic Element Hierarchy

Graphic Object

A **:GraphicObject** is an abstract class defined as a **:GraphicElement** that can be drawn such as a graphic shape (oval, ellipse, rectangle, path) , Raster, Glyphs (text and graphic symbols) .

NOTE	The term **SY_GraphicObject** is defined in ISO 19117 as "an abstract specialization of **SY_GraphicElement** as a graphic object defined in a graphic specification language. Graphic objects are graphic elements, such as ovals, rectangles, or paths. A graphic object in turn has properties, such as location attributes, size attributes, color attributes, etc."

The **:GraphicObject** is the base class of a hierarchy illustrated below. The detail of each subclass is described in next sections.

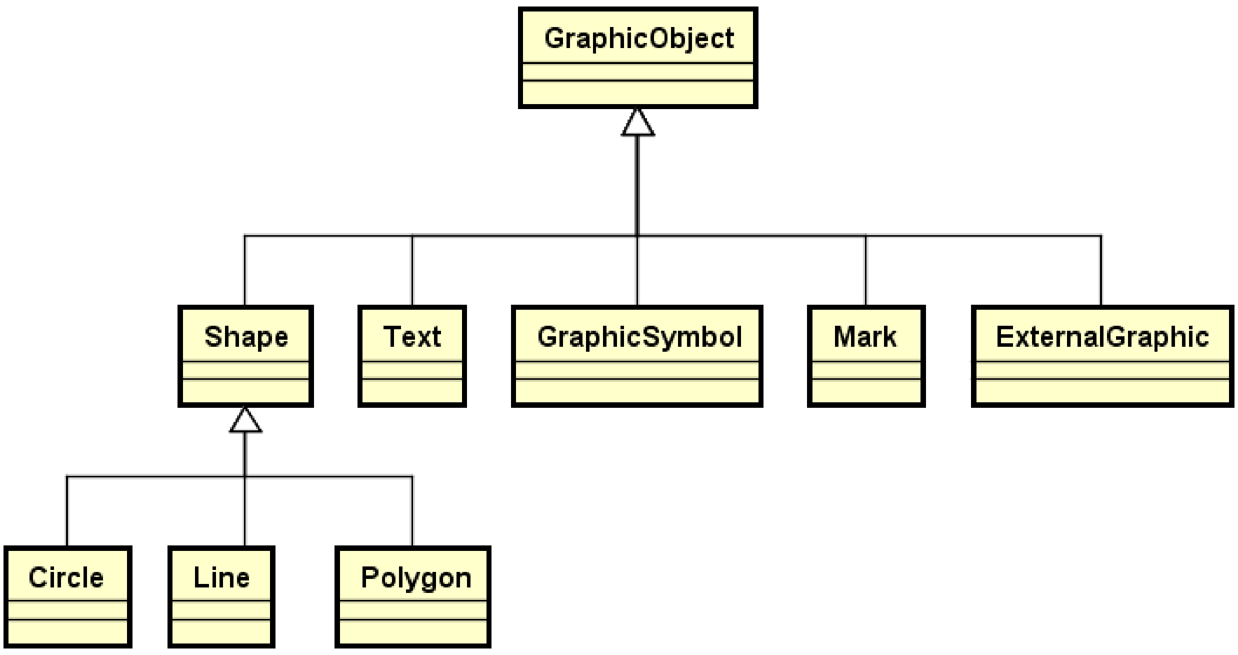


Figure 22. Graphic Object Hierarchy

Shape

A **:Shape** is a **:GraphicObject** defining a geometric figure.

NOTE	The term **:Shape** is defined in SVG as an element defined by some combination of straight lines and curves. Specifically:**path**, **rect**, **circle**, **ellipse**, **line**, **polyline**, and **polygon**.

The description of the geometry of the shape can be defined in different geometry description language such as GML, WKT and SVG. The ontology reuses the definition of GeoSPARQL **geosparql:asWKT**, **geosparql:asGML** to represent respectively geometry encoded in WKT and GML. GeoSPARQL introduces two datatypes to validate the encoding (**geosparql:WKTLiteral** and **geosparql:GMLLiteral**). To accommodate more complex paths, the graphic ontology introduces an additional property (**asSVGPath**) and datatype for encoding SVG Path (**:svgPathLiteral**), which uses the syntax of path defined in SVG.

Table 20. Properties of the Shape Class

Property	Usage Note	Range	Multiplicity and use
geosparql:asWKT	Define the geometry using Well-Known-Text Encoding	geosparql:wktLiteral	0..1 (optional)
geosparql:asGML	Define the geometry using GML Geometry Encoding	geosparql:gmlLiteral	0..1 (optional)
svgPath	specifies a path of the shape using SVG graphic language	:svgPathLiteral	0..1 (optional)

Graphic Symbol

A **:GraphicSymbol** is a **:GraphicObject** with an inherent shape, color(s), and possibly size. A **:GraphicSymbol** can be very informally defined as “a little picture” and can be of either a raster or vector-graphic source type. The term “GraphicSymbol” is used since the term “symbol” is similar to “Symbolizer” which is used in a different context in SE."

NOTE The term **:GraphicSymbol** is synonymous to **Graphic** in the SE Implementation Specification, which is defined as a “graphic symbol” with an inherent shape, color(s), and possibly size. A “graphic” can be very informally defined as “a little picture” and can be of either a raster or vector-graphic source type. The term “graphic” is used since the term “symbol” is similar to “Symbolizer” which is used in a different context in SE."

Table 21. Properties of the Graphic Symbol Class

Property	Usage Note	Range	Multiplicity and use
:externalGraphic	external graphics contained a graphic symbol	:ExternalGraphic	0..1 (optional)
:mark	Marks contained in a graphic symbol	:Mark	0..1 (optional)
:opacity	opacity of the color or the content the current object is filled with	xsd:decimal (0.0 to 1.0), representing percent opacity	0..1 (optional)
:size	Absolute size	xsd:decimal, Non-negative real number in pixels	0..1 (optional)

Property	Usage Note	Range	Multiplicity and use
:transform	Transform applied tot he graphic symbol	:Transformation	0..* (optional)
:**anchorPoint**	The location inside object to use for anchoring to main geometry point	:AnchorPoint	0..1 (optional)
:**anchorPoint**	The location inside object to use for anchoring to main geometry point	:AnchorPoint	0..1 (optional)

External Graphic

An **ExternalGraphic** gives a reference to an external raster or vector graphical object. It allows a reference to be made to an external graphic file with a Web URL or to in-line content. The **:onlineResource** property gives the URL and the **:format** property identifies the expected document MIME type of a successful fetch. Knowing the MIME type in advance allows the styler to select the best-supported format from the list of URLs with equivalent content. The **:inlineContent** property allows the content of an external graphic object to be included in-line as a xsd:base64Binary datatype.

NOTE The term **ExternalGraphic** is defined in the SE Implementation Specification as an element that "allows a reference to be made to an external graphic file with a Web URL or to in-line content."

Table 22. Properties of the ExternalGraphic Class

Property	Usage Note	Range	Multiplicity and use
:onlineResource	URL of external graphic	URL	0..1 (conditional)
:inlineContent	Content of external graphic	xsd:base64Binary	0..1 (conditional)
:format	MIME type of external graphic	xsd:string	1 (mandatory)

Mark

A **:Mark** is a **:GraphicObject** that describes a graphic symbol based on a shape with coloring applied to it.

NOTE The term **:Mark** is defined in SE as an element that defines a “shape” which has coloring applied to it. The Mark element serves two purposes. It allows the selection of simple shapes, and, in combination with the capability to select and mix multiple external-URL graphics and marks, it allows a style to be specified that can produce a usable result in a best-effort rendering environment, provided that a simple Mark is included at the bottom of the list of sources for every Graphic.

Table 23. Properties of the Mark Class

Property	Usage Note	Range	Multiplicity and use
:fill	Mark fill	:Fill	0..1 (optional)
:stroke	Marks contained in a graphic symbol	:Mark	0..1 (optional)
:shape	The shape associated with the Mark	:Shape	0..1 (conditional)
:onlineResource	URL of the graphic symbol	URI	0..1 (conditional)
:inlineContent	Content of the graphic symbol	xsd:base64Binary	0..1 (conditional)
:format	format or MIME type	xsd:string	0..1 (conditional)
:markIndex	Index to an individual mark in a mark archive; include when needed to uniquely identify mark.	xsd:string	0..1 (optional)

Text

A **:Text** is a **:GraphicObject** that describes a graphic symbol based on a shape with coloring applied to it.

NOTE The term **:Text** is defined in SVG as a "graphic element consisting of text. The attributes and properties on the text element indicate such things as the writing direction, font specification and painting attributes which describe how exactly to render the characters."

Table 24. Properties of the Text Class

Property	Usage Note	Range	Multiplicity and use
:font	The font information for the label	:Font	0..1 (optional)
:graphicSymbol	A graphic symbol to be displayed behind the label text	:GraphicSymbol	0..1 (optional)
:textLabel	The text content for the label	xsd:string	0..1 (optional)
:labelPlacement	sets the position of the label relative to its associated geometry.	:LabelPlacement	0..1 (conditional)
:halo	Creates a colored background around the label text, for improved legibility.	:cssColorLiteral	0..1 (optional)
:fill	The fill style of the label text	:Fill	0..1 (optional)

A.2.5.5 Graphic Datatypes

The Graphic ontology introduces a couple of datatypes that are used to indicate to an RDF processor how to convert and validate string literal to internal value.

cssColorLiteral

A.2.5.6 Graphic Properties

GraphicProperty

A **:GraphicProperty** is defined as a **:GraphicElement** that represents attributes that are used to modify the appearance of the graphic objects, such as its size, orientation, color, outline stroke, fill pattern, fonts, font size. While in several standards such as ISO 19117 and OGC SE, graphic properties are encoded as name/value pairs, which provide flexibility to add future extensions with new names, they do not provide any semantic information to the name and make it difficult to define the adequate restrictions of graphic properties on specific graphic objects. It is also prone to misinterpretation by implementers.

The use of ontology provides a clear semantic for each property and relationships to other properties. For example, **:font-color**, **:stroke-color** and **:fill-color** are defined as subproperties of **:color**. The graphic ontology introduces two ways to model graphic properties. The first one is by defining subproperties of the datatype property **:graphicProperty** (for example **:stroke-width**, **:stroke-color**, **:font-size**). The second way is by defining a subclass of the **:GraphicProperty** such as **:Stroke**, **:Color**, **:Font**, **:Fill**, which are often used as reusable containers for of multiple graphic properties.

NOTE The term **GraphicProperty** is defined in ISO 19117 as "a templatized specialization of **SY_GraphicElement** used to define graphic properties, such as location attributes, size attributes, color attributes, etc. As such it shall implement all inherited attributes, operations and associations. "

AnchorPoint

A **:AnchorPoint** gives the location inside of a Graphic Object to use for anchor the graphic object to the main-geometry point. The coordinates are given as two floating-point numbers in the **AnchorPointX** and **AnchorPointY** elements each with values between 0.0 and 1.0 inclusive. The default point is X=**0.5**, Y=**0.5**, which is at the middle height and middle length of the graphic/label text.

Table 25. Properties of the AnchorPoint Class

Property	Usage Note	Range	Multiplicity and use
:**anchorPointX**	The x-coordinate location inside object to use for anchoring to main geometry point	AnchorPoint	0..1 (optional)
:**anchorPointY**	The y-coordinate location inside object to use for anchoring to main geometry point	AnchorPoint	0..1 (optional)

Stroke

A **:Stroke** is a **:GraphicProperty** that describes a linear stroke. The **:GraphicStroke** element describes graphics repeated linearly and is described below. There are three basic types of strokes: solid-color, **GraphicFill** (stipple), and repeated linear **GraphicStroke**. A repeated linear graphic is plotted linearly and has its graphic Symbolizer bent around the curves of the line string, and a graphic fill has the pixels of the line rendered with a repeating area-fill pattern. If neither a

GraphicFill nor GraphicStroke element is given, then the **LineStyle** will render a solid color.

NOTE The term **:Stroke** is defined in SE as a "graphical element of the LineSymbolizer that encapsulates the graphical-Symbolization parameters for linear geometries."

Table 26. Properties of the Stroke Class

Property	Usage Note	Range	Multiplicity and use
:graphicStroke	A graphic to be used instead of a stroke	:GraphicStroke	0..n (optional)
:graphicFill	A graphic to be repeated in the area instead of a solid fill color	:GraphicFill	0..1 (optional)
:stroke-color	Color	xsd:cssColorLiteral	0..1 (optional)
:stroke-opacity	Opacity	xsd:decimal, 0.0 to 1.0, representing percent opacity	0..1 (optional)
:stroke-width	Width	xsd:decimal, Non-negative real number in pixels.	0..1 (optional)
:stroke-linecap	Linecap	xsd:string, either “butt”, “round”, or “square”	0..1 (optional)
:stroke-linejoin	Linejoin	:Linejoin, either "miter", "round", or "bevel"	0..1 (optional)
:stroke-dasharray	Dash array – a sequence of distances (dash, space, dash, etc.) to draw	xsd:string of separated non-negative real numbers in pixels	0..1 (optional)

Property	Usage Note	Range	Multiplicity and use
:stroke-dashoffset	Dash offset – the distance into the dasharray at which to start drawing	xsd:decimal, Non-negative real number in pixels	0..1 (optional)

GraphicStroke

A **:GraphicStroke** is a **:GraphicProperty** that describes a graphic symbol repeated linearly. It contains a Graphic symbol and may contain an InitialGap and/or Gap if relevant. The **:graphicSymbol** specifies the linear :**GraphicSymbol**. Proper stroking with a linear graphic symbol requires two "hot-spot" points within the space of the graphic symbol to indicate where the rendering line starts and stops. In the case of raster images with no special mark-up, this line will be assumed to be middle pixel row of the image, starting from the first pixel column and ending at the last pixel column. **GraphicStrokes** should not be used for **Strokes** that are part of Marks

NOTE The term **:GraphicStroke** is defined in SE with the same definition.

Table 27. Properties of the GraphicStroke Class

Property	Usage Note	Range	Multiplicity and use
:**graphicSymbol**	The linear graphic symbol to be drawn	:GraphicSymbol	1..n (mandatory)
:**initialGap**	How far away the first graphic symbol will be drawn relative to the start of the rendering line	xsd:decimal, Non-negative real number in pixels	0..n (optional)
:**gap**	Distance between two graphic symbols	xsd:decimal, Non-negative real number in pixels	0..n (optional))

Fill

A **:Fill** is a **:GraphicProperty** that describes a graphic symbol repeated linearly. It contains a Graphic symbol and may contain an InitialGap and/or Gap if relevant. The **:hasGraphicSymbol** specifies the linear :**GraphicSymbol**. Proper stroking with a linear graphic symbol requires two "hot-spot" points within the space of the graphic symbol to indicate where the rendering line starts and stops. In the case of raster images with no special mark-up, this line will be assumed to be middle pixel row of the image, starting from the first pixel column and ending at the last pixel column. **GraphicStrokes** should not be used for **Strokes** that are part of Marks

NOTE The term **:Fill** is defined in SE with the same definition.

Table 28. Properties of the Fill Class

Property	Usage Note	Range	Multiplicity and use
:fill-color	Color	:cssColorLiteral	0..1 (optional)
:fill-opacity	Opacity	xsd:decimal, 0.0 to 1.0, representing percent opacity	0..1 (optional)

GraphicFill

A **:GraphicFill** is a **:GraphicProperty** that describes the graphic that will be used to fill the area of a polygon. GraphicFills should not be used for Strokes that are part of Marks.

NOTE The term **:GraphicFill** is defined in SE with the same definition.

Table 29. Properties of the GraphicFill Class

Property	Usage Note	Range	Multiplicity and use
:graphicSymbol	The linear graphic symbol to be drawn.	:GraphicSymbol	1..n (mandatory)

Font

A **:Font** is a **:GraphicProperty** used to represent text or symbols.

NOTE The term **:Font** is defined in SE as an element that "identifies a font of a certain family, style, and size". Four types of **SvgParameter** are allowed, “**font-family**”, “**font-style**”, “**font-weight**”, and “**font-size**”.

Table 30. Properties of the Font Class

Property	Usage Note	Range	Multiplicity and use
:font-family	gives the family name of a font to use	:FontFamily	0..n (optional)

Property	Usage Note	Range	Multiplicity and use
:font-style	gives the style to use for a font.	xsd:string, with the allowed values being “normal”, “italic”, and “oblique”	0..1 (optional)
:font-size	gives the size to use for the font in pixels	xsd:decimal	0..1 (optional)
:font-weight	gives the amount of weight or boldness to use for a font	xsd:string, with allowed values being "normal" or "bold"	0..1 (optional)
:fill-color	Color	:cssColorLiteral	0..1 (optional)

LabelPlacement

The **:LabelPlacement** element is used to position a label relative to a graphic object. It is superclass of **:PointPlacement** and **:LinePlacement**.

LinePlacement

Table 31. Properties of the LinePlacement Class

Property	Usage Note	Range	Multiplicity and use
:initialGap	How far away the first label will be drawn relative to the start of the rendering line	xsd:decimal, Non-negative real number in pixels	0..1 (optional)
:gap	Distance between two labels	xsd:decimal, Non-negative real number in pixels	0..1 (optional))
:perpendicularOffset	gives the perpendicular distance away from a line to draw a label	:uoms	0..1 (optional))

Property	Usage Note	Range	Multiplicity and use
:generalizeLine	The boolean generalizeLine property allows the actual geometry, be it a linestring or polygon to be generalized for label placement. This is for example useful for labeling polygons inside their interior when there is need for the lavel to ressemble the shape of polygon. (See OGC Symbology Encoding)	xsd:booolean	0..1 (optional))
:isAligned	boolean indicating is the label is aligned with the line	xsd:boolean	0..1 (optional))
:isRepeated	If IsRepeated is 'true', the label will be repeatedly drawn along the line with InitialGap and Gap defining the spaces at the beginning and between labels.	xsd:boolean	0..1)

PointPlacement

Table 32. Properties of the PointPlacement Class

Property	Usage Note	Range	Multiplicity and use
:rotation	Clockwise rotation about the graphic object’s anchor point	xsd:decimal (real number in degrees)	0..1 (optional)
:anchorPoint	The location inside object to use for anchoring to main geometry point	:AnchorPoint	0..1 (optional)
:displacement	Displacement from “hot-spot” point	:Translation	0..1 (optional)

Halo

A **:Halo** is a type of **Fill** that is applied to the backgrounds of font glyphs. The use of halos greatly improves the readability of text labels.

Table 33. Properties of the Halo Class

Property	Usage Note	Range	Multiplicity and use
:fill	Halo fill	:Fill	0..1 (optional)
:radius	provides the absolute size of a halo radius in pixels encoded as a floating-point number.	xsd:decimal	0..1 (optional)

Transformation

A **:Transformation** is a **:GraphicProperty** used to apply a type of transformation, such as scaling, translation, or rotation, to a graphic object.

Size

The **Size** element gives the absolute size of the graphic object in uoms encoded as a floating-point number. The default size for an object is context-dependent. Negative values are not allowed. The default size of an image format (such as GIF) is the inherent size of the image. The default size of a format without an inherent size (such as SVG which are not specially marked) is defined to be 16 pixels in height and the corresponding aspect in width. If a size is specified, the height of the graphic object will be scaled to that size and the corresponding aspect will be used for the width. An expected common use case will be for image graphics to be on the order of 200 pixels in linear size and to be scaled to lower sizes. On systems that can resample these graphic objects “smoothly,” the results will be visually pleasing.

Table 34. Properties of the Scale Class

Property	Usage Note	Range	Multiplicity and use
:scaleX	Scale factor on the X-axis direction." The default value is 0.0	xsd:decimal	1
:scaleY	Scale factor on the Y-axis direction." The default value is 0.0	xsd:decimal	1

Rotation

The **Rotation** element is a basic transformation that gives the rotation of a graphic object in the clockwise direction about its center point in decimal degrees, encoded as a floating-point number. Negative values mean counter-clockwise rotation. The default value is 0.0 (no rotation). Note that there is no connection between source geometry types and rotations; the point used for plotting has no inherent direction. Also, the point within the graphic object about which it is rotated is format dependent. If a format does not include an inherent rotation point, then the point of rotation should be the centroid.

Table 35. Properties of the Rotation Class

Property	Usage Note	Range	Multiplicity and use
:rotation	Rotation angle in the clockwise direction about its center point in decimal degrees, encoded as a floating-point number. Negative values mean counter-clockwise rotation. The default value is 0.0 (no rotation).	xsd:decimal	1

Translation

The **Translation** gives the X and Y displacements from the “hot-spot” point. This element may be used to avoid over-plotting of multiple graphic objects used as part of the same point symbol. The displacements are in units of measure above and to the right of the point. The default displacement is X=0, Y=0. If Displacement is used in conjunction with Size and/or Rotation then the graphic object shall be scaled and/or rotated before it is displaced. The term displacement, in this case, is synonymous to **translation**.

Table 36. Properties of the Translation Class

Property	Usage Note	Range	Multiplicity and use
:displacementX	Displacement on the X-axis direction." The default value is 0.0	xsd:decimal	1
:displacementY	Displacement on the Y-axis direction." The default value is 0.0	xsd:decimal	1

Composite Transformation

A **composite transformation** is two or more transformations performed one after the other in order of the list.

Table 37. Properties of the Composite Transformation Class

Property	Usage Note	Range	Multiplicity and use
:operations	Transformation operation list executed in order of the list	rdf:List of Transformation instance	1

A.2.6 Binding Microtheory

The Binding ontology defines a vocabulary for binding an **Expression** to a property. This ontology is built on top of the Expression Ontology, which defines Expressions and their properties.

While Bindings are used only on instances of **symbolizer:Symbolizer** in the Portrayal service, the ontology is highly flexible and allows for binding an **Expression** to any property on any object.

A.2.7 Namespaces

The default namespace prefix for the Binding ontology is defined below in the **binding** entry. Other namespaces used by the ontology are defined thereafter.

Table 38. Namespace Prefixes

Prefix	Namespace
binding	http://www.opengis.net/ont/binding#
expression	http://www.opengis.net/ont/expression#
xsd	http://www.w3.org/2001/XMLSchema#

A.2.7.1 Binding Ontology Classes

This figure shows a UML Diagram of all classes and properties included in the Binding Ontology.

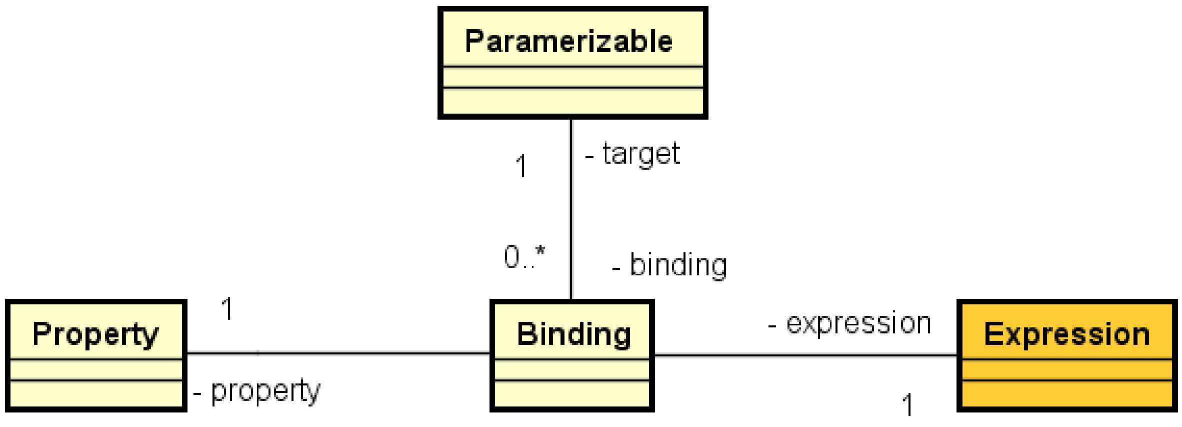


Figure 23. Binding Ontology

A.2.8 Binding Hierarchy

The top and only concept of the Binding ontology is **Binding**. A **Binding** describes how an **Expression** is bound to some property on some target object.

A.2.9 Binding

Binding is the top class of the Binding ontology.

Table 39. Properties of the Binding Class

JSON Property	RDF Property	Usage Note	Range	Multiplicity and use
expression	:expression	The bound Expression	expression:Expression	1
toProperty	:toProperty	URI of the property the Expression is bound to	xsd:anyURI	1
toTarget	:toTarget	URI of the target object, which contains the bound property. If the target URI is missing, it is assumed that the target is whatever object has the Binding.	xsd:anyURI	0..1

In addition to the properties of the **Binding** object, the Binding ontology contains a property **hasBinding**, which is used to indicate that some object has a **Binding** associated with it.

Table 40. Binding Class Related Properties

JSON Property	RDF Property	Usage Note	Range	Multiplicity and use
hasBinding	:hasBinding	The Bindings associated with this object and its sub-objects	:Binding	0..n

A.2.10 Expression Microtheory

The Expression ontology defines a way to store an expression in any machine-readable language. This ontology is defined as a separate ontology as we anticipate that it could be used in a number of standards that requires using expression for expressing conditions or derived values (for example in Web Processing Service (WPS) or workflow-related services).

A.2.11 Namespaces

The default namespace prefix for the Expression ontology is defined below in the **expression** entry. Other namespaces used by the ontology are defined thereafter.

Table 41. Namespace Prefixes

Prefix	Namespace
expr	http://www.opengis.net/ont/expression#
xsd	http://www.w3.org/2001/XMLSchema#

A.2.11.1 Expression Ontology Classes

This figure shows a UML Diagram of all classes and properties included in the Expression Ontology.

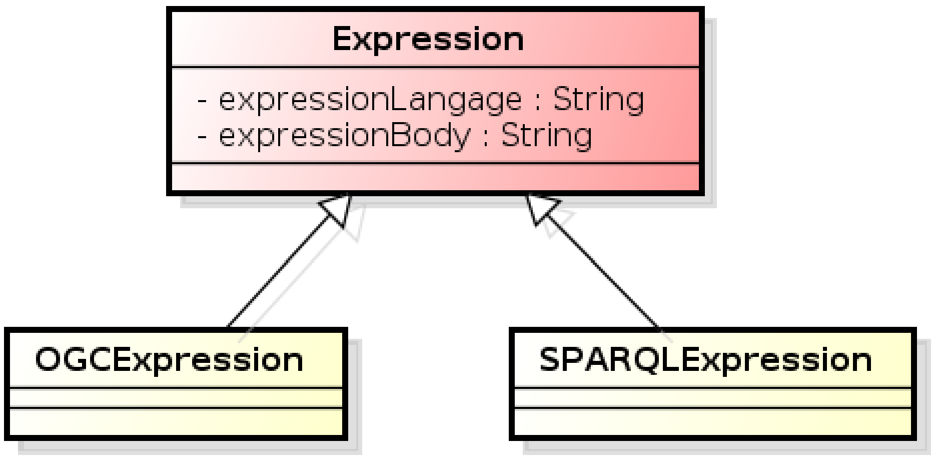


Figure 24. Expression Ontology

A.2.12 Expression Hierarchy

The top concept of the Expression ontology is **Expression**. An **Expression** describes a machine-readable expression and the language the expression is written in. While an **Expression** as defined can store any machine-readable expression, it is extensible for more specific use cases.

We have created two subclasses of **Expression** for the Portrayal service, **OGCExpression** and **SPARQLExpression**. These were created for ease of use and classification purpose for reasoning and do not provide any additional functionality over the parent **Expression** concept.

A.2.13 Expression

Expression is the top class of the expression ontology, and its properties are inherited by all subclasses.

Table 42. Properties of the Symbolizer Class

JSON Property	RDF Property	Usage Note	Range	Multiplicity and use
expressionLanguage	expr:expressionLanguage	Language of the expression. It is defined an authoritative URI.	xsd:anyURI	1
espressionBody	expr:expressionBody	Content of the expression	xsd:string	1

A.2.14 OGC Expression

The **OGCExpression** class is a specialized class for holding an OGC Expression. It has the same properties as its parent, **Expression**.

A.2.15 SPARQL Expression

The **SPARQLExpression** class is a specialized class for holding a SPARQL query. It has the same properties as its parent, **Expression**.

A.2.16 Layer Microtheory

This microtheory was introduced at the end of the Testbed 13 implementation phase to support the creation, update, cloning and search of map layers. As a consequence, it needs additional refinements in future testbeds. This section should only be considered as informative and could be used as a starting point for designing layer and map management in the portrayal service in future testbeds.

The model differs from the ones defined in current OGC services, in the sense that it accommodates different data sources using different models and format encodings (XML, JSON, CSV, Shape, WFS, GML). These data sources are typically accessible from a URL. Each data source can have a number of parameters that are represented as a set of key value pairs that are specific for each source types. The definition of the source parameters needs to be published by the service through dedicated endpoints (needs to be investigated in the future).

The focus of the modeling on the layer within the Portrayal Service was to capture the information necessary to perform the rendering of the layer. The design was not focused on the capture of metadata to enable search and discovery of layers in a semantic registry, which was investigated in the Semantic Registry Testbed 13 Thread. While there are some overlaps between both models, our focus was to capture the essential metadata needed to render the layer.

Future testbeds will need to investigate the reconciliation of the Layer/Map SRIM profile and the layer description of the portrayal service needed to perform its rendering. The role of each service should also need to be clarified, in particular, where metadata about layers should be captured.

A.2.16.1 Layer Concept

This concept defines a Map Layer that is defined from a DataSource of features (or coverage in the future) and a style. A layer has descriptive metadata, a geographic bounding box, scaling hints and reference to a data sources and one or more styles or symbolizers.

Mandatory properties

Table 43. Layer Mandatory Properties

JSON Property	URI	Range	Usage Note	Cardinality
id	none	string	Internal identifier used to refer in REST API	1

JSON Property	URI	Range	Usage Note	Cardinality
uri	rdf:id	URI	The URI of the resource (effort should done to make it resolvable)	1
type	rdf:type	Fixed value layer:Layer	The RDFS Class that the item instantiates	1
title	dct:title	rdfs:Literal	This property contains a name given to the layer. This property can be repeated for parallel language versions of the name.	1..n
description	dct:description	rdfs:Literal	This property contains a free-text account of the layer. This property can be repeated for parallel language versions of the description.	1..n
dataSource	layer:dataSource	DataSource	Reference to Data Source specification	1

A.2.16.2 Recommended properties

Table 44. Layer Recommended Properties

Property	URI	Range	Usage Note	Cardinality
layerName	1	string	Layer name used as identifier in API (example WMS)	0..1
layerType	layer:layerType	uri	Concept uri of layer type taxonomy	0..1 (?)
contactPoint	dcat:contactPoint	vcard:VCard	This property contains contact information that can be used for sending comments about the layer	0..n
publisher	dct:publisher	foaf:Agent	This property refers to an entity (organization) responsible for making the layer available	0..n
keyword/tag	dcat:keyword	rdfs:Literal	This property contains a keyword or tag describing the layer	0..n
theme/category	dcat:theme, subproperty of dct:subject	skos:Concept	This property refers to a category of the layer. A layer may be associated with multiple themes.	0..n
creation date	dct:created	rdfs:Literal typed as xsd:date or xsd:dateTime	This property contains the date of creation of the register item.	0..1

Property	URI	Range	Usage Note	Cardinality
release date	dct:issued	rdfs:Literal typed as xsd:date or xsd:dateTime	This property contains the date of formal issuance (e.g., publication) of the register item.	0..1
update/modification date	dct:modified	rdfs:Literal typed as xsd:date or xsd:dateTime	This property contains the most recent date on which the register item was modified.	0..1
featureType	layer:featureType	string	Feature type identifier	0..1 (?)
minScaleDenominator	layer:minScaleDenominator	xsd:integer	Suggested Minimum scale denominator for usage of the layer	0..1
maxScaleDenominator	layer:maxScaleDenominator	xsd:integer	Suggested Maximum scale denominator for usage of the layer	0..1
supportedStyleIds	none	string	Reference to style identifiers managed by the service	0..n
supportedStyle	layer:supportedStyle	Style	References to supported Styles (in RDF)	0..n
styles	layer:style	style:Style	Reference to Style specification (inline or by uri) in RDF	0..n
styleIds	none	string	Reference to style identifiers (JSON) managed by the service	0..n
symbolizers	layer:symbolizer	symbolizer:Symbolizer	Reference to Symbolizer Object (inline or by uri)	0..n
symbolizerIds	none	string	reference to symbolizer ids managed by service (JSON)	0..n
supportedCRS	layer:supportedCRS	string	Reference to standard identifier (use of URL)	0..n
geographicBoundingBox	extent:geographicBoundingBox	extent:GeographicBoundingBox	Geographic Bounding Box of the layer (can be calculated on server side when layer is submitted)	0..1

A.2.16.3 DataSource Concept

A DataSource represents an instance of a specific Data source type. A Data source can be accessed through a URL or be inline (GeoJSON annotations for example). Each data source type can have a set of parameters that can be defined as key/value pairs in GeoJSON or as RDF properties of the Data Source Type definition in the ontology (needs to be investigated). The publication of the parameters in the API needs to be determined in future testbed.

Table 45. DataSource Properties

JSON Property	URI	Range	Usage Note	Cardinality
id	none	string	Internal identifier used to refer in REST API	1
url	dcat:accessURL	URL	Access URL of the datasource	0..1
type	rdf:type	rdfs:Class subclass of DataSource	The RDFS Class that of the datasource	1
title	dct:title	rdfs:Literal	This property contains a name given to the datasource. This property can be repeated for parallel language versions of the name.	1..n
description	dct:description	rdfs:Literal	This property contains a free-text account of the data source. This property can be repeated for parallel language versions of the description.	1..n
version	dct:version	string	version of the data source (for example 1.3 for WMS)	0..1
mimeType	dcat:mimeType	string	mime type of the data source if downloadable	0..1
config	none	Set of key/value pairs	Map of key value pairs corresponding to parameter bindings (for JSON)	0..1
inlineData	layer:inlineData	string	inline encoding of the data (for example GeoJSON for annotations)	0..1

Appendix B: Semantic Portrayal Service REST API

B.1 Overview

This document describes the update of the RESTful Semantic Portrayal Service API, initially designed during Testbed 12. It is anticipated that a variety of clients may be using the Semantic Portrayal Service, and as a consequence it is difficult to accommodate the needs of every type of client. It is almost certain that the REST API will evolve and be modified as more requirements and features are added to the service. The hypermedia-driven API provides a robust approach to evolve the API without breaking the client ecosystem, as long as the clients are using the semantics of the link relation types. For this reason, the service has adopted a hypermedia-driven RESTful API by default, which meets level 3 of the Richardson Maturity Model [http://martinfowler.com/articles/richardsonMaturityModel.html].

The default serialization of the information model in the service is JSON, as it is understood by most programming languages. However, to support machine-processable information, the JSON model was enforced to be compatible with Linked data by using JSON-LD Context. In this way, data can be converted to Linked Data Representation and be reasoned on and linked to other information expressed as Linked Data. The REST API does support content negotiation to return a response in Linked Data Format (RDF/XML, Turtle, NTriple) when applicable. The REST API in this document provides the core minimum functionalities that are based on the **Portrayal Ontologies** described in Appendix A.

NOTE The example URL used in this documentation used the hostname http://localhost:port. This hostname should be replaced by the entry point (baseURL) of the service that you want to access in the format: http://{hostname}:{port}/{rootPath}

B.2 HTTP verbs

The RESTful API tries to adhere as closely as possible to standard HTTP and REST conventions in its use of HTTP verbs. The following HTTP verbs are used by the Service.

Verb	Usage
`GET`	Used to retrieve a resource
`POST`	Used to create a new resource
`PUT`	Used to update an existing resource with a complete update
`DELETE`	Used to delete an existing resource

B.3 HTTP status codes

The Service REST API tries to adhere as closely as possible to standard HTTP and REST conventions in its use of HTTP status codes. The following table summarizes the status codes and usage.

Status code	Usage
`200 OK`	The request completed successfully
`201 Created`	A new resource has been created successfully. The resource's URI is available from the response's `Location` header
`204 No Content`	An update to an existing resource has been applied successfully
`400 Bad Request`	The request was malformed. The response body will include an error providing further information
`404 Not Found`	The requested resource did not exist
`405 Method not allowed`	The request was made of a resource using a request method not supported by that resource; for example, using GET on a form which requires data to be presented via POST, or using PUT on a read-only resource.
`409 Conflict`	The request could not be completed due to a conflict with the current state of the resource. This code is only allowed in situations where it is expected that the user might be able to resolve the conflict and resubmit the request.
`415 Unsupported media type`	The server is refusing to service the request because the entity of the request is in a format not supported by the requested resource for the requested method.
`500 Internal Server Error`	The Web server encountered an unexpected condition that prevented it from fulfilling the request by the client

B.4 Headers

Every response has the following header(s):

Name	Description
Content-Type	The Content-Type of the payload, e.g. `application/hal+json`

Future headers may be added for managing access control to the resources managed by the service.

B.5 Errors

Whenever an error response (status code >= 400) is returned, the body will contain a JSON object that describes the problem. The error object has the following structure:

Path	Type	Description
timestamp	Number	The time, in milliseconds, at which the error occurred
status	Number	The HTTP status code, e.g. `400`
error	String	The HTTP error that occurred, e.g. `Bad Request`
message	String	A description of the cause of the error
path	String	The path to which the request was made

For example, a request that attempts to apply a non-existent resource to a symbolset item will produce a `400 Bad Request` response:

```
HTTP/1.1 400 Bad Request
Content-Type: application/json;charset=UTF-8
Content-Length: 189

{
  "timestamp" : 1461855421813,
  "status" : 400,
  "error" : "Bad Request",
  "message" : "The resource item 'http://localhost:8080/styles/123' does not exist",
  "path" : "/schemas"
}
```

B.6 Paging and Sorting

B.6.1 Paging

Rather than return everything from a large result set, the REST API recognizes some URL parameters that will influence the page size (`size` parameter) and starting page number (`page` parameter) as well as sorting of the result set (`sort` parameter).

See the following example, where the page size is set to 5 and request the third page (page 2) as page numbers are zero-indexed:

```
GET /symbolsets?page=2&size=5 HTTP/1.1
Host: localhost
```

The paginated results in HAL format returns the following response:

```
{
    "_embedded": {

        ...data...

    },
    "_links": {
        "self": {
            "href": "http://localhost:8080/symbolsets"
        },
        "first": {
            "href": "http://localhost:8080/symbolsets?page=0&size=5"
        },
        "prev": {
            "href": "http://localhost:8080/symbolsets?page=1&size=5"
        },
        "next": {
            "href": "http://localhost:8080/symbolsets?page=3&size=5"
        },
        "last": {
            "href": "http://localhost:8080/symbolsets?page=5&size=5"
        },
    },
    "page": {
        "size": 5,
        "totalElements": 27,
        "totalPages": 6,
        "number": 2
    }
}
```

Each paged response will return links to the first, previous, next, and last page of results based on the current page using the IANA defined link relations first [http://www.w3.org/TR/html5/links.html#link-type-first], prev [http://www.w3.org/TR/html5/links.html#link-type-prev], next [http://www.w3.org/TR/html5/links.html#link-type-next], last [http://www.w3.org/TR/html5/links.html#link-type-last]. If you are currently at the first page of results, however, no `prev` link will be rendered. The same is true for the last page of results: no `next` link will be rendered.

The paginated results also have extra data about the page settings , including the size of a page, total elements, total pages, and the page number you are currently viewing. This extra information makes it very easy for the consumer to configure UI tools like sliders or indicators to reflect the overall position the user is in viewing the data. For example, the document above shows a reference to the third page (with page numbers indexed to 0 being the first).

B.6.2 Sorting

The REST API recognizes sorting parameters. To have your results sorted on a particular property, add a `sort` URL parameter with the name of the property you want to sort the results on. You can control the direction of the sort by appending a `,` to the the property name plus either `asc`or `desc`.

The following examples will sort results by title in ascending order:

```
GET /symbolsets?sort=title,asc HTTP/1.1
Host: www.mydomain.com
```

To sort the results by more than one property, keep adding as many `sort=PROPERTY` parameters as you need. They will be added in the order they appear in the query string.

B.7 Search Results

A number of endpoints of the service return search results. They usually support Level 2 (JSON) and Level 3 (HAL+JSON) responses. The search results contains the collection of matched items, paging information and optionally faceted aggregation results. The following describes the response structure for each format.

B.7.1 HAL+JSON Search Results

Path	Type	Description
`_embedded`	`Array`	The HAL _embedded field that contains a collection of instances
embedded._collectionName[]	`Array`	The array of items instances defined a given JSON schema. The *collectionName* may varied depending of the types of items contained in the collection.
`aggregations[]`	`Array`	The aggregation results as defined in the Aggregation JSON Schema. This field is optional is no faceted search is performed on the endpoint.
`aggregations[].name`	`String`	The name of the aggregation (or facet)
`aggregations[].metrics`	`Object`	The metrics object containing global statistics of the object
`aggregations[].metrics.count`	`Number`	The total count of the aggregation
`aggregations[].buckets[]`	`Array`	The array of buckets of the aggregation
`aggregations[].buckets[].label`	`String`	The label of the bucket
`aggregations[].buckets[].count`	`String`	The count of the label for the bucket
`_links`	`Object`	Links to other states
`page`	`Object`	The page state. See Paging section
`page.size`	`Number`	The page size
`page.totalElements`	`Number`	The total elements matched by the search request
`page.totalPages`	`Number`	The total number of pages in the results
`page.number`	`Number`	The current page number (starts at 0)

Path	Type	Description
`_links[]`	`Array`	Array of hypermedia links to other reachable states.

B.7.2 JSON Search results

When the JSON or JSON-LD response is retrieved, the matched items are placed in an array referred to by the *results* field. Aggregations results are present only when faceted search is performed and paging information are returned.

Path	Type	Description
`results[]`	`Array`	The array of vocabulary instances that matches the search criteria
`aggregations[]`	`Array`	The aggregation results as defined in the Aggregation JSON Schema. This field is optional is no faceted search is performed on the endpoint.
`aggregations[].name`	`String`	The name of the aggregation (or facet)
`aggregations[].metrics`	`Object`	The metrics object containing global statistics of the object
`aggregations[].metrics.count`	`Number`	The total count of the aggregation
`aggregations[].buckets[]`	`Array`	The array of buckets of the aggregation
`aggregations[].buckets[].label`	`String`	The label of the bucket
`aggregations[].buckets[].count`	`String`	The count of the label for the bucket
`page`	`Object`	The page state. See <<_paging,Paging section>
`page.size`	`Number`	The page size
`page.totalElements`	`Number`	The total elements matched by the search request
`page.totalPages`	`Number`	The total number of pages in the results
`page.number`	`Number`	The current page number (starts at 0)

B.7.3 Aggregation JSON Schema

In many use cases, it is useful to aggregate the search results according some facets values (number of items per topic, per publisher, etc). Each facet is composed of a name, global metrics (right now only total count is supported), a set of buckets containing a label (unique value of the facet) and the total count for each label within the context of the facet and search results. The JSON schema of the Aggregation results is defined in the following table.

Path	Type	Description
`aggregations[]`	`Array`	The aggregation results
`aggregations[].name`	`String`	The name of the aggregation (or facet)
`aggregations[].metrics`	`Object`	The metrics object containing global statistics of the object
`aggregations[].metrics.count`	`Number`	The total count of the aggregation
`aggregations[].buckets[]`	`Array`	The array of buckets of the aggregation
`aggregations[].buckets[].label`	`String`	The label of the bucket
`aggregations[].buckets[].count`	`String`	The count of the label for the bucket

B.8 Resources Summary

The following resources are the core resources supported by the Semantic Portrayal Service.

Resources	Description	Operations
Capabilities	This resource describes the capabilities of the service	GET
Items	Represents any portrayal artifacts managed by a backend portrayal registry and supports faceted search on portrayal information.	GET
StyleSets	represents a collection of Styles and supports search on a collection of StyleSet instances based on search criteria	GET
StyleSet	represents an instance of a StyleSet	GET
Styles	represents a collection of Styles and supports search on a collection of Style instances based on search criteria	GET
Style	represents an instance of a Style	GET
SymbolSets	represents a collection of Symbol Sets and supports search on a collection of symbol sets based on search criteria	GET
SymbolSet	represents a collection of Symbol Sets and supports search on a collection of symbol sets based on search criteria	GET
GlyphRenderer	This resource supports rendering of a symbol or symbolizer to support legend rendering and previous of symbolizer/symbols	GET
LayerRenderer	This resource supports rendering of data for a given style	GET, POST
MapRenderer	This resource supports rendering of map composed of multiple layers using WMS GetMap operation	GET
Import	Import portrayal information for well defined standard (SLD, Linked Data)	POST
Export	Export Portrayal information to well defined standards (RDF, SLD 1.0, SLD 1.1)	GET
JSONLD Context	This resource returns the JSON-LD context associated with the JSON representation returned by the service	GET
SPARQL	This resource provides a SPARQL service query endpoint that can accomodate more advanced query capabilities on the portrayal information	GET
Layers (experimental)	represents a collection of user defined layers and supports search on a collection of Layer instances based on search criteria	GET, POST
Layer (experimental)	represents an instance of a Layer	GET, PUT, DELETE

B.9 Level 2 REST Endpoints

The following table describes the endpoint URL paths for each resource for a Level 2 REST API. These paths are considered **NON-NORMATIVE** but rather **INFORMATIVE**, as changes of path templates may occur in the future, requiring updates of clients and version control of APIs. The use of hypermedia REST API is advised to be isolated from these changes. However this information is provided to support frameworks that work with Level 2 REST API (such as AngularJS)

Path	HTTP Methods	Consume	Produce
/	GET,HEAD		application/hal+json
/capabilities	GET,HEAD		• application/hal+json • application/json
/items	GET,HEAD		• application/hal+json • application/json
/items	POST	• application/json • application/rdf+xml, • text/turtle, • text/n3, • application/n-triples	• application/hal+json • application/json • application/rdf+xml, • text/turtle, • text/n3, • application/n-triples

Path	HTTP Methods	Consume	Produce
/items/{id}	GET,HEAD		• application/hal+json • application/json • application/rdf+xml, • text/turtle, • text/n3, • application/n-triples
/items/instance	GET,HEAD		• application/hal+json • application/json • application/rdf+xml, • text/turtle, • text/n3, • application/n-triples
/items/{id}	PUT	• application/json • application/rdf+xml, • text/turtle, • text/n3, • application/n-triples	• application/hal+json • application/json • application/rdf+xml, • text/turtle, • text/n3, • application/n-triples
/items/{id}	DELETE		
/stylesets	GET,HEAD		• application/hal+json • application/json

Path	HTTP Methods	Consume	Produce
/stylesets/instance	GET,HEAD		• application/hal+json • application/json • application/rdf+xml, • text/turtle, • text/n3, • application/n-triples
/stylesets/{id}	GET,HEAD		• application/hal+json • application/json • application/rdf+xml, • text/turtle, • text/n3, • application/n-triples
/styles	GET,HEAD		• application/hal+json • application/json
/styles/instance	GET,HEAD		• application/hal+json • application/json • application/rdf+xml, • text/turtle, • text/n3, • application/n-triples

Path	HTTP Methods	Consume	Produce
/styles/{id}	GET,HEAD		• application/hal+json • application/json • application/rdf+xml, • text/turtle, • text/n3, • application/n-triples
/symbolsets	GET,HEAD		• application/hal+json • application/json
/symbolsets/instance	GET,HEAD		• application/hal+json • application/json • application/rdf+xml, • text/turtle, • text/n3, • application/n-triples
/symbolsets/{id}	GET,HEAD		• application/hal+json • application/json • application/rdf+xml, • text/turtle, • text/n3, • application/n-triples

Path	HTTP Methods	Consume	Produce
/symbols	GET,HEAD		• application/hal+json • application/json
/symbols/instance	GET,HEAD		• application/hal+json • application/json • application/rdf+xml, • text/turtle, • text/n3, • application/n-triples
/symbols/{id}	GET,HEAD		• application/hal+json • application/json • application/rdf+xml, • text/turtle, • text/n3, • application/n-triples
/layers	POST	• application/json	• application/hal+json • application/json
/layers/{id}	GET,HEAD		• application/hal+json • application/json

Path	HTTP Methods	Consume	Produce
/layers/{id}	PUT	• application/json	• application/hal+json • application/json
/layers/{id}	DELETE		
/layers/{id}/render	GET		• Supported output graphic formats
/layers/{id}/render	POST	application/json	• Supported output graphic formats
/renderer/symbolizer	GET,HEAD		• Supported output graphic formats
/renderer/symbol	GET,HEAD		• Supported output graphic formats
/renderer/layer	GET,HEAD		• Supported output graphic formats
/renderer/layer	POST		• Supported output graphic formats
/renderer/map	GET,HEAD		• Supported output graphic formats
/renderer/map	POST		• Supported output graphic formats

Path	HTTP Methods	Consume	Produce
/sparql	GET,HEAD		• application/sparql-results+xml • application/sparql-results+json • text/csv • text/tab-separated-values • application/rdf+xml, • text/turtle, • text/n3, • application/n-triples
/sparql	POST	• application/x-www-form-urlencoded • application/sparql-query	• application/sparql-results+xml • application/sparql-results+json • text/csv • text/tab-separated-values • application/rdf+xml, • text/turtle, • text/n3, • application/n-triples

B.10 Link relation types

The following table describes the list of link relation types used by the Semantic Portrayal Service. The OGC namespace is used to define for the relationships that can be reused by other services such as capabilites, apiDocumentation. The links specific to the portrayal service have the following relation type URI template: http://www.opengis.net/rels/portrayal/{rel}.

Relation type	URI	Description
ogc:capabilities	http://www.opengis.net/rels/capabilities	Reference to the capabilities of the service
portrayal:items	http://www.opengis.net/rels/portrayal/renderer/symbol	Reference to the portrayal item search endpoint
portrayal:styles	http://www.opengis.net/rels/portrayal/styles	Reference to a collection of Style instances in the portrayal service
portrayal:style	http://www.opengis.net/rels/portrayal/style	Reference to an instance of Style in the portrayal service
portrayal:symbolSets	http://www.opengis.net/rels/portrayal/symbolSets	Reference to a collection of SymbolSet instances in the portrayal service
portrayal:symbolSet	http://www.opengis.net/rels/portrayal/symbolSet	Reference to an instance of SymbolSet in the portrayal service
portrayal:symbols	http://www.opengis.net/rels/portrayal/symbols	Reference to a collection of Symbol instances in the portrayal service
portrayal:symbol	http://www.opengis.net/rels/portrayal/symbol	Reference to an instance of Symbol in the portrayal service
portrayal:mapRenderer	http://www.opengis.net/rels/portrayal/renderer/map	Reference to the portrayal map renderer
portrayal:symbolRenderer	http://www.opengis.net/rels/portrayal/renderer/symbol	Reference to the portrayal symbol renderer
portrayal:symbolizerRenderer	http://www.opengis.net/rels/portrayal/renderer/symbolizer	Reference to the portrayal symbolizer renderer
portrayal:layerRenderer	http://www.opengis.net/rels/portrayal/renderer/layer	Reference to the portrayal layer renderer
portrayal:importer	http://www.opengis.net/rels/portrayal/import	Reference to the portrayal information importer
portrayal:exporter	http://www.opengis.net/rels/portrayal/import	Reference to the portrayal information exporter

Relation type	URI	Description
ogc:jsonldContext	http://www.opengis.net/rels/jsonldContext	Reference to the JSON-LD context to apply to the JSON content to transform it to Linked Data
ogc:documentation	http://www.opengis.net/rels/restdoc	Reference to the REST documentation of the service

NOTE Due to the time constraints, the hypermedia links for all the endpoints have not been all implemented during this testbed. Future testbeds will need to address the use of hyperlinks once all the endpoints are stabilized.

B.11 Content negotiation

Most of the resource can serve multiple representations including:

Format	Mime type	Description
HAL+JSON	application/hal+json	It is the **default format** of the service. The JSON payload is compatible with JSON-LD and is aligned with the Portrayal Ontology Ontology.
JSON-LD	application/ld+json	Compliant with the Portrayal Ontology using the JSON-LD context.
JSON	application/ljson	Compliant with the Portrayal Ontology Ontology using the JSON-LD context.
RDF/XML	application/rdf+xml	Compliant with the Portrayal Ontology.
Turtle	text/turtle	Compliant with the Portrayal Ontology.
N-Triples	text/ntriples	Compliant with thePortrayal Ontology.

Renderers produce different formats depending of their type such as PNG, GIF, JPEG, SVG, KML etc.

B.12 Semantic Portrayal Resources

This section describes the list of resources made accessible by the Semantic Portrayal Service. The RESTful API has an entry point (service root) which provides links which represents the current state transitions supported by the service.

B.12.1 Service Root

The root is the entry point of the RESTful Semantic Portrayal service – it is what the client comes into contact with when consuming the API for the first time. If the HATEOAS constraint is to be considered and implemented throughout by clients, then this is the place to start. The root endpoint provides **a set of links** with well-defined relation types that corresponds to the supported capabilities of the service. As the Semantic Portrayal Service API evolves in the future, there would be many more links, each with it's own semantics defined by the type of link relation [http://tools.ietf.org/html/rfc5988#section-5.3].

The API is considered as RESTful as it is fully discoverable **from the root** and with **no prior knowledge** – meaning the client should be able to navigate the API by doing a GET on the root. Moving forward, all state changes are driven by the client using the available and discoverable transitions that the REST API provides in representations (hence Representational State Transfer).

B.12.1.1 Accessing the root endpoint

To get access to the root of the service and get the list of links, a `GET` request is sent to the base url of the service.

B.12.1.2 Request structure

This request sends a 'GET' request to the base url of the service. Here the HTTP request associated to get the list of links supported by the service:

```
GET / HTTP/1.1
Host: localhost
```

B.12.1.3 Query Parameters

None

B.12.1.4 Response structure

The response is returning a HAL response (application/hal+json) with the **_links** object that contains links associated with relation types (used as property names in JSON).

Path	Type	Description
@context	String	JSON-LD Context applicable to JSON response of the service

Path	Type	Description
type	String	The type of the resource (always Service)
title	String	Title of the service
description	String	Description of the service
categories	Array	Categories of the service
_links	Object	Links to other resources

B.12.1.5 Links

The following table describes the current link relation types supported by the service.

Relation	Description
portrayal:capabilities	Refers to the capabilities of this service.
portrayal:styles	Refers to the style search endpoint supported by this service
portrayal:symbols	Refers to the symbol search endpoint supported by this service
portrayal:symbolSets	Refers to the symbolSet search endpoint supported by this service
portrayal:renderer	Refers to the renderer endpoint supported by this service
portrayal:sparql	Refers to the SPARQL endpoint supported by this service
ogc:jsonLdContext	Refers to JSON Context
curies	The curies to use to expand the link relation types to URI

B.12.1.6 Example response

The following example response shows the current links supported by the service.

```
HTTP/1.1 200 OK
Content-Type: application/hal+json
Content-Length: 921

{
    "type": "srim:Service",
    "title": "Semantic Portrayal Service",
    "description": "Semantic Portrayal Service prototype for OGC Testbed12",
    "category": [
        "http://www.opengis.net/specs/testbed12/semanticPortrayalService"
    ],
    "publisher": [{
        "name": "Image Matters LLC",
            "uri": "http://www.imagemattersllc.com",
            "type": "org:Organization"
        }],
        "version": "0.1",
        "_links": {
            "self": {
                "href": "http://localhost:8082"
            },
            "portrayal:capabilities": {
                "href": "http://localhost:8082/capabilities"
            },
            "portrayal:items": {
                "href": "http://localhost:8082/items"
            },
            "portrayal:layers": {
                "href": "http://localhost:8082/layers"
            },
            "portrayal:symbols": {
                "href": "http://localhost:8082/symbols"
            },
            "portrayal:symbolsets": {
                "href": "http://localhost:8082/symbolsets"
            },
            "portrayal:styles": {
                "href": "http://localhost:8082/styles"
            },
            "portrayal:mapRenderer": {
                "href": "http://localhost:8082/renderer/map"
            },
            "portrayal:layerRenderer": {
                "href": "http://localhost:8082/renderer/layer"
            },
            "portrayal:symbolRenderer": {
                "href": "http://localhost:8082/renderer/symbol"
            },
            "portrayal:symbolizerRenderer": {
                "href": "http://localhost:8082/renderer/symbolizer"
            },
            "portrayal:legendRenderer": {
                "href": "http://localhost:8082/renderer/legend"
            },
            "portrayal:import": {
                "href": "http://localhost:8082/import"
            },
            "portrayal:export": {
                "href": "http://localhost:8082/export"
            },
            "ogc:jsonldContext": {
                "href": "http://localhost:8082/context"
            },
            "portrayal:sparql": {
```

```
            "href": "http://localhost:8082/sparql"
        },
        "curies": [{
            "href": "http://www.opengis.net/rels/portrayal/{rel}",
            "name": "portrayal",
            "templated": true
        }]
    }
}
```

B.12.2 Capabilities

This resource describes the capabilities of the service, including the supported portrayal items, application profiles, formats.

B.12.2.1 Query Parameters

There is no query parameter to retrieve the capabilities.

B.12.2.2 Example request

The following is an example of GET Request performed on the Capability resource.

```
GET /capabilities HTTP/1.1
Accept: application/hal+json
Host: localhost
```

B.12.2.3 Response structure

The response of Capability request has the following structure:

Path	Type	Description
`keywords[]`	`Array`	Keywords associated with the service description
`itemClasses[]`	`Array`	ItemClasses supported by the registry
`itemClasses[].id`	`String`	The ID of the ItemClass
`itemClasses[].uri`	`String`	The URI of the ItemClass
`itemClasses[].type`	`String`	The RDF type of the ItemClass (always srim:ItemClass)
`itemClasses[].title`	`String`	The human readable title of the ItemClass
`itemClasses[].description`	`String`	The human readable description of the ItemClass

Path	Type	Description
`rendererFormats[]`	Array	Supported Renderer Formats supported by the service
`importerFormats[]`	Array	Supported Importer Formats by the service
`importerFormats[].id`	`string	Identifier of the type of the imported format
`importerFormats[].title`	`string`	Title of the type of the imported format
`importerFormats[].description`	`string`	Description of the imported format
`importerFormats[].mimeType`	`string`	Mime type the imported format if applicable
`importerFormats[].encoding`	`string`	Encoding of the imported format if applicable
`importerFormats[].schemas[]`	`string`	Supported Schemas of the imported format
`supportedDataSourceTypes[]`	`String`	Supported DataSource Types by the service
`supportedDataSourceTypes[].id`	`string`	Identifier of the type of the data source format
`supportedDataSourceTypes[].title`	`string`	Title of the type of the data source format
`supportedDataSourceTypes[].description`	`string`	Description of the type of the data source format
`supportedDataSourceTypes[].mimeType`	`string`	Mime type the data source format if applicable
`supportedDataSourceTypes[].encoding`	`string`	Encoding of the data source format if applicable
`supportedDataSourceTypes[].schemas[]`	`string`	Supported Schemas of the type of the data source format
`defaultCRS`	`String`	Default CRS supported by the service
`supportedCRSs[]`	`Array`	Supported RS by the service
`_links`	`Object`	The hypermedia links to other states

B.12.2.4 Links

The following table describes the current link relation types in the HAL capabilities response

Relation	Description
`service`	Reference to the entry point of the service

Relation	Description
self	Refers to this endpoint.
curies	Refers to the curies defined for the links

B.12.2.5 Example response

The following example response shows how capabilities of the service is represented.

```
HTTP/1.1 200 OK
Content-Type: application/hal+json
Content-Length: 30456

{
    "rendererFormats": [
        "image/svg+xml",
        "image/png",
        "image/jpeg",
        "image/gif",
        "image/geotiff"
    ],
    "itemClasses": [{
        "id": "style:Style",
        "uri": "http://www.opengis.net/ont/portrayal/style#Style",
        "type": "srim:ItemClass",
        "title": "Style",
        "description": "Style class"
    }, {
        "id": "style:StyleSet",
        "uri": "http://www.opengis.net/ont/portrayal/style#StyleSet",
        "type": "srim:ItemClass",
        "title": "StyleSet",
        "description": "StyleSet class"
    }, {
        "id": "style:SymbolSet",
        "uri": "http://www.opengis.net/ont/portrayal/symbol#SymbolSet",
        "type": "srim:ItemClass",
        "title": "SymbolSet",
        "description": "SymbolSet class"
    }, {
        "id": "style:PortrayalRuleSet",
        "uri": "http://www.opengis.net/ont/portrayal/style#PortrayalRuleSet",
        "type": "srim:ItemClass",
        "title": "Portrayal RuleSet",
        "description": "PortrayalRuleSet class"
    }, {
        "id": "style:FeatureTypeStyle",
        "uri": "http://www.opengis.net/ont/portrayal/style#FeatureTypeStyle",
        "type": "srim:ItemClass",
        "title": "FeatureType Style",
        "description": "FeatureType class"
    }, {
        "id": "symbol:Symbol",
        "uri": "http://www.opengis.net/ont/portrayal/symbol#Symbol",
        "type": "srim:ItemClass",
        "title": "Portrayal Symbol",
        "description": "Symbol class"
    }, {
        "id": "symbol:SymbolSet",
        "uri": "http://www.opengis.net/ont/portrayal/symbol#SymbolSet",
        "type": "srim:ItemClass",
        "title": "Symbol Set",
        "description": "SymbolSet Class"
    }, {
        "id": "style:PortrayalRule",
        "uri": "http://www.opengis.net/ont/portrayal/style#PortrayalRule",
        "type": "srim:ItemClass",
        "title": "Portrayal Rule",
        "description": "Portrayal Rule"
    }, {
        "id": "style:PortrayalRuleList",
        "uri": "http://www.opengis.net/ont/portrayal/style#PortrayalRuleList",
        "type": "srim:ItemClass",
        "title": "Portrayal RuleList",
        "description": "Portrayal RuleList "
    }, {
        "id": "style:PortrayalRuleList",
        "uri": "http://www.opengis.net/ont/portrayal/style#PortrayalRuleList",
        "type": "srim:ItemClass",
        "title": "Portrayal RuleList",
        "description": "Portrayal RuleList "
    }],
    "_links": {
        "self": {
            "href": "http://localhost:8082/capabilities"
        },
        "service": {
            "href": "http://localhost:8082"
        }
    }
}
```

B.12.3 JSON-LD Context

The JSON produced by the Semantic Portrayal Service is compatible with the Portrayal SRIM Application Profile using the Portrayal ontologies described in Appendix A, by using JSON-LD context. The context is made accessible through an endpoint, so it can be referred to and imported by JSON-LD processors to be converted into a Linked Data representation adhering to the Portrayal SRIM Profile.

B.12.3.1 Query Parameters

There is no query parameter to retrieve the JSON-LD context.

B.12.3.2 Example request

The following is an example of GET Request performed on the JSON-LD context resource.

```
GET /context HTTP/1.1
Host: localhost
```

B.12.3.3 Response structure

The response of JSON-LD Context is conforming to the standard JSON-LD.

Path	Type	Description
@context	Object	JSON-LD Context applicable to JSON response of the service

NOTE Due to time constraint, the JSON-LD context has not been updated during this testbed with the latest update in the model.

B.12.3.4 Example response

The following example response shows a JSON-LD context of the service.

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: 2222

{
    "@context": {
        "style": "http://www.opengis.net/ont/portrayal/style#",
        "symbol": "http://www.opengis.net/ont/portrayal/symbol#",
        "symbolizer": "http://www.opengis.net/ont/portrayal/symbolizer#",
        "graphic": "http://www.opengis.net/ont/portrayal/graphic#",
        "pav": "http://purl.org/pav/",
        "dct": "http://purl.org/dc/terms/",
        "owl": "http://www.w3.org/2002/07/owl#",
        "xsd": "http://www.w3.org/2001/XMLSchema#",
        "skos": "http://www.w3.org/2004/02/skos/core#",
        "srim": "http://www.opengis.net/ont/testbed/12/srim#",
        "rdfs": "http://www.w3.org/2000/01/rdf-schema#",
        "dcat": "http://www.w3.org/ns/dcat#",
        "iso639-2": "http://id.loc.gov/vocabulary/iso639-2/",
        "lingvoj": "http://www.lingvoj.org/ontology#",
        "foaf": "http://xmlns.com/foaf/0.1/",
        "ldp": "http://www.w3.org/ns/ldp#",
        "locn": "http://www.w3.org/ns/locn#",
        "adms": "http://www.w3.org/TR/vocab-adms/#",
        "extent": "http://www.opengis.net/ont/extent#",
        "idvoc": "http://www.opengis.net/ont/identifier#",
        "link": "http://www.opengis.net/ont/link#",
        "gr": "http://www.heppnetz.de/ontologies/goodrelations/v1#",
        "org": "http://www.socialml.org/ontologies/organization#",
        "schema": "http://www.opengis.net/ont/testbed12/srim/profile/schema#",
        "vcard": "http://www.w3.org/2006/vcard/ns#",
        "type": "@type",
        "uri": {
            "@id": "rdf:id",
            "@container": "@language"
        },
        "value": "@value",
        "lang": "@language",
        "results": "ldp:contains",
        "title": "dct:title",
        "titleMap": {
            "@id": "dct:title",
            "@container": "@language"
        },
        "description": "dct:description",
        "descriptionMap": {
            "@id": "dct:description",
            "@container": "@language"
        },
        "prefLabel": "skos:prefLabel",
        "prefLabelMap": {
            "@id": "skos:prefLabel",
            "@container": "@language"
        },
        "category": {
            "@id": "dct:type",
            "@type": "@id"
        },
        "version": "pav:version",
        "creator": {
            "@id": "dct:creator",
            "@type": "@id"
        },
        "versionNotes": "adms:versionNotes",
        "versionNotesMap": {
            "@id": "adms:versionNotes",
            "@container": "@language"
        },

        "contributor": {
            "@id": "dct:contributor",
            "@type": "@id"
        },
        "publisher": {
            "@id": "dct:publisher",
            "@type": "@id"
        },
        "language": {
            "@id": "dct:language",
            "@type": "@id"
        },
        "iso2Code": "lingvoj:iso1",
        "iso3Code": "lingvoj:iso2",
        "label": "rdfs:label",
        "name": "foaf:name",
        "nameMap": {
            "@id": "foaf:name",
            "@container": "@language"
        },
        "sameAs": {
            "@id": "owl:sameAs",
            "@type": "@id"
        },
        "distribution": {
            "@id": "dcat:distribution",
            "@type": "@id"
        },
        "license": {
            "@id": "dct:license",
            "@type": "@id"
        },
        "created": "dct:created",
        "issued": "dct:issued",
        "modified": "dct:modified",

        "mediaType": "dcat:mediaType",
        "country-name": "vcard:country-name",
        "region": "vcard:region",
        "locality": "vcard:locality",
        "postal-code": "vcard:postal-code",
        "street-address": "vcard:street-address",

        "inScheme": {
            "@id": "skos:inScheme",
            "@type": "@id"
        },
        "conformsTo": {
            "@id": "dct:conformsTo",
            "@type": "@id"
        },
        "page": {
            "@id": "foaf:page",
            "@type": "@id"
        },
        "rights": {
            "@id": "dct:rights",
            "@type": "@id"
        },

        "hasValue": {
            "@id": "vcard:hasValue",
            "@type": "@id"
        },

        "landingPage": {
            "@id": "dcat:landingPage",
            "@type": "@id"
        },
        "accrualPeriodicity": {
            "@id": "dct:accrualPeriodicity",
            "@type": "@id"
        },
        "purpose": "srim:purpose",

        "itemClass": {
            "@id": "srim:itemClass",
            "@type": "@id"
        },
        "keyword": "dcat:keyword",
        "identifier": "dct:identifier",

        "theme": {
            "@id": "dcat:theme",
            "@type": "@id"
        },
        "audience": {
            "@id": "dct:audience",
            "@type": "@id"
        },
        "function": {
            "@id": "srim:function",
            "@type": "@id"
        },
        "subject": {
            "@id": "dct:subject",
            "@type": "@id"
        },
        "project": {
            "@id": "foaf:project",
            "@type": "@id"
        },
        "hasIdentifier": {
            "@id": "id:hasIdentifier",
            "@type": "@id"
        },
        "depiction": {
            "@id": "foaf:depiction",
            "@type": "@id"
        },
        "provenance": {
            "@id": "dct:provenance",
            "@type": "@id"
        },
        "contactPoint": {
            "@id": "dcat:contactPoint",
            "@type": "@id"
        },
        "hasGeographicExtent": {
            "@id": "extent:hasGeographicExtent",
            "@type": "@id"
        },
        "temporal": {
            "@id": "dct:temporal",
            "@type": "@id"
        },
        "spatial": {
            "@id": "dct:spatial",
            "@type": "@id"
        },
        "accessRights": {
            "@id": "dct:accessrights",
            "@type": "@id"
        },

        "rightsHolder": {
            "@id": "dct:rightsHolder",
            "@type": "@id"
        },

        "subOrganizationOf": {
            "@id": "org:subOrganizationOf",
            "@type": "@id"
        },

        "item": {
            "@id": "srim:item",
            "@type": "@id"
        },
        "status": "srim:status",

        "itemType": {
            "@id": "srim:itemType",
            "@type": "@id"
        },
        "organization-name": "vcard:organization-name",
        "address": {
            "@id": "vcard:address",
            "@type": "@id"
        },
        "fn": "vcard:fn",
        "hasEmail": {
            "@id": "vcard:hasEmail",
            "@type": "@id"
        },
        "vcardTitle": "vcard:title",
        "hasTelephone": {
            "@id": "vcard:hasTelephone",
            "@type": "@id"
        },
        "featureType": "style:featureType",
        "rules": "style:hasRule",
        "symbolSet": "symbol:symbolSet",
        "hasRuleSet": "style:hasRuleSet",
        "styles": "style:hasStyle",
        "ruleset": "style:hasRuleSet",
        "minScaleDenominator": "style:minScaleDenominator",
        "maxScaleDenominator": "style:maxScaleDenominator",
        "constraint": "style:hasConstraint",
        "constraintLanguage": "style:constraintLanguage",
        "constraintLanguageVersion": "style:constraintLanguageVersion",
        "ruleCondition": "style:has",
        "body": "style:body",
        "browseGraphic": "symbol:browseGraphic",
        "specification": "symbol:specification",
        "denotes": "symbol:denotes",
        "uom": "symbolizer:uom",
        "comp-op": "symbolizer:comp-op",
        "symbolizer": "symbolizer:symbolizer",
        "geometryProperty": "symbolizer:geometryProperty",
        "propertyName": "symbolizer:propertyName",
        "hasGraphicContent": "graphic:hasGraphicContent",
        "graphicObject": "graphic:graphicObject",
        "graphicSymbol": "graphic:graphicSymbol",
        "mark": "graphic:shape",
        "shape": "graphic:graphicObject",
        "hasGraphicProperty": "graphic:hasGraphicProperty",
        "fontFamily": "graphic:fontFamily",
        "fontStyle": "graphic:fontStyle",
        "hasColor": "graphic:hasColor",
        "fill": "graphic:fill",
        "font": "graphic:font",
        "graphicStroke": "graphic:graphicStroke",
        "graphicFill": "graphic:graphicFill",
        "onlineResource": "graphic:onlineResource",
        "stroke": "graphic:stroke",
        "wellKnownShape": "graphic:wellKnownShape",
        "labelPlacement": "graphic:labelPlacement",
        "pointPlacement": "graphic:pointPlacement",
        "linePlacement": "graphic:linePlacement",
        "graphicContent": "graphic:graphicContent",
        "externalGraphic": "graphic:externalGraphic",
        "graphicProperty": "graphic:graphicProperty",
        "colorName": "graphic:colorName",
        "cssColor": "graphic:cssColor",
        "fill-property": "graphic:fill-property",
        "fill-color": "graphic:fill-color",
        "fill-opacity": "graphic:fill-opacity",
        "font-property": "graphic:font-property",
        "font-code": "graphic:font-code",
        "font-size": "graphic:font-size",
        "font-weight": "graphic:font-weight",
        "format": "graphic:format",
        "gap": "graphic:gap",
        "halo": "graphic:halo",
        "radius": "graphic:radius",
        "asWKT": "graphic:asWKT",
        "initialGap": "graphic:initialGap",
        "perpendicularOffset": "graphic:perpendicularOffset",
        "stroke-property": "graphic:stroke-property",
        "stroke-color": "graphic:stroke-color",
        "stroke-dasharray": "graphic:stroke-dasharray",
        "stroke-dashoffset": "graphic:stroke-dashoffset",
        "stroke-linecap": "graphic:stroke-linecap",
        "stroke-linejoin": "graphic:stroke-linejoin",
        "stroke-opacity": "graphic:stroke-opacity",
        "stroke-linejoin": "graphic:stroke-linejoin",
        "stroke-width": "graphic:stroke-width",
        "svgPath": "graphic:svgPath",
        "anchorPointX": "graphic:anchorPointX",
        "anchorPointY": "graphic:anchorPointY",
        "opacity": "graphic:opacity",
        "inlineContent": "graphic:inlineContent",
        "markIndex": "graphic:markIndex",
        "textLabel": "graphic:textLabel",
        "isRepeated": "graphic:isRepeated",
        "isAligned": "graphic:isAligned",
        "generalizeLine": "graphic:generalizeLine";
    }
}
```

B.12.4 Portrayal Items

For this testbed, a search endpoint for any portrayal items was introduced, in addition to the dedicated endpoint for each type of portrayal artifacts (symbol, styles, symbolizers). This enables faceted search across multiple types and facilitate the delegation of the search with the semantic portrayal registry based on SRIM, as they use similar parameters. It also enables the creation, update, deletion and retrieval of individual portrayal items by simply delegating the operation to the Semantic Portrayal Registry.

For the sake of brevity and avoiding duplication, the endpoint specifications for items and item resources are similar to the Semantic Registry API developed during the testbed 12. Please refer to the Testbed 12 Semantic Registry ER for more details.

B.12.4.1 Styles

A **Style** defines the set of portrayal rules to apply to some geospatial data. **Style** has subclasses such as style:FeatureTypeStyle and **style:CoverageStyle**. The **Styles resource** is used to search style instances based on some search criteria.

JSON Schema

The following defines the JSON schema for the Style derived from the Style Ontology defined in Appendix A. It can also be derived from the JSON-LD Context.

Path	Type	Description	Card.
`id`	`String`	Internal identifier for the symbol (which can be used in Level 2 API)	1
`uri`	`String`	Linked Data URI for the Style (equivalent to @id in JSON-LD)	0..1
`type`	`String`	The type (class) of the Style (symbol:Style subclasses such as style:FeatureTypeStyle).	1
`title`	`String`	The title of the Style	1
`titleMap`	`Object`	The title map for each language of the title . Each key corresponds to the two letter language identifier name (for example "en")	0..1
`description`	`String`	The description of the Style	0..1
`descriptionMap`	`Object`	The description map for each language of the description. Each key corresponds to the two letter language identifier name (for example "en")	0..1

Path	Type	Description	Card.
`created`	`String`	The date of creation (XSD datetime format) (generated by the service)	1
`modified`	`String`	The date of the last modification (XSD datetime format) (generated by the service)	0..1
featureType	uri	URI of FeatureType associated with a FeatureTypeStyle instance	0..1
rules[]	Object	Array of PortrayalRule URIs. See ontology for Symbolizer in Appendix C	1

Search Styles

The search of the Styles is performed by performing a HTTP `GET` request on the Styles resource. The style search supports free text, by types, ids, or uris, CQL constraint and can return results with aggregations on specific facets.

Query Parameters

The following query parameters are supported in the query:

Parameter	Description	Cardinality
`q`	Text to search in textual fields	0..1
`type`	One or more Style types (style:FeatureTypeStyle or other subclasses)	0..n
`uri`	One or more URIs of Style instances	0..n
`id`	One or more id of Style instances	0..n
`includeFacet`	Boolean or list of facet names to include for aggregation computation. If the value is true, include all facets supported by the server. If only a subset of the facets are needed, a comma delimited of field names can be set.	0..1
`facet.fieldname`	Constraint values of a given facet field name	0..n
`constraint`	A constraint expressed in CQL. This is used to express more advanced query filtering	0..1
`fields`	One or more fields to be included in the response. Use JSON path dot notation for referring paths	0..n
`pageNumber`	The number of the current page as defined in Paging section	0..1
`pageSize`	The count of items on the current page as defined in Paging section	0..1

Parameter	Description	Cardinality
`sort`	The sorting parameters as defined in Sorting section.	0..n

Example request

The following request performs a style search of all types

```
GET /styles HTTP/1.1
Accept: application/hal+json
Host: localhost
```

The following request search styles for the featureType *'http://www.opengis.net/testbed11/ont/incident/ems#EMSIncident* using a CQL constraint

```
(featureType='http://www.opengis.net/testbed11/ont/incident/ems#EMSIncident')
```

```
GET
/styles?constraint=(featureType=%27http%3A%2F%2Fwww.opengis.net%2Ftestbed11%2Font%2Fin
cident%2Fems%23EMSIncident%27) HTTP/1.1
Accept: application/hal+json
Host: localhost
```

Response structure

The response search is structured according the Search Results Schema. The items of the results conforms to the Style JSON Schema. The collection name used in the _ *embedded* section of the HAL response is called *portrayal:items*.

Links

The following link relation types are provided in the response to allow the transition to others states from the Style search results embedded in the response.

Relation	Description
`self`	Refers to this resource itself
`service`	Refers to the root of the portrayal service
`curies`	Refers to the curies defined for the links
`first`	The first page of results
`last`	The last page of results
`next`	The next page of results
`prev`	The previous page of results

Example HAL+JSON Response

The embedded objects in the response of the request conform to the Style JSON Schema. The collection name used in the _ *embedded* section of the HAL response is called *portrayal:items*. The rest of the response returns paging information and links to other states.

```
HTTP/1.1 200 OK
Content-Type: application/hal+json
Content-Length: 2707

{
    "_embedded": {
        "portrayal:items": [{
            "id": "345a19896b477721787a82742cc04770",
            "uri":
"http://www.opengis.net/testbed/12/portrayal/ems/style#EMSIncidentStyle",
            "type": "style:FeatureTypeStyle",
            "title": "EMS Incident Type Style",
            "description": "EMS Incident Type Style",
            "created": "2016-11-18T03:18:12.661Z",
            "modified": "2016-11-18T03:18:15.024Z",
            "_links": {
                "self": {
                    "href":
"http://localhost:8082/styles/345a19896b477721787a82742cc04770"
                }
            }
        }, {
            "id": "c7a30cb460ad27d98fc4ddd303a7c4ab",
            "uri":
"http://www.opengis.net/testbed/12/portrayal/hswg/style#HSWGIncidentStyle",
            "type": "style:FeatureTypeStyle",
            "title": "HSWG Incident Type Style",
            "description": "HSWG Incident Type Style",
            "created": "2016-11-18T03:18:15.457Z",
            "modified": "2016-11-18T03:18:15.842Z",
            "_links": {
                "self": {
                    "href":
"http://localhost:8082/styles/c7a30cb460ad27d98fc4ddd303a7c4ab"
                }
            }
        }]
    },
    "_links": {
        "self": {
            "href": "http://localhost:8082/styles"
        },
        "service": {
            "href": "http://localhost:8082"
        },
        "curies": [{
            "href": "http://www.opengis.net/rels/portrayal/{rel}",
            "name": "portrayal",
            "templated": true
        }]
    },
    "aggregations": [],
    "page": {
        "size": 20,
        "totalElements": 2,
        "totalPages": 1,
        "number": 0
    }
}
```

Example JSON Response

The result objects in the response of the request conforms to the Style JSON Schema. The rest of the response returns paging information.

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: 850

{
    "results": [{
        "id": "345a19896b477721787a82742cc04770",
        "uri":
"http://www.opengis.net/testbed/12/portrayal/ems/style#EMSIncidentStyle",
        "type": "style:FeatureTypeStyle",
        "title": "EMS Incident Type Style",
        "description": "EMS Incident Type Style",
        "created": "2016-11-18T03:18:12.661Z",
        "modified": "2016-11-18T03:18:15.024Z"
    }, {
        "id": "c7a30cb460ad27d98fc4ddd303a7c4ab",
        "uri":
"http://www.opengis.net/testbed/12/portrayal/hswg/style#HSWGIncidentStyle",
        "type": "style:FeatureTypeStyle",
        "title": "HSWG Incident Type Style",
        "description": "HSWG Incident Type Style",
        "created": "2016-11-18T03:18:15.457Z",
        "modified": "2016-11-18T03:18:15.842Z"
    }],
    "page": {
        "size": 20,
        "number": 0,
        "totalElements": 2
    }
}
```

B.12.4.2 Style

The Style resource is used to retrieve a Style instance.

Retrieve a Style

To retrieve a particular instance of Style, a HTTP `GET` request will get the details of a Style. They are two ways to retrieve a instance of a style, using an internal id in the path or by using its Linked Data URL (using uri as query parameter).

Query Parameters

Parameter	Description	Cardinality
uri	URI of the style if the instance needs to be retrieved by URI. URI should be encoded to escape special characters.	0..1

Example request

The following HTTP request performs a GET Request to get an instance of a style identified by its internal id *345a19896b477721787a82742cc04770*

```
GET /styles/345a19896b477721787a82742cc04770 HTTP/1.1
Accept: application/hal+json
Host: localhost
```

The following query gets the instance identified by the URI *http://www.opengis.net/testbed/12/portrayal/ems/style#EMSIncidentStyle*

```
GET
/styles/instance?uri=http%3A%2F%2Fwww.opengis.net%2Ftestbed%2F12%2Fportrayal%2Fems%2Fs
tyle%23EMSIncidentStyle HTTP/1.1
Accept: application/hal+json
Host: localhost
```

Response structure

The response of the instance request can be retrieved in HAL+JSON, JSON-LD and Linked Data formats. The JSON Structure of the reponse conforms to the Style JSON Schema. The Linked Data formats conforms to the Style Ontology.

Links

The following table defines the link relation types accessible from a style instance that provide transitions to other states related to the style instance.

Relation	Description
`self`	Refers to this resource itself
`service`	Refers to the root of the portrayal service
`portrayal:styles`	Refers to the style search endpoint
`curies`	Refers to the curies defined for the links

Example HAL+JSON Format Response (Level 3 REST API)

The HAL+JSON response contains the description of the Style in JSON-LD (which can be converted to Linked Data by applying the JSON-LD context of the service). In addition, it provides links to other states that can be followed by clients following the semantic of the link relation types described in the Style Link section.

```
HTTP/1.1 200 OK
Content-Type: application/hal+json
Content-Length: 1895

{
    "id": "345a19896b477721787a82742cc04770",
    "uri": "http://www.opengis.net/testbed/12/portrayal/ems/style#EMSIncidentStyle",
    "type": "style:FeatureTypeStyle",
    "title": "EMS Incident Type Style",
    "description": "EMS Incident Type Style",
    "created": "2016-11-18T03:18:12.661Z",
    "modified": "2016-11-18T03:18:15.024Z",
    "featureType": "http://www.opengis.net/testbed11/ont/incident/ems#EMSIncident",
    "rules": [

"http://www.opengis.net/testbed/12/portrayal/ems/style#ems.incident.wind.strongWind-
portrayal-rule",

"http://www.opengis.net/testbed/12/portrayal/ems/style#ems.incident.geophysical.magnet
icStorm-portrayal-rule",

"http://www.opengis.net/testbed/12/portrayal/ems/style#ems.incident.crime.bombThreat-
portrayal-rule",

"http://www.opengis.net/testbed/12/portrayal/ems/style#ems.incident.hazardousMaterial.
biologicalHazard-portrayal-rule",
        .... truncated ....

"http://www.opengis.net/testbed/12/portrayal/ems/style#ems.incident.geophysical.earthq
uake-portrayal-rule",

"http://www.opengis.net/testbed/12/portrayal/ems/style#ems.incident.crime.bombExplosio
n-portrayal-rule"
    ],
    "_links": {
        "self": {
            "href": "http://localhost:8082/styles/345a19896b477721787a82742cc04770"
        },
        "portrayal:styles": {
            "href": "http://localhost:8082/styles"
        },
        "service": {
            "href": "http://localhost:8082"
        },
        "curies": [{
            "href": "http://www.opengis.net/rels/portrayal/{rel}",
            "name": "portrayal",
            "templated": true
        }]
    }
}
```

Example JSON(-LD) Format Response (Level 2 REST API)

The following HTTP request performs a GET Request to get an instance of a Style in JSON format.

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: 1495

{
    "id": "345a19896b477721787a82742cc04770",
    "uri": "http://www.opengis.net/testbed/12/portrayal/ems/style#EMSIncidentStyle",
    "type": "style:FeatureTypeStyle",
    "title": "EMS Incident Type Style",
    "description": "EMS Incident Type Style",
    "created": "2016-11-18T03:18:12.661Z",
    "modified": "2016-11-18T03:18:15.024Z",
    "featureType": "http://www.opengis.net/testbed11/ont/incident/ems#EMSIncident",
    "rules": [

"http://www.opengis.net/testbed/12/portrayal/ems/style#ems.incident.wind.strongWind-
portrayal-rule",

"http://www.opengis.net/testbed/12/portrayal/ems/style#ems.incident.geophysical.magnet
icStorm-portrayal-rule",

"http://www.opengis.net/testbed/12/portrayal/ems/style#ems.incident.crime.bombThreat-
portrayal-rule",

"http://www.opengis.net/testbed/12/portrayal/ems/style#ems.incident.hazardousMaterial.
biologicalHazard-portrayal-rule",
        .... truncated ....

"http://www.opengis.net/testbed/12/portrayal/ems/style#ems.incident.geophysical.earthq
uake-portrayal-rule",

"http://www.opengis.net/testbed/12/portrayal/ems/style#ems.incident.crime.bombExplosio
n-portrayal-rule"
    ]
}
```

The JSON-(LD) response is identical to the HAL+JSON except it does not have the hyperlinks to other states. The client will need to build the URL to reach other states (by reading documentation of API).

Example Turtle Format Response (Linked Data API)

The following HTTP request performs a GET Request to get an instance of a Style in Turtle format.

```
GET /styles/345a19896b477721787a82742cc04770 HTTP/1.1
Accept: text/turtle
Host: localhost
```

The response returns a TTL document that can be processed and interpreted by machines using the

Style ontology.

```
HTTP/1.1 200 OK
Content-Type: text/turtle
Content-Length: 10395

@prefix dcterms: <http://purl.org/dc/terms/> .
@prefix style: <http://www.opengis.net/ont/portrayal/style#> .

<http://www.opengis.net/testbed/12/portrayal/ems/style#EMSIncidentStyle>
        a                    style:FeatureTypeStyle ;
        dcterms:created      "2016-11-
18T03:18:12.661Z"^^<http://www.w3.org/2001/XMLSchema#dateTime> ;
        dcterms:description  "EMS Incident Type Style" ;
        dcterms:modified     "2016-11-
18T03:18:15.024Z"^^<http://www.w3.org/2001/XMLSchema#dateTime> ;
        dcterms:title        "EMS Incident Type Style" ;
        style:featureType
<http://www.opengis.net/testbed11/ont/incident/ems#EMSIncident> ;
        style:hasRule

<http://www.opengis.net/testbed/12/portrayal/ems/style#ems.incident.wind.strongWind-
portrayal-rule> ,

<http://www.opengis.net/testbed/12/portrayal/ems/style#ems.incident.geophysical.magnet
icStorm-portrayal-rule> ,

<http://www.opengis.net/testbed/12/portrayal/ems/style#ems.incident.crime.bombThreat-
portrayal-rule> ,

        .... truncated.....


<http://www.opengis.net/testbed/12/portrayal/ems/style#ems.incident.crime.bombExplosio
n-portrayal-rule> .
```

B.12.4.3 Symbols

A **Symbol** defines a graphic representation of a feature. The **Symbols resource** is used to search symbol instances based on some search criteria.

JSON Schema

The following defines the JSON schema for the Symbol derived from the Symbol Ontology defined in Appendix A. The JSON-LD Context of the service can be used to convert the JSON document compliant to this schema to the Linked Data Representation.

Path	Type	Description	Cardinality
`id`	`String`	Internal identifier for the symbol (which can be used in Level 2 API)	1
`uri`	`String`	Linked Data URI for the Symbol (equivalent to @id in JSON-LD)	0..1
`type`	`String`	The type (class) of the Symbol (symbol:Symbol or (future) subclasses).	1
`title`	`String`	The title of the symbol	1
`titleMap`	`Object`	The title map for each language of the title . Each key corresponds to the two letter language identifier name (for example "en")	0..1
`description`	`String`	The description of the item	0..1
`descriptionMap`	`Object`	The description map for each language of the description. Each key corresponds to the two letter language identifier name (for example "en")	0..1
`created`	`String`	The date of creation (XSD datetime format) (generated by the service)	1
`modified`	`String`	The date of the last modification (XSD datetime format) (generated by the service)	0..1
symbolizers[]	Array	Array of symbolizers used to render the symbol. See ontology for Symbolizer in Appendix A	1

Search Symbols

The search of the Symbols is performed by performing a HTTP `GET` request on the Symbols resource. The symbol search supports free text, by ids, by uris, CQL constraint and can return results with aggregations on specific facets.

Query Parameters

The following query parameters are supported in the query:

Parameter	Description	Cardinality
`q`	Text to search in textual fields	0..1
`uri`	One or more URIs of Symbol instances	0..n
`id`	One or more id of Symbol instances	0..n
`includeFacet`	Boolean or list of facet names to include for aggregation computation. If the value is true, include all facets supported by the server. If only a subset of the facets are needed, a comma delimited of field names can be set.	0..1
`facet.fieldname`	Constraint values of a given facet field name	0..n
`constraint`	A constraint expressed in CQL. This is used to express more advanced query filtering	0..1
`fields`	One or more fields to be included in the response. Use JSON path dot notation for referring paths	0..n
`pageNumber`	The number of the current page as defined in Paging section	0..1
`pageSize`	The count of items on the current page as defined in Paging section	0..1
`sort`	The sorting parameters as defined in Sorting section.	0..n

Example request

The following performs a symbol search containing the keyword 'Rail'.

```
GET /symbols?q=Rail HTTP/1.1
Accept: application/hal+json
Host: localhost
```

Response structure

The response search is structured according to the Search Results Schema. The items of the results conforms to the Symbol JSON Schema. The collection name used in the _ *embedded* section of the HAL response is called *portrayal:items*.

Links

The following link relation types are provided in the response to allow the transition to others states from the symbol search results embedded in the response.

Relation	Description
`self`	Refers to this resource itself
`service`	Refers to the root of the portrayal service
`curies`	Refers to the curies defined for the links

Relation	Description
`first`	The first page of results
`last`	The last page of results
`next`	The next page of results
`prev`	The previous page of results

Example HAL+JSON Response

The embedded objects in the response of the request conforms to the Symbol JSON Schema. The collection name used in the _ *embedded* section of the HAL response is called *portrayal:items*. The rest of the response returns paging information and links to other states.

```
HTTP/1.1 200 OK
Content-Type: application/hal+json
Content-Length: 2707

{
    "_embedded": {
        "portrayal:items": [{
            "id": "41f4b3ed5976016d071cf30498bf6358",
            "uri":
"http://www.opengis.net/testbed/12/hswg/symbols#RailHijackingSymbol",
            "type": "symbol:Symbol",
            "title": "Rail Hijacking",
            "created": "2016-11-17T15:41:35.436Z",
            "modified": "2016-11-17T15:41:35.951Z",
            "_links": {
                "self": {
                    "href":
"http://localhost:8082/symbols/41f4b3ed5976016d071cf30498bf6358"
                }
            }
        }, {
            "id": "d6c8d924a629514d322bb25f563f8948",
            "uri":
"http://www.opengis.net/testbed/12/hswg/symbols#RailAccidentSymbol",
            "type": "symbol:Symbol",
            "title": "Rail Accident",
            "created": "2016-11-17T15:42:18.602Z",
            "modified": "2016-11-17T15:42:18.717Z",
            "_links": {
                "self": {
                    "href":
"http://localhost:8082/symbols/d6c8d924a629514d322bb25f563f8948"
                }
            }
        }, {
            "id": "d6792c5cb00daa3b9e2c161846cc7ef4",
            "uri":
```

```
"http://www.opengis.net/testbed/12/hswg/symbols#RailIncidentSymbol",
            "type": "symbol:Symbol",
            "title": "Rail Incident",
            "created": "2016-11-17T15:42:11.281Z",
            "modified": "2016-11-17T15:42:11.712Z",
            "_links": {
                "self": {
                    "href":
"http://localhost:8082/symbols/d6792c5cb00daa3b9e2c161846cc7ef4"
                }
            }
        }]
    },
    "_links": {
        "self": {
            "href": "http://localhost:8082/symbols?q=Rail"
        },
        "service": {
            "href": "http://localhost:8082"
        },
        "curies": [{
            "href": "http://www.opengis.net/rels/portrayal/{rel}",
            "name": "portrayal",
            "templated": true
        }]
    },
    "aggregations": [],
    "page": {
        "size": 20,
        "totalElements": 3,
        "totalPages": 1,
        "number": 0
    }
}
```

Example JSON Response

The result objects in the response of the request conforms to the Symbol JSON Schema. The rest of the response returns paging information.

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: 1007

{
    "results": [{
        "id": "41f4b3ed5976016d071cf30498bf6358",
        "uri": "http://www.opengis.net/testbed/12/hswg/symbols#RailHijackingSymbol",
        "type": "symbol:Symbol",
        "title": "Rail Hijacking",
        "created": "2016-11-17T15:41:35.436Z",
        "modified": "2016-11-17T15:41:35.951Z"
    }, {
        "id": "d6c8d924a629514d322bb25f563f8948",
        "uri": "http://www.opengis.net/testbed/12/hswg/symbols#RailAccidentSymbol",
        "type": "symbol:Symbol",
        "title": "Rail Accident",
        "created": "2016-11-17T15:42:18.602Z",
        "modified": "2016-11-17T15:42:18.717Z"
    }, {
        "id": "d6792c5cb00daa3b9e2c161846cc7ef4",
        "uri": "http://www.opengis.net/testbed/12/hswg/symbols#RailIncidentSymbol",
        "type": "symbol:Symbol",
        "title": "Rail Incident",
        "created": "2016-11-17T15:42:11.281Z",
        "modified": "2016-11-17T15:42:11.712Z"
    }],
    "page": {
        "size": 20,
        "number": 0,
        "totalElements": 3
    }
}
```

B.12.4.4 Symbol

The Symbol resource is used to retrieve a Symbol instance.

Retrieve a Symbol

To retrieve a particular instance of Symbol, a HTTP `GET` request will get the details of a Symbol. There are two ways to retrieve a instance of a schema, using an internal id in the path or by using its Linked Data URL (using uri as query parameter).

Query Parameters

Parameter	Description	Cardinality
uri	URI of the symbol if the instance needs to be retrieved by URI. URI should be encoded to escape special characters.	0..1

Example request

The following HTTP request performs a GET Request to get an instance of a symbol identified by its internal id *06aab0337e96ea6e2535a2620b8e9a90*

```
GET /symbols/06aab0337e96ea6e2535a2620b8e9a90 HTTP/1.1
Accept: application/hal+json
Host: localhost
```

The following query gets the instance identified by the URI *http://www.opengis.net/testbed/12/ems/symbols#ems.incident.wind-symbol*

```
GET
/symbols/instance?uri=http%3A%2F%2Fwww.opengis.net%2Ftestbed%2F12%2Fems%2Fsymbols%23em
s.incident.wind-symbol HTTP/1.1
Accept: application/hal+json
Host: localhost
```

Response structure

The response of the instance request can be retrieved in HAL+JSON, JSON-LD and Linked Data formats. The JSON Structure of the reponse conforms to the Symbol JSON Schema. The Linked Data formats conform to the Symbol Ontology.

Links

The following table defines the link relation types accessible from a symbol instance that provide transitions to other states related to the symbol instance.

Relation	Description
`self`	Refers to this resource itself
`service`	Refers to the root of the portrayal service
`portrayal:symbols`	Refers to the symbols search endpoint
`portrayal:renderer`	Refers to the renderer of the symbol.
`curies`	Refers to the curies defined for the links

Example HAL+JSON Format Response (Level 3 REST API)

The HAL+JSON response contains the description of the Symbol in JSON-LD (which can be converted to Linked Data by applying the JSON-LD context of the service). In addition, it provides links to other states that can be followed by clients following the semantic of the link relation types described in the Symbol Link section.

```
HTTP/1.1 200 OK
Content-Type: application/hal+json
Content-Length: 1895

{
    "id": "06aab0337e96ea6e2535a2620b8e9a90",
    "uri": "http://www.opengis.net/testbed/12/ems/symbols#ems.incident.wind-symbol",
    "type": "symbol:Symbol",
    "register": [
        "portrayal"
    ],
    "title": "wind",
    "created": "2016-11-17T15:39:40.261Z",
    "modified": "2016-11-17T15:39:40.711Z",
    "symbolName": "ems.incident.wind",
    "specification": "https://cms.masas-x.ca.s3.amazonaws.com/EMS_Symbology_v1.0.pdf",
    "symbolSet": "http://www.opengis.net/testbed/12/ems/symbols#EMSSymbolSet",
    "browseGraphic": {
        "type": "graphic:ExternalGraphic",
        "uri": "http://ows.usersmarts.com/ems/icons/tier1/Base/ems.incident.wind.png",
        "title": "ems.incident.wind icon",
        "description": "icon for ems.incident.wind",
        "onlineResource":
"http://ows.usersmarts.com/ems/icons/tier1/Base/ems.incident.wind.png",
        "format": "image/png"
    },
    "symbolizers": [{
        "uri": "http://www.opengis.net/testbed/12/ems/symbols#ems.incident.wind-
pointSymbolizer",
        "type": "symbolizer:PointSymbolizer",
        "title": "PointSymbolizer for wind symbol",
        "graphicSymbol": {
            "type": "graphic:GraphicSymbol",
            "externalGraphic": {
                "type": "graphic:ExternalGraphic",
                "uri":
"http://ows.usersmarts.com/ems/icons/tier1/Base/ems.incident.wind.png",
                "title": "ems.incident.wind icon",
                "description": "icon for ems.incident.wind",
                "onlineResource":
"http://ows.usersmarts.com/ems/icons/tier1/Base/ems.incident.wind.png",
                "format": "image/png"
            }
        }
    }],
    "denotes": [
        "http://www.opengis.net/taxonomy/ems#ems.incident.wind"
    ],
    "_links": {
        "self": {
            "href": "http://localhost:8082/symbols/06aab0337e96ea6e2535a2620b8e9a90"
        },
        "portrayal:render": {
            "href":
"http://localhost:8082/symbols/06aab0337e96ea6e2535a2620b8e9a90/render"
        },
        "portrayal:symbols": {
            "href": "http://localhost:8082/symbols"
        },
        "service": {
            "href": "http://localhost:8082"
        },
        "curies": [{
            "href": "http://www.opengis.net/rels/portrayal/{rel}",
            "name": "portrayal",
            "templated": true
        }]
    }
}
```

Example JSON(-LD) Format Response (Level 2 REST API)

The following HTTP request performs a GET Request to get an instance of a Symbol in JSON format.

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: 1495

{
    "id": "06aab0337e96ea6e2535a2620b8e9a90",
    "uri": "http://www.opengis.net/testbed/12/ems/symbols#ems.incident.wind-symbol",
    "type": "symbol:Symbol",
    "register": [
        "portrayal"
    ],
    "title": "wind",
    "created": "2016-11-17T15:39:40.261Z",
    "modified": "2016-11-17T15:39:40.711Z",
    "symbolName": "ems.incident.wind",
    "specification": "https://cms.masas-x.ca.s3.amazonaws.com/EMS_Symbology_v1.0.pdf",
    "symbolSet": "http://www.opengis.net/testbed/12/ems/symbols#EMSSymbolSet",
    "browseGraphic": {
        "type": "graphic:ExternalGraphic",
        "uri": "http://ows.usersmarts.com/ems/icons/tier1/Base/ems.incident.wind.png",
        "title": "ems.incident.wind icon",
        "description": "icon for ems.incident.wind",
        "onlineResource":
"http://ows.usersmarts.com/ems/icons/tier1/Base/ems.incident.wind.png",
        "format": "image/png"
    },
    "symbolizers": [{
        "uri": "http://www.opengis.net/testbed/12/ems/symbols#ems.incident.wind-
pointSymbolizer",
        "type": "symbolizer:PointSymbolizer",
        "title": "PointSymbolizer for wind symbol",
        "graphicSymbol": {
            "type": "graphic:GraphicSymbol",
            "externalGraphic": {
                "type": "graphic:ExternalGraphic",
                "uri":
"http://ows.usersmarts.com/ems/icons/tier1/Base/ems.incident.wind.png",
                "title": "ems.incident.wind icon",
                "description": "icon for ems.incident.wind",
                "onlineResource":
"http://ows.usersmarts.com/ems/icons/tier1/Base/ems.incident.wind.png",
                "format": "image/png"
            }
        }
    }],
    "denotes": [
        "http://www.opengis.net/taxonomy/ems#ems.incident.wind"
    ]
}
```

The JSON-(LD) response is identical to the HAL+JSON except it does not have the hyperlinks to other states. The client will need to build the URL to reach other states (by reading documentation of API).

Example Turtle Format Response (Linked Data API)

The following HTTP request performs a GET Request to get an instance of a Symbol in Turtle format.

```
GET /symbols/06aab0337e96ea6e2535a2620b8e9a90 HTTP/1.1
Accept: text/turtle
Host: localhost
```

The response returns a TTL document that can be processed and interpreted by machines using the Portrayal ontology.

```
HTTP/1.1 200 OK
Content-Type: text/turtle
Content-Length: 1395

@prefix symbol: <http://www.opengis.net/ont/portrayal/symbol#> .
@prefix dcterms: <http://purl.org/dc/terms/> .

<http://www.opengis.net/testbed/12/ems/symbols#ems.incident.wind-symbol>
        a                     symbol:Symbol ;
        dcterms:title         "wind" ;
        symbol:browseGraphic
<http://ows.usersmarts.com/ems/icons/tier1/Base/ems.incident.wind.png> ;
        symbol:specification  <https://cms.masas-
x.ca.s3.amazonaws.com/EMS_Symbology_v1.0.pdf> ;
        symbol:symbolName     "ems.incident.wind" ;
        symbol:symbolSet
<http://www.opengis.net/testbed/12/ems/symbols#EMSSymbolSet> ;
        <http://www.opengis.net/ont/portrayal/symbolizer#symbolizer>
                <http://www.opengis.net/testbed/12/ems/symbols#ems.incident.wind-
pointSymbolizer> .

<http://ows.usersmarts.com/ems/icons/tier1/Base/ems.incident.wind.png>
        a
<http://www.opengis.net/ont/portrayal/graphic#ExternalGraphic> ;
        dcterms:description  "icon for ems.incident.wind" ;
        dcterms:title        "ems.incident.wind icon" ;
        <http://www.opengis.net/ont/portrayal/graphic#format>
                "image/png" ;
        <http://www.opengis.net/ont/portrayal/graphic#onlineResource>
                "http://ows.usersmarts.com/ems/icons/tier1/Base/ems.incident.wind.png"
.

<http://www.opengis.net/testbed/12/ems/symbols#ems.incident.wind-pointSymbolizer>
        a
<http://www.opengis.net/ont/portrayal/symbolizer#PointSymbolizer> ;
        dcterms:title  "PointSymbolizer for wind symbol" ;
        <http://www.opengis.net/ont/portrayal/graphic#graphicSymbol>
                [ a       <http://www.opengis.net/ont/portrayal/graphic#GraphicSymbol>
;
                  <http://www.opengis.net/ont/portrayal/graphic#externalGraphic>

<http://ows.usersmarts.com/ems/icons/tier1/Base/ems.incident.wind.png>
                ] .
```

B.12.4.5 SymbolSets

A **SymbolSet** defines a set of Symbols. The **SymbolSets resource** is used to search SymbolSet instances based on some search criteria.

JSON Schema

The following defines the JSON schema for the SymbolSet derived from the Symbol Ontology defined in Appendix A. It can also be derived from the JSON-LD Context

Path	Type	Description	Card.
`id`	`String`	Internal identifier for the SymbolSet (which can be used in Level 2 API)	1
`uri`	`String`	Linked Data URI for the SymbolSet (equivalent to @id in JSON-LD)	0..1
`type`	`String`	The type (class) of the SymbolSet (symbol:SymbolSet).	1
`title`	`String`	The title of the symbol	1
`titleMap`	`Object`	The title map for each language of the title . Each key corresponds to the two letter language identifier name (for example "en")	0..1
`description`	`String`	The description of the item	0..1
`descriptionMap`	`Object`	The description map for each language of the description. Each key corresponds to the two letter language identifier name (for example "en")	0..1
`created`	`String`	The date of creation (XSD datetime format) (generated by the service)	1
`modified`	`String`	The date of the last modification (XSD datetime format) (generated by the service)	0..1
`specification`	`URL`	The reference to a specification document	0..1
symbols[]	Array	Array of symbols URIs.	0..1

Search SymbolSets

The search of the SymbolSets is performed by performing a HTTP `GET` request on the SymbolSets resource. The symbolSet search supports free text, by ids, by uris, CQL constraint and can return results with aggregations on specific facets.

Query Parameters

The following query parameters are supported in the query:

Parameter	Description	Cardinality
`q`	Text to search in textual fields	0..1
`uri`	One or more URIs of Symbol instances	0..n
`id`	One or more id of Symbol instances	0..n
`includeFacet`	Boolean or list of facet names to include for aggregation computation. If the value is true, include all facets supported by the server. If only a subset of the facets are needed, a comma delimited of field names can be set.	0..1
`facet.fieldname`	Constraint values of a given facet field name	0..n
`constraint`	A constraint expressed in CQL. This is used to express more advanced query filtering	0..1
`fields`	One or more fields to be included in the response. Use JSON path dot notation for referring paths	0..n
`pageNumber`	The number of the current page as defined in Paging section	0..1
`pageSize`	The count of items on the current page as defined in Paging section	0..1
`sort`	The sorting parameters as defined in Sorting section.	0..n

Example request

The following performs a symbolSet search containing the keyword 'Rail'.

```
GET /symbolsets HTTP/1.1
Accept: application/hal+json
Host: localhost
```

Response structure

The response search is structured according the Search Results Schema. The items of the results conform to the SymbolSet JSON Schema. The collection name used in the _ *embedded* section of the HAL response is called *portrayal:items*.

Links

The following link relation types are provided in the response to allow the transition to others states from the symbolSet search results embedded in the response.

Relation	Description
self	Refers to this resource itself
service	Refers to the root of the portrayal service
curies	Refers to the curies defined for the links
first	The first page of results
last	The last page of results
next	The next page of results
prev	The previous page of results

Example HAL+JSON Response

The embedded objects in the response of the request conforms to the SymbolSet JSON Schema. The collection name used in the _ *embedded* section of the HAL response is called *portrayal:items*. The rest of the response returns paging information and links to other states.

```
HTTP/1.1 200 OK
Content-Type: application/hal+json
Content-Length: 2707

{
    "_embedded": {
        "portrayal:items": [{
            "id": "4cabf46007df9b652151927033e265c6",
            "uri": "http://www.opengis.net/testbed/12/ems/symbols#EMSSymbolSet",
            "type": "symbol:SymbolSet",
            "description": "Standard Canadian Emergency Mapping Symbology (EMS)
SymbolSet version 1.0",
            "created": "2016-11-17T15:38:35.856Z",
            "modified": "2016-11-17T15:38:36.457Z",
            "_links": {
                "self": {
                    "href":
"http://localhost:8082/symbolsets/4cabf46007df9b652151927033e265c6"
                }
            }
        }, {
            "id": "bf83c8b6f4033ffd4c93a2be14ebe476",
            "uri": "http://www.opengis.net/testbed/12/hswg/symbols#HSWGSymbolSet",
            "type": "symbol:SymbolSet",
            "description": "Home Security Working Group (HSWG) SymbolSet version 1.0",
            "created": "2016-11-17T15:38:37.357Z",
            "modified": "2016-11-17T15:38:37.786Z",
            "_links": {
                "self": {
                    "href":
"http://localhost:8082/symbolsets/bf83c8b6f4033ffd4c93a2be14ebe476"
                }
            }
        }]
    },
    "_links": {
        "self": {
            "href": "http://localhost:8082/symbolsets"
        },
        "service": {
            "href": "http://localhost:8082"
        },
        "curies": [{
            "href": "http://www.opengis.net/rels/portrayal/{rel}",
            "name": "portrayal",
            "templated": true
        }]
    },
    "aggregations": [],
    "page": {
        "size": 20,
        "totalElements": 2,
        "totalPages": 1,
        "number": 0
    }
}
```

Example JSON Response

The result objects in the response of the request conforms to the SymbolSet JSON Schema. The rest of the response returns paging information.

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: 850

{
    "results": [{
        "id": "4cabf46007df9b652151927033e265c6",
        "uri": "http://www.opengis.net/testbed/12/ems/symbols#EMSSymbolSet",
        "type": "symbol:SymbolSet",
        "description": "Standard Canadian Emergency Mapping Symbology (EMS) SymbolSet
version 1.0",
        "created": "2016-11-17T23:13:47.142Z",
        "modified": "2016-11-17T23:13:48.534Z"
    }, {
        "id": "bf83c8b6f4033ffd4c93a2be14ebe476",
        "uri": "http://www.opengis.net/testbed/12/hswg/symbols#HSWGSymbolSet",
        "type": "symbol:SymbolSet",
        "description": "Home Security Working Group (HSWG) SymbolSet version 1.0",
        "created": "2016-11-17T23:13:50.354Z",
        "modified": "2016-11-17T23:13:50.922Z"
    }],
    "page": {
        "size": 20,
        "number": 0,
        "totalElements": 2
    }
}
```

B.12.4.6 SymbolSet

The SymbolSet resource is used to retrieve a SymbolSet instance.

Retrieve a SymbolSet

To retrieve a particular instance of SymbolSet, a HTTP `GET` request will get the details of a SymbolSet. They are two ways to retrieve an instance of a SymbolSet, using an internal id in the path or by using its Linked Data URL (using uri as query parameter).

Query Parameters

Parameter	Description	Cardinality
uri	URI of the symbol if the instance needs to be retrieved by URI. URI should be encoded to escape special characters.	0..1

Example request

The following HTTP request performs a GET Request to get an instance of a symbol identified by its internal id *4cabf46007df9b652151927033e265c6*

```
GET /symbolsets/4cabf46007df9b652151927033e265c6 HTTP/1.1
Accept: application/hal+json
Host: localhost
```

The following query gets the instance identified by the URI *http://www.opengis.net/testbed/12/ems/symbols#EMSSymbolSet*

```
GET
/symbolsets/instance?uri=http%3A%2F%2Fwww.opengis.net%2Ftestbed%2F12%2Fems%2Fsymbols%2
3EMSSymbolSet HTTP/1.1
Accept: application/hal+json
Host: localhost
```

Response structure

The response of the instance request can be retrieved in HAL+JSON, JSON-LD and Linked Data formats. The JSON Structure of the response conforms to the SymbolSet JSON Schema. The Linked Data formats conforms to the Symbol Ontology.

Links

The following table defines the link relation types accessible from a symbolSet instance that provide transitions to other states related to the symbolSet instance.

Relation	Description
`self`	Refers to this resource itself
`service`	Refers to the root of the portrayal service
`portrayal:symbolSets`	Refers to the symbols search endpoint
`curies`	Refers to the curies defined for the links

Example HAL+JSON Format Response (Level 3 REST API)

The HAL+JSON response contains the description of the SymbolSet in JSON-LD (which can be converted to Linked Data by applying the JSON-LD context of the service). In addition, it provides links to other states that can be followed by clients following the semantic of the link relation types described in the SymbolSet Link section.

```
HTTP/1.1 200 OK
Content-Type: application/hal+json
Content-Length: 1895

{
    "id": "4cabf46007df9b652151927033e265c6",
    "uri": "http://www.opengis.net/testbed/12/ems/symbols#EMSSymbolSet",
    "type": "symbol:SymbolSet",
    "register": [
        "portrayal"
    ],
    "description": "Standard Canadian Emergency Mapping Symbology (EMS) SymbolSet
version 1.0",
    "created": "2016-11-17T15:38:35.856Z",
    "modified": "2016-11-17T15:38:36.457Z",
    "specification": "https://cms.masas-x.ca.s3.amazonaws.com/EMS_Symbology_v1.0.pdf",
    "symbols": [
        "http://www.opengis.net/testbed/12/ems/symbols#ems.incident.temperature-
symbol",

"http://www.opengis.net/testbed/12/ems/symbols#ems.incident.railway.railwayAccident-
symbol",

"http://www.opengis.net/testbed/12/ems/symbols#ems.incident.crime.industrialCrime-
symbol",

"http://www.opengis.net/testbed/12/ems/symbols#ems.incident.hazardousMaterial.infectio
usDisease-symbol",

"http://www.opengis.net/testbed/12/ems/symbols#ems.incident.wind.hurricaneForceWind-
symbol",

"http://www.opengis.net/testbed/12/ems/symbols#ems.incident.missingPerson.silver-
symbol",

"http://www.opengis.net/testbed/12/ems/symbols#ems.incident.hazardousMaterial.poisonou
sGas-symbol",

"http://www.opengis.net/testbed/12/ems/symbols#ems.incident.meteorological.hurricane-
symbol",
        .... (truncated) ....
        "http://www.opengis.net/testbed/12/ems/symbols#ems.incident.crime.retailCrime-
symbol",

"http://www.opengis.net/testbed/12/ems/symbols#ems.incident.animalHealth.animalDieOff-
symbol"
    ],
    "_links": {
        "self": {
            "href":
"http://localhost:8082/symbolsets/4cabf46007df9b652151927033e265c6"
        },
        "portrayal:symbolSets": {
            "href": "http://localhost:8082/symbolsets"
        },
        "service": {
            "href": "http://localhost:8082"
        },
        "curies": [{
            "href": "http://www.opengis.net/rels/portrayal/{rel}",
            "name": "portrayal",
            "templated": true
        }]
    }
}
```

Example JSON(-LD) Format Response (Level 2 REST API)

The following HTTP request performs a GET Request to get an instance of a Symbol in JSON format.

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: 1495

{
    "id": "4cabf46007df9b652151927033e265c6",
    "uri": "http://www.opengis.net/testbed/12/ems/symbols#EMSSymbolSet",
    "type": "symbol:SymbolSet",
    "register": [
        "portrayal"
    ],
    "description": "Standard Canadian Emergency Mapping Symbology (EMS) SymbolSet
version 1.0",
    "created": "2016-11-17T15:38:35.856Z",
    "modified": "2016-11-17T15:38:36.457Z",
    "specification": "https://cms.masas-x.ca.s3.amazonaws.com/EMS_Symbology_v1.0.pdf",
    "symbols": [
        "http://www.opengis.net/testbed/12/ems/symbols#ems.incident.temperature-
symbol",

"http://www.opengis.net/testbed/12/ems/symbols#ems.incident.railway.railwayAccident-
symbol",

"http://www.opengis.net/testbed/12/ems/symbols#ems.incident.crime.industrialCrime-
symbol",

"http://www.opengis.net/testbed/12/ems/symbols#ems.incident.hazardousMaterial.infectio
usDisease-symbol",

"http://www.opengis.net/testbed/12/ems/symbols#ems.incident.wind.hurricaneForceWind-
symbol",

"http://www.opengis.net/testbed/12/ems/symbols#ems.incident.missingPerson.silver-
symbol",

"http://www.opengis.net/testbed/12/ems/symbols#ems.incident.hazardousMaterial.poisonou
sGas-symbol",

"http://www.opengis.net/testbed/12/ems/symbols#ems.incident.meteorological.hurricane-
symbol",
         .... (truncated) ....
        "http://www.opengis.net/testbed/12/ems/symbols#ems.incident.crime.retailCrime-
symbol",

"http://www.opengis.net/testbed/12/ems/symbols#ems.incident.animalHealth.animalDieOff-
symbol"
    ]
}
```

The JSON-(LD) response is identical to the HAL+JSON except it does not have the hyperlinks to other states. The client will need to build the URL to reach other states (by reading documentation of API).

Example Turtle Format Response (Linked Data API)

The following HTTP request performs a GET Request to get an instance of a SymbolSet in Turtle format.

```
GET /symbolsets/4cabf46007df9b652151927033e265c6 HTTP/1.1
Accept: text/turtle
Host: localhost
```

The response returns a TTL document that can be processed and interpreted by machines using the Portrayal ontology.

```
HTTP/1.1 200 OK
Content-Type: text/turtle
Content-Length: 1395

<http://www.opengis.net/testbed/12/ems/symbols#EMSSymbolSet>
        a       <http://www.opengis.net/ont/portrayal/symbol#SymbolSet> ;
        <http://purl.org/dc/terms/description>
                "Standard Canadian Emergency Mapping Symbology (EMS) SymbolSet version
1.0" ;
        <http://www.opengis.net/ont/portrayal/symbol#specification>
                "https://cms.masas-x.ca.s3.amazonaws.com/EMS_Symbology_v1.0.pdf" ;
        <http://www.opengis.net/ont/portrayal/symbol#symbol>

<http://www.opengis.net/testbed/12/ems/symbols#ems.incident.temperature-symbol> ,
<http://www.opengis.net/testbed/12/ems/symbols#ems.incident.railway.railwayAccident-
symbol> ,
<http://www.opengis.net/testbed/12/ems/symbols#ems.incident.crime.industrialCrime-
symbol>,
                .... truncated ...

<http://www.opengis.net/testbed/12/ems/symbols#ems.incident.crime.retailCrime-symbol>
,
<http://www.opengis.net/testbed/12/ems/symbols#ems.incident.animalHealth.animalDieOff-
symbol> .
```

B.12.4.7 Render a Layer

To render a particular dataset located at a URL, a HTTP `GET` request can be performed on the renderer endpoint. The endpoint provides the ability to customize the style and size of the map layer image provided in the response.

Render GET Request Query Parameters

Parameter	Type	Description	Cardinality
id	String	Identifier of the layer managed by the service (optional if part of path)	0..1
url	String	URL to access source geospatial data	0..1
bbox	String	Bounding box of geospatial dagiventa to render in the given CRS	1
style	String	Identifier of the style to render with (id). If this parameter is present, the **symbolizer** and **propertyName** fields must not be.	0..1
symbolizer	String	Comma-separated list of symbolizer identifiers. If this parameter is present the **style** parameter must not be present.	0..1
propertyName	String	Comma-separated list of propertyNames in the format *{namespace}name* (or simply *name*). If this parameter is present, the **symbolizer** parameter must also be present (and of the same list length) and the **style** parameter must not be present.	0..1
crs	String	Target Coordinate Reference System (default is CRS:84)	0..1
width	integer	Width in pixels of the rendered map layer	0..1
height	integer	Height in pixels of the rendered map layer	0..1

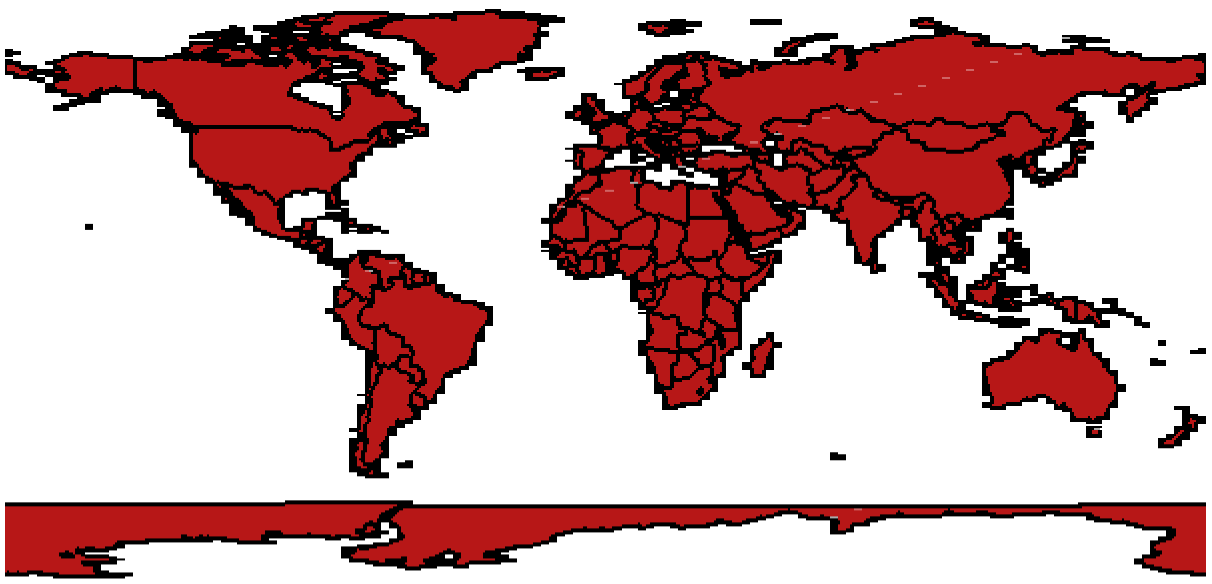
Example GET Request

The following gets a GeoJSON file from a URL and renders it to a map layer using a style stored in the portrayal service.

```
GET
/renderer?url=https://raw.githubusercontent.com/johan/world.geo.json/master/countries.
geo.json&sourceType=geojson&featureType=featureType&namespace&style=4decb2fa8cf327de8c
6cbe6806d653f0&bbox=-180,-90,180,90&crs=CRS:84&width=600 HTTP/1.1
Accept: image/png
Host: localhost
```

Example Response

The service responds with a PNG image depicting the desired featureType in the requested style.



Alternatively, a HTTP `POST` request can be performed on the layer renderer endpoint.

If a dataset from a URL is to be used, the body of the `POST` should be a JSON payload of type **Layer** (as defined in Appendix A). If a dataset from an attached file is to be used, the body of the `POST` should contain both a JSON **Layer** and the attached file.

Render POST Request Parameters

Parameter	Type	Description	Cardinality
request	Layer	The user-defined Layer description (see Layer model in Appendix A)	1
file	File	File containing geospatial data in a format supported by the renderer	0..1

B.12.4.8 Symbol Renderer

The Symbol Renderer resource is used to render a symbol into a graphic representation such as SVG, PNG, TIFF, JPEG using its symbolizer definitions. This endpoint provides the ability to render the symbol into different formats and size.

Render a Symbol

To render a particular instance of Symbol, a HTTP `GET` request is performed on the symbol renderer endpoint. The request should accept one of the supported graphic formats advertised in the capabilities by the service. The endpoint provides the ability to customize the size of the graphic returned in the response.

Query Parameters

Parameter	Type	Description	Cardinality
id	String	Identifier of the symbol (optional if symbol id part the REST path)	0..1
uri	String	URI of the symbol (only for generic renderer endpoint). URL can be retrieved remotely if not resolved locally (using Linked data representation)	0..1
width	integer	Width in pixel of the rendered symbol	0..1
height	integer	Height in pixel of the rendered symbol	0..1

Response Structure

The Response of the render action is according to the mime format accepted by the renderer, which is defined in the capabilities resource.

Examples

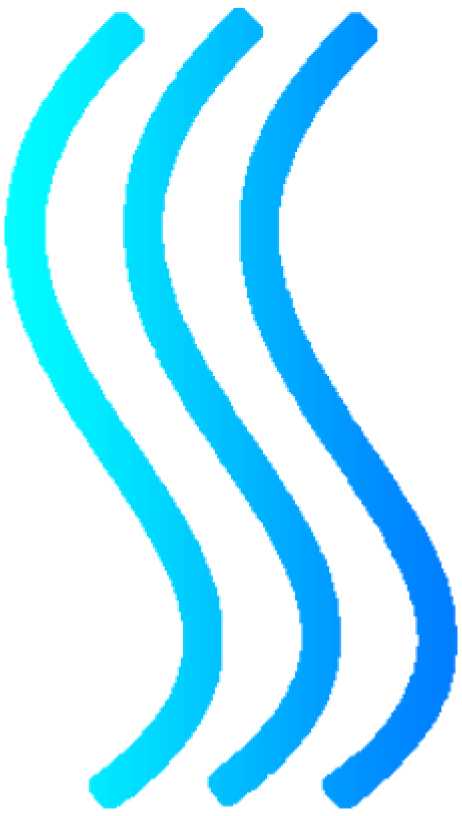
The following example requests the renderer to render the symbol Wind Incident to a PNG image of size 512x 512, using the renderer endpoint of the symbol

```
GET /symbols/06aab0337e96ea6e2535a2620b8e9a90/render?width=512&height=512 HTTP/1.1
Accept: image/png
Host: localhost
```

An alternative way to get the rendering of the symbol is by using the generic symbol renderer by providing its id or uri.

```
GET /renderer/symbols?id=06aab0337e96ea6e2535a2620b8e9a90&width=512&height=512
HTTP/1.1
Accept: image/png
```

The image produces is the following:



B.12.4.9 Symbolizer Renderer

The Symbolizer Renderer resource is used to render a symbolizer into a graphic representation such as SVG, PNG, TIFF, JPEG using its symbolizer definitions. This endpoint provides the ability to render the symbolizer into different formats and size that can be used for preview of symbolizer or generate legend items for styles.

Render a Symbolizer

To render a particular instance of Symbolizer, a HTTP `GET` request is performed on the symbolizer renderer endpoint. The request should accept one of the supported graphic formats advertised in the capabilities by the service. The endpoint provides the ability to customize the size of the graphic returned in the response.

Query Parameters

Parameter	Type	Description	Cardinality
id	String	Identifier of the symbolizer (optional if symbol id part the REST path)	0..1

Parameter	Type	Description	Cardinality
uri	String	URI of the symbolizer (only for generic renderer endpoint). URL can be retrieved remotely if not resolved locally (using Linked data representation)	0..1
width	integer	Width in pixel of the rendered symbolizer	0..1
height	integer	Height in pixel of the rendered symbolizer	0..1

Response Structure

The Response of the render action is according to the mime format accepted by the renderer, which is defined in the capabilities resource.

Examples

The following example requests the renderer to render the symbol Wind Incident to a PNG image of size 512x 512, using the renderer endpoint of the symbol

```
GET /symbolizer/dsds96dea6dd35a2620b8e9a90/render?width=512&height=512 HTTP/1.1
Accept: image/png
Host: localhost
```

An alternative way to get the rendering of the symbolizer is by using the generic symbol renderer by providing its id or uri.

```
GET /renderer/symbolizer?id=dsds96dea6dd35a2620b8e9a90&width=512&height=512 HTTP/1.1
Accept: image/png
```

Custom symbolizer rendering can also possibly be achieved by POSTing a Symbolizer specification to the generic symbolizer renderer endpoint. However this aspect was not tested during the testbed. Custom symbolizer were posted to the item endpoint to generate a unique id and this id was used by the renderer to generate its representation.

Render a Map

To render a Map composed of layers managed by the server, the GetMap operation parameters of WMS was adopted. The layer identifiers used by the API corresponds to the layerName property associated with the Layer object. The STYLE parameter was used to reference a Style, or a list of symbolizers or symbols. For more details about this operation, refer to the OGC Web Map Service specification.

B.12.4.10 SPARQL Service

The Semantic Portrayal Service provides a SPARQL service endpoint, which implements the SPARQL Protocol [https://www.w3.org/TR/sparql11-protocol], that can accept a SPARQL query on portrayal managed by the service. Both HTTP GET and HTTP POST are supported. This endpoint was not tested during the testbed.

Query Parameters

	HTTP Method	Query String Parameters	Request Content Type	Request Message Body
query via GET	GET	query (exactly 1)	None	None
query via URL-encoded POST	POST	None	application/x-www-form-urlencoded	• URL-encoded, ampersand-separated query parameters. • query (exactly 1)
query via POST directly	POST	None	application/sparql-query	Unencoded SPARQL query string

Example Request

The following is an example of a SPARQL query to fetch symbol instances URI and symbol names that belongs to the SymbolSet identified with the URI *http://www.opengis.net/testbed/12/ems/symbols#EMSSymbolSet*

```
PREFIX symbol:<http://www.opengis.net/ont/portrayal/symbol#>
SELECT ?symbol  ?symbolName WHERE {
  ?symbol a symbol:Symbol;
          symbol:symbolName ?symbolName;
          symbol:symbolSet
<http://www.opengis.net/testbed/12/ems/symbols#EMSSymbolSet>.
}
LIMIT 10
```

The request can be done using a HTTP Get request

```
GET
/sparql?query=PREFIX%20symbol%3A%3Chttp%3A%2F%2Fwww.opengis.net%2Font%2Fportrayal%2Fsy
mbol%23%3E%0ASELECT%20%3Fsymbol%20%20%3FsymbolName%20WHERE%20%7B%0A%20%20%3Fsymbol%20a
%20symbol%3ASymbol%3B%0A%20%20%20%20%20%20%20%20%20symbol%3AsymbolName%20%3FsymbolN
ame%3B%0A%20%20%20%20%20%20%20%20%20symbol%3AsymbolSet%20%3Chttp%3A%2F%2Fwww.opengi
s.net%2Ftestbed%2F12%2Fems%2Fsymbols%23EMSSymbolSet%3E.%0A%7D%20%0ALIMIT%2010 HTTP/1.1
Accept: application/sparql-results+json
Host: localhost
```

Response Structure

The SPARQL response conforms to the SPARQL specification. The SPARQL Protocol uses the response status codes defined in HTTP to indicate the success or failure of an operation.

The response body of a successful query operation with a 2XX response is either:

- A SPARQL Results Document in XML, JSON, or CSV/TSV format (for SPARQL Query forms SELECT and ASK); or,
- A RDF graph serialized, in the RDF/XML syntax or an equivalent RDF graph serialization (for SPARQL Query forms DESCRIBE and CONSTRUCT). The content type of the response to a successful query operation must be the media type defined for the format of the response body.

Example response

The following example shows a SPARQL query response

```
HTTP/1.1 200 OK
Content-Type: application/sparql-results+json
Content-Length: 2352
{
  "head": {
    "vars": [ "symbol" , "symbolName" ]
  } ,
  "results": {
    "bindings": [
      {
        "symbol": { "type": "uri" , "value":
"http://www.opengis.net/testbed/12/ems/symbols#ems.incident.airQuality-symbol" } ,
        "symbolName": { "type": "literal" , "value": "ems.incident.airQuality" }
      } ,
      {
        "symbol": { "type": "uri" , "value":
"http://www.opengis.net/testbed/12/ems/symbols#ems.incident.animalHealth-symbol" } ,
        "symbolName": { "type": "literal" , "value": "ems.incident.animalHealth" }
      } ,
      {
        "symbol": { "type": "uri" , "value":
"http://www.opengis.net/testbed/12/ems/symbols#ems.incident.animalHealth.animalDieOff-
symbol" } ,
        "symbolName": { "type": "literal" , "value":
"ems.incident.animalHealth.animalDieOff" }
      } ,
      {
        "symbol": { "type": "uri" , "value":
"http://www.opengis.net/testbed/12/ems/symbols#ems.incident.animalHealth.animalFeed-
symbol" } ,
        "symbolName": { "type": "literal" , "value":
"ems.incident.animalHealth.animalFeed" }
      } ,
      {
        "symbol": { "type": "uri" , "value":
"http://www.opengis.net/testbed/12/ems/symbols#ems.incident.aviation-symbol" } ,
        "symbolName": { "type": "literal" , "value": "ems.incident.aviation" }
      } ,
      {
        "symbol": { "type": "uri" , "value":
"http://www.opengis.net/testbed/12/ems/symbols#ems.incident.aviation.aircraftCrash-
symbol" } ,
        "symbolName": { "type": "literal" , "value":
"ems.incident.aviation.aircraftCrash" }
      } ,
      {
        "symbol": { "type": "uri" , "value":
"http://www.opengis.net/testbed/12/ems/symbols#ems.incident.aviation.aircraftHijacking
-symbol" } ,
        "symbolName": { "type": "literal" , "value":
"ems.incident.aviation.aircraftHijacking" }
      } ,
      {
        "symbol": { "type": "uri" , "value":
"http://www.opengis.net/testbed/12/ems/symbols#ems.incident.aviation.airportClosure-
symbol" } ,
        "symbolName": { "type": "literal" , "value":
"ems.incident.aviation.airportClosure" }
      } ,
      {
        "symbol": { "type": "uri" , "value":
"http://www.opengis.net/testbed/12/ems/symbols#ems.incident.aviation.airspaceClosure-
symbol" } ,
        "symbolName": { "type": "literal" , "value":
"ems.incident.aviation.airspaceClosure" }
      } ,
      {
        "symbol": { "type": "uri" , "value":
"http://www.opengis.net/testbed/12/ems/symbols#ems.incident.aviation.noticeToAirmen-
symbol" } ,
        "symbolName": { "type": "literal" , "value":
"ems.incident.aviation.noticeToAirmen" }
      }
    ]
```

```
  }
}
```

B.12.4.11 Layers

At the end of the testbed, endpoints were added to perform CRUD Operations for Layer. The initial model for Layer is described in Appendix A. The endpoints for the Layer management implemented for this testbed are described in the following table. This API will need more refinement in future testbeds, so detailed description of the operations is not provided.

Path	HTTP Methods	Description	Consume	Produce
/layers	POST	Create a new Layer	• application/json	• application/hal+json • application/json
/layers/{id}	GET,HEAD	Get the details of Layer identified by id		• application/hal+json • application/json
/layers/{id}	PUT	Updated layer with given id	• application/json	• application/hal+json • application/json
/layers/{id}	DELETE	Delete layer with given Id		
/layers/{id}/render	GET	Render a layer with given id		• Supported output graphic formats

Appendix C: Revision History

Table 46. Revision History

Date	Release	Editor	Primary clauses modified	Descriptions
May 31,2016	Stephane Fellah	.1	all	initial version

Appendix D: Bibliography

[1] OGC,: OGC Testbed 11 Demonstration. (2015).

[2] Testbed-11 Symbology Mediation Engineering Report, OGC document 15-058

[3] Implementing Linked Data and Semantically Enabling OGC Services Engineering Report, OGC document 15-054

[4] CCI Ontology Engineering Report,OGC document 14-049

[5] Guidelines for Successful OGC Interface Standards, OGC document 00-014r1

[6] SKOS Reference, W3C Recommendation, 18 August 2009. Latest version available at http://www.w3.org/TR/skos-reference .

[7] RDF 1.1 Turtle: Terse RDF Triple Language. W3C Recommendation, 25 February 2014. The latest edition is available at http://www.w3.org/TR/turtle/

[8] Gruber, Thomas R. "Toward principles for the design of ontologies used for knowledge sharing?" International journal of human-computer studies 43, no. 5 (1995): 907-928.

[9] Stuckenschmidt, Heiner, and Michel Klein. "Structure-based partitioning of large concept hierarchies." In The Semantic Web–ISWC 2004, pp. 289-303. Springer Berlin Heidelberg, 2004.

[10] Stuckenschmidt, Heiner, Christine Parent, and Stefano Spaccapietra, eds.Modular ontologies: concepts, theories and techniques for knowledge modularization. Vol. 5445. Springer, 2009.

[11] McCann, S., Brackin, R., Hobona, G.: OGC Testbed-13: DCAT/SRIM Engineering Report, OGC 17-040, Open Geospatial Consortium (2018)