Open Geospatial Consortium, Inc.

Date: 2008-09-12

Reference number of this document: OGC 07-124r2

Version: 0.1.0

Category: Discussion Paper

Editor(s): Chris Holmes

OGC® OWS-5 KML Engineering Report

Warning

This document is not an OGC Standard. This document is an OGC Discussion Paper and is therefore not an official position of the OGC membership. It is distributed for review and comment. It is subject to change without notice and may not be referred to as an OGC Standard. Further, an OGC Discussion Paper should not be referenced as required or mandatory technology in procurements.

Preface

Suggested additions, changes, and comments on this Discussion Paper are welcome and encouraged. Such suggestions may be submitted by email message or by making suggested changes in an edited copy of this document.

The changes made in this document version, relative to the previous version, are tracked by Microsoft Word, and can be viewed if desired. If you choose to submit suggested changes by editing this document, please first accept all the current changes, and then make your suggested changes with change tracking on.

Forward

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 Inc. 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.

Contents

Page

Figures

Page

Tables

Page

OGC® OWS-5 KML Engineering Report

1 Introduction

1.1 Scope

This OGC™ document is a report on the findings of the OWS-5 Agile Geography KML thread, giving recommendations for revisions and additions to the KML specification.

1.2 Document contributor contact points

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

Name	Organization
Holmes, Chris	The Open Planning Project
Turner, Andrew	Mapufacture
Ingrassia, Chris	Fortius One
Singh, Raj	OGC
Burggraf, David	Galdos

1.3 Revision history

Date	Release	Editor	Primary clauses modified	Description
2007-09-17	0.0.1	Holmes, Chris	All	Initial Version
2007-09-17	0.0.2	Turner, Andrew	2.1	Added Mobile phone use case and fixed some minor formatting
2007-10-20	0.0.3	Singh, Raj	9.1, 10	Core module/profiles refinement, GML change
2007-11-15	0.0.4	Holmes, Chris	All	Rewrite in to Engineering Document
2007-12-05	0.0.5	Holmes, Chris	10	Additions

2008-01-21	0.0.6	Holmes, Chris	All	Formatting, new sections
2008-03-09	0.0.7	Holmes, Chris	6.2, 6.2.1.1, 8	Cleanup, no more GML in KML, many context changes. Added Use Case 3 from C. Shorter
2008-04	1.0.0	Singh, Raj	7	Edits on OWS Context information
2008-7	0.9.0	C. Reed	Various	Make ready for posting as DP

1.4 Future work

It is anticipated that recommendations detailed in this document will be taken up for consideration by the KML and OWS Context standards working groups and considered for future versions of those standards.

1.5 Forward

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 Inc. 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.

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 06-121r3, *OpenGIS® Web Services Common Specification*

In addition to this document, this report includes several XML Schema Document files as specified in Annex A.

3 Terms and definitions

For the purposes of this report, the definitions specified in Clause 4 of the OWS Common Implementation Specification [OGC 06-121r3] shall apply. In addition, the following terms and definitions apply.

4 Conventions

4.1 Abbreviated terms

CRS	Coordinate Reference System
EPSG	European Petroleum Survey Group
GML	Geography Markup Language
GeoRSS	Geographically Encoded Objects for RSS
HTTP	Hypertext Transfer Protocol
MIME	Multipurpose Internet Mail Extensions
OGC	Open Geospatial Consortium
OWS	OGC Web Service, or Open Web Service
RSS	Really Simple Syndication
SLD	Styled Layer Descriptor
TBD	To Be Determined
TBR	To Be Reviewed
URI	Universal Resource Identifier
URL	Uniform Resource Locator
WCS	Web Coverage Service
WFS	Web Feature Service
WMS	Web Map Service
XML	Extensible Markup Language
1D	One Dimensional
2D	Two Dimensional
3D	Three Dimensional
4D	Four Dimensional

5 KML Overview

This Discussion Paper is about the use of KML, an encoding used to express geographic annotation and visualization on existing or future web-based online maps (2d) and earth browsers (3d). KML uses a tag-based structure with nested elements and attributes and is based on the XML standard.

Prior to 2007, KML was almost exclusively an internal configuration language for the Google Earth software. In 2007, Google submitted KML for standardization within OGC, which started two separate, parallel initiatives. One was the creation of an international standard version of KML, and this document reports on the other, an interoperability experiment to investigate ways in which KML could evolve to better service multiple, heterogeneous software systems, and better integrate with existing geo-centric information systems. The KML standardization process resulted in a KML standard called OGC® KML 2.2. That document was released about the same time as this one, so in some cases this document will conflict or diverge from the guidance found in the standard OGC® KML. It is therefore important to keep in mind that OWS-5 KML (KML as discussed in this document) is experimental, and does not represent the opinion of OGC as a whole.

This document details the results of those efforts. It investigates the harmonization of KML with relevant OpenGIS standards. It also proposes a consistent method for using KML on different technology platforms by sub-setting the language into modules designed for deployment on mobile devices, Web browsers, and “geo-browsers” (Google Earth, ArcExplorer, etc.). The sub-setting, or profiling, work not only enables easier and more logical implementation of KML on disparate platforms, but also enables easier compliance testing by making clear which tags are required and which are optional in a particular setting. The work done here should feed in to future versions of the KML encoding standard.

6 Summary

The main design goal decided upon was to divide KML into a core of functionality that all must implement, and then a set of modules of optional capabilities. These were divided into 'extended data' (what is called properties or attribute data in traditional GIS), 'styling', 'services', and '3d'. And then there is 'core', which is the minimal subset of KML would be expected for any implementation. Also included in this discussion is the design of a Feature Portrayal Service to render WFS (or any web based data) with an SLD to output KML.

7 KML "core" and modules

7.1 Introduction

The sheer number of KML files in the world points to the ease with which developers and everyday users can create such files, as the current specification is defined very flexibly and many of the tools that read KML are very forgiving. Creators of KML files have a very powerful language available to them, but can choose to use only a small subset. But any developer who has tried to implement a reader is quickly overwhelmed by the number of tags she must potentially read with her client, and without prior experience will have little idea which ones to implement.

KML has a large number of features, from simple geometry markup, to 3-D models, image pyramids, camera views, and so on. As KML is implemented by a variety of application developers in different markets and on different devices, expecting all KML clients to implement all parts is unreasonable. Two examples stand out. Mobile phones are unlikely to support the same level of 3D functionality as desktop applications that can take advantage of the 3D acceleration available in graphics cards today. And users of traditional GIS applications will expect to be able to easily visually integrate KML-provided data with spatial databases, Web Feature Services, etc. There are certain aspects of KML that make sense on some of the clients, but not on others.

To encourage more implementations that read and write KML, some sort of division and prioritization is a high priority. Two fundamental needs exist. One is to specify "profiles" of KML that target the feature sets different markets require. The other is to present KML in a way that specific user communities can effectively participate in the format's evolution without causing unnecessary wholesale changes that disrupt the entire KML community.

OWS-5 participants have defined four major industry visualization platforms for KML: **mobile devices (mobile); Web browsers (browser); 3D spinning globes (3D); and GIS desktop applications (GIS)**. In terms of spatial data services, we believe the existing OGC communities should use these target client platforms as a guide when deciding what profile of KML a particular data access service, such as WFS or SOS, should support.

The aforementioned four KML profiles are defined by grouping the current KML tags into functional sets called modules. These modules are then grouped into profiles in a non-exclusive manner. For example, 3D and GIS profiles both use the 3D module, and all profiles use the Metadata module. A main benefit to having the concept of modules in addition to the concept of profiles relates to OGC process. The OGC Technical Committee Working Groups may be the most active in defining and developing modules, while groups more interested in market concerns, like Domain Working Groups and the Planning Committee, may have a more active role in defining profiles. The concept of a core and modules also makes compliance testing much easier, and lowers the bar for new implementations to be certified for certain levels of functionality, instead of being

required to implement KML grammar that can not be visualized effectively on a particular platform.

7.2 KML Modules

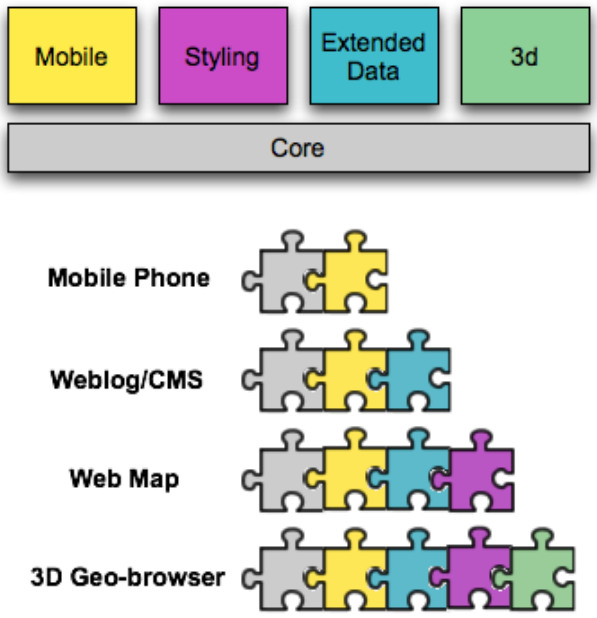

Figure 1 – KML Modules and Deployment Platforms

As one can see above, there are five modules defined

a) **Core:** basic elements that all software must support to call itself a KML reader.

b) **Mobile:** elements mobile devices like cell phones and PDAs should support.

c) **Styling:** The elements to specify how the placemark should look.

d) **Extended data:** Attribute data in the GIS sense, to specify untyped data and typed data (with its defining schema)

e) **3d:** The elements that are specific to 3d applications, like LookAt, region definitions and 3d model support

These can be combined in various ways for different applications. To illustrate, a simple phone may just want locations and some simple metadata (or attributes) of that location to place on top of a street map – for example, nearby restaurants and their opening hours. As the client (or library) becomes richer, additional modules can be implemented. A Weblog extension may support styling (icons, line colors) for publishing KML from posts.

So, applications will be able to simply define what modules they support so that users can clearly understand the type of functionality they can expect. In addition, these libraries

and applications can achieve OGC verification by implementing the defined feature set in the modules they use.

One topic of discussion was defining a 'services' module, which would add grammar to KML to describe bindings to OGC web services like WMS, WFS, WCS as well as things like GML and GeoRSS. After much debate and discussion, the group decided that this would be duplicative work, as the OWS Context document serves the same purpose. Therefore, to add this feature to KML might increase the complexity of KML for people who don't work with OGC web services, and not add much value to those who do. Our work in this area instead concentrated on collaborating with the OWS Context working group to help define the integration of KML into OWS Context.

7.3 Core

The KML Core module includes the minimal subset of KML that would be expected for any implementation. Currently the elements of KML that should exist in Core include:

`<kml>` – the root element

`<Document>` – the container for features, styles, and schemas

`<atom>` – tags used for attribution

`<Folder>` - used to arrange other Features hierarchically (Folders, Placemarks, NetworkLinks)

`<Placemark>` - a Feature with associated Geometry (note a feature can be an overlay or a container)

`<Link>` and `<NetworkLink>` – basic linking to other files should be supported in core

<ScreenOverlay> - draws an image overlay fixed to the screen, such as for a compass, logo or legend

`<TimeStamp>` and `<TimeSpan>` – A moment or span in time

`<Geometry>` – Same as Simple Features for GML, see below

This list was derived from the current KML 2.2 tags and signifies the general concepts that the group believes should be in KML Core - allow a user to express location, time, and possibly link to another KML document or service. In addition, providing the current Atom support allows for attribution and other interesting things for linking to HTML and GeoRSS.

7.3.1 Geometry Markup

Geometries for KML will remain the same as it has been in KML 2.2, which is a slightly modified version of GML 2.1.2, in the KML namespace. Future efforts may seek to align with Simple Features for GML. However, at this time the group decided that there were

few clear advantages of upgrading to the newer GML, but a clear disadvantage would be completely breaking backwards compatibility, requiring a rewrite of every KML client currently in existence.

The group experimented with implementations using GML 3 Simple Features, and ran into problems with how to define the 'extrudes' element, which incorporates the height of the feature as an integral part of the geometry. To work properly this would have required a fundamental change to Simple Features for GML, which would not be easy to do. 3d clients require the height attribute to be included, so that renderers can efficiently calculate whether the geometry should be shown. So further work remains on aligning KML Geometries and GML, for this testbed it was not deemed necessary.

7.4 Extended Data

With the current KML 2.2 specification there is an unneeded split between typed and untyped data, making them incompatible. A provider of KML must determine ahead of time whether to produce KML with SchemaData or Data tags. Obviously Google Earth understands both, but client implementers, such as OpenLayers or Mapufacture, often will be forced to make a decision which to support.

Fortunately, there is a better way. Both untyped and typed data can make use of the Data element, with SchemaData offering additional typing information. But if a KML client doesn't understand a SchemaData element and all that it offers it can still make use of the Data, just in an untyped way. This should require only a few small changes from the current method:

The following examples are all taken from:
http://code.google.com/apis/kml/documentation/extendeddata.html

```
 <ExtendedData>
  <Data name="string">
    <displayName>...</displayName>            <!-- string -->
    <value>...</value>                        <!-- string -->
  </Data>
</ExtendedData>

<ExtendedData>
  <SchemaData schemaUrl="anyURI">
    <SimpleData name=""> ... </SimpleData>    <!-- string -->
  </SchemaData>
</ExtendedData>
```

Using the untyped 'Data' element or the typed 'SchemaData' element (referring to a schema location at 'schemaUrl' are the two ways of currently specifying extended data.. What we can do instead is make the schemaUrl an optional attribute of ExtendedData:

```
<ExtendedData schemaUrl="anyURI"> (schemaUrl is optional)
  <Data name=""> ... </Data>
</ExtendedData>
```

So some examples. The basic case of 'Data' just drops the extraneous 'value' tags:

```
    <ExtendedData>
      <Data name="holeNumber">
        <value>1</value>
      </Data>
      <Data name="holeYardage">
        <value>234</value>
      </Data>
      <Data name="holePar">
        <value>4</value>
      </Data>
    </ExtendedData>
```

becomes:

```
   <ExtendedData>
      <Data name="holeNumber">1</Data>
      <Data name="holeYardage">234</Data>
      <Data name="holePar">4</Data>
   </ExtendedData>
```

The basic schema data case retains the same schema, but uses 'data' instead of 'simpledata':

```
  <Schema name="TrailHeadType" id="TrailHeadTypeId">
    <SimpleField type="string" name="TrailHeadName">
      <displayName><![CDATA[<b>Trail Head Name</b>]]>
</displayName>
    </SimpleField>
    <SimpleField type="double" name="TrailLength">
      <displayName><![CDATA[<i>Length in miles</i>]]>
</displayName>
    </SimpleField>
    <SimpleField type="int" name="ElevationGain">
      <displayName><![CDATA[<i>Change in altitude</i>]]></displayName>
    </SimpleField>
  </Schema>
    <ExtendedData>
      <SchemaData schemaUrl="#TrailHeadTypeId">
        <SimpleData name="TrailHeadName">Mount Everest
  </SimpleData>
        <SimpleData name="TrailLength">347.45</SimpleData>
        <SimpleData name="ElevationGain">10000</SimpleData>
      </SchemaData>
    </ExtendedData>
```

becomes:

```
  <Schema name="TrailHeadType" id="TrailHeadTypeId">
    <SimpleField type="string" name="TrailHeadName">
      <displayName><![CDATA[<b>Trail Head Name
 </b>]]></displayName>
```

```
    </SimpleField>
    <SimpleField type="double" name="TrailLength">
      <displayName><![CDATA[<i>Length in
  miles</i>]]></displayName>
    </SimpleField>
    <SimpleField type="int" name="ElevationGain">
      <displayName><![CDATA[<i>Change in altitude</i>]]>
</displayName>
    </SimpleField>
  </Schema>
      <ExtendedData schemaUrl="#TrailHeadTypeId">
        <Data name="TrailHeadName">Mount Everest</Data>
        <Data name="TrailLength">347.45</Data>
        <Data name="ElevationGain">10000</Data>
    </ExtendedData>
```

Arbitrary XML data would be handled the same way it is in KML 2.2:

```
    <ExtendedData xmlns:camp="http://campsites.com">
      <camp:number>14</camp:number>
      <camp:parkingSpaces>2</camp:parkingSpaces>
      <camp:tentSites>4</camp:tentSites>
    </ExtendedData>
```

The one thing that appears to be lost is display names for untyped data. But even in the example given of the 'advanced' used of display name, this can be redone with less repetition:

```
<?xml version="1.0" encoding="UTF-8"?>
<kml xmlns="http://earth.google.com/kml/2.2">
  <Document>
    <name>Entity-Replacement</name>
    <Style id="displayName-value">
      <BalloonStyle>
        <text>
          <![CDATA[
          This is $[name]<br/>
          $[holeNumber/displayName] $[holeNumber]<br/>
          $[holePar/displayName] $[holePar]<br/>
          $[holeYardage/displayName] $[holeYardage]
            ]]>
        </text>
      </BalloonStyle>
    </Style>
    <!-- Shared style sample
      Two Placemarks use the same balloon template
    -->
    <Placemark>
      <name>Club house</name>
      <styleUrl>#displayName-value</styleUrl>
      <ExtendedData>
        <Data name="holeNumber">
          <displayName><![CDATA[
            <b>This is hole </b>
```

```
          ]]></displayName>
          <value>1</value>
        </Data>
        <Data name="holePar">
          <displayName><![CDATA[
            <i>The par for this hole is </i>
          ]]></displayName>
          <value>4</value>
        </Data>
        <Data name="holeYardage">
          <displayName><![CDATA[
            <b><i>The yardage is </i></b>
          ]]></displayName>
          <value>234</value>
        </Data>
      </ExtendedData>
      <Point>
        <coordinates>-111.956,33.5043</coordinates>
      </Point>
    </Placemark>
    <Placemark>
      <name>By the lake</name>
      <styleUrl>#Entity-Replacement</styleUrl>
      <ExtendedData>
        <Data name="holeNumber">
          <displayName><![CDATA[
            <b>This is hole </b>
          ]]></displayName>
          <value>5</value>
        </Data>
        <Data name="holePar">
          <displayName><![CDATA[
            <i>The par for this hole is </i>
          ]]></displayName>
          <value>5</value>
        </Data>
        <Data name="holeYardage">
          <displayName><![CDATA[
            <b><i>The yardage is </i></b>
          ]]></displayName>
          <value>523</value>
        </Data>
      </ExtendedData>
      <Point>
        <coordinates>-111.95,33.5024</coordinates>
      </Point>
    </Placemark>
  </Document>
</kml>
```

Can get across the exact same information, in an untyped way, with:

```
<?xml version="1.0" encoding="UTF-8"?>
<kml xmlns="http://earth.google.com/kml/2.2">
  <Document>
    <name>Entity-Replacement</name>
```

```
    <Style id="displayName-value">
      <BalloonStyle>
        <text>
          <![CDATA[
          This is $[name]<br/>
          <b>This is hole </b> $[holeNumber]<br/>
          <i>The par for this hole is </i> $[holePar]<br/>
          <b><i>The yardage is </i></b> $[holeYardage]
            ]]>
        </text>
      </BalloonStyle>
    </Style>
    <!-- Shared style sample
      Two Placemarks use the same balloon template
    -->
    <Placemark>
      <name>Club house</name>
      <styleUrl>#displayName-value</styleUrl>
      <ExtendedData>
        <Data name="holeNumber">1</Data>
        <Data name="holePar">4</Data>
        <Data name="holeYardage">234</Data>
      </ExtendedData>
      <Point>
        <coordinates>-111.956,33.5043</coordinates>
      </Point>
    </Placemark>
    <Placemark>
      <name>By the lake</name>
      <styleUrl>#Entity-Replacement</styleUrl>
      <ExtendedData>
        <Data name="holeNumber">5</Data>
        <Data name="holePar">5</Data>
        <Data name="holeYardage">523</Data>
      </ExtendedData>
      <Point>
        <coordinates>-111.95,33.5024</coordinates>
      </Point>
    </Placemark>
  </Document>
</kml>
```

This is cleaner, putting the display info in the template. Theoretically there is one edge case, if users actually wanted to put different display names in different placemarks. But treating the display name as another value also easily solves this:

```
<ExtendedData>
    <Data name="holeNumber">5</Data>
     <Data name="holeNumberDisplay">
       <![CDATA[<b>This is hole</b>]]></Data>
    <Data name="holePar">5</Data>
    <Data name="holeYardage">523</Data>
</ExtendedData>
```

This approach of combining type and untyped data allows nice flexibility between the two approaches. If at some later time that user found a program that consumes KML extended data but needs schemaData they could just create a Schema and reference it from their data tags, instead of having to rewrite the full file. In the following case the schema would have been added to the same document as GolfCourseTypeId.

```
<ExtendedData schemaUrl=”#GolfCourseTypeId”>
  <Data name="holeNumber">5</Data>
  <Data name="holePar">5</Data>
  <Data name="holeYardage">523</Data>
</ExtendedData>
```

And if someone received a bunch of typed data but then wanted to add their own untyped data to it, they'd just have to remove the schemaUrl, and then could add all they wanted.

Combining SchemaData and Data leads to a large upside by not forcing producers to choose between typed and untyped data, and by allowing clients to not have to write additional code to understand data coming from an typed source, even if they just want to display the un-typed data.

7.5 Styling

Styling is a very important aspect of KML, as it is very much a visual language. While many lightweight clients may not implement styling, it is expected that most full clients will want to portray the KML as providers specify.

There are two groups of style elements that KML clients who wish to support it need to implement. The first is ColorStyle and its related elements – LineStyle, PolyStyle, IconStyle and LabelStyle plus BalloonStyle and ListStyle. These specify a color and any additional information on how placemarks should be rendered. The second is StyleSelector and its Style and StyleMap children, plus styleUrl. These are how the various color elements are laid out. Style is a container for set color styles, and StyleMap contains a map of Styles to be used for highlighting and the like. styleUrl lets an individual feature refer to a style.

The OWS-5 testbed desired to make changes to the style elements of KML, since the current specification does a poor job of separating presentation from data, with style being an explicit attribute of features. It is possible to define a style and refer to it by URL, but each placemark must refer back to the URL, instead of just inheriting a style sheet.

OGC's SLD specification nicely separates presentation, but also feels quite heavyweight and verbose compared to KML, and likely would have poor uptake.

Another option presented was to use CSS3 directly. This would involve allowing named classes, and likely could cover a wide range of cases while leveraging a very well known standard instead of writing another one. More investigation is needed to figure out if this is in fact feasible, and would cover all use cases.

There was, however, fairly significant pushback from implementers, and it was beyond the resources of test bed participants to make a full proposal and implementation. But at some point we recommend a harmonization between SLD and KML, hopefully simplifying the former for simple use cases, while allowing the benefits of separation of data and presentation for the latter. A lightweight shared profile could be ideal, with each allowing a few more capabilities for their particular use case.

In the meantime, the recommended way to keep data and presentation separate is with OGC services and SLD. A component WMS can render a data source (WFS and WCS) with a supplied SLD in to a KML document. This is discussed at length in section XXX. And a WMS that fully supports SLD can also render the data that it is already serving according to the rules of a user supplied SLD. Since this use case can be met with existing OGC standards it is not as high a priority to harmonize styling at this time.

7.6 3d

The 3d module has been discussed the least. The main reasons are that the 3d functionality is mature, being that the primary KML deployment platform to date has been Google Earth, and that no one participating in this testbed implemented a 3d client.

7.7 Vendor Tags

Though also outside the scope of this testbed, the group feels there should be a flexible mechanism for vendors to add additional tags to KML that are not yet ready to be standardized. This will enable a variety of implementers to experiment with the KML format in a flexible way without needing to modify the specification. If several vendors are all adding tags for similar goals then they should work in a testbed to standardize those and add an additional module to the specification.

NOTE: Since this writing, the international standard OGC KML 2.2 was released with formal guidance for extending KML by vendors, but the timing was such that it was not available for OWS-5.

Some of the current tags in KML 2.2 should likely be classified as vendor tags, since they are outside the classifiable scope of any of the modules enumerated above. Image Pyramid and Photo Overlays are one such example, that all 3d globes may not even desire to implement, so should perhaps not be part of the 3d profile. Other OGC specs suffer from limited extension points – modifications that one wants to make often break XML Schema validation – so there should be a very flexible mechanism for implementers to easily extend KML.

8 KML MIME Type and XML Namespace

The MIME type and XML namespace for KML are both vendor-specific to Google. As part of becoming an open standard these should move to vendor-neutral OGC locations.

8.1 KML MIME Type

The MIME types used for KML 2.2 and KML 2.2 compressed are:

```
application/vnd.google-earth.kml+xml
application/vnd.google-earth.kmz
```

The next version of KML should have new MIME types. This group recommends:

```
application/kml+xml
application/kmz
```

This is also a useful way for clients to request different versions of KML from WMS servers, in the future one should be able to use a subtype clause in the mime type to disambiguate versioning, for example:

```
application/kml+xml;subtype=kml2.3
```

8.2 KML XML Namespace

Additionally, the namespace for version 2.2 KML documents is:

```
xmlns="http://earth.google.com/kml/2.2”
```

This should be updated to a more neutral namespace to align with the other OGC namespaces. The group recommends:

```
xmlns="http://www.opengis.net/kml/2.2”
```

NOTE: Since this writing, the international standard OGC KML 2.2 was released with namespace guidance identical to that presented here. The KML MIME types, however, did not change.

9 KML and OGC Web Services

By 'Services' this document refers to resources that are accessible on the web that provide geospatial information. Foremost among these are the major OGC standards – WMS, WFS and WCS, along with the Sensor services. But services should be flexible enough to refer to other sources of geospatial information, GeoRSS feeds, base layers provided by Google Maps and Virtual Earth, results of OpenSearch queries and whatever may gain popularity next.

Some years ago, some OGC members decided to create a file format that captured the *state* of a user's interaction with a map. This started with the recognition that the process of zooming in on a map, adding various Web Mapping Services to it, and choosing layers and styles was a resource-intensive, value-add activity that was worth persisting across sessions, so that one could restart work from the view you finished with, or even share that view with other parties. This concept was very successful, and was soon extended to include not just Web Mapping Services, but also Web Feature Services and GML data (in OWS-4 Decision Support Thread). This file format is called the OWS Context Document, and what it basically does is allow one to store all the information required to re-create a service request, including but not limited to location, styling and coordinate referencing information.

Example: http://www.ogcnetwork.net/schemas/owc/0.2.1/owsContext.xml

KML has served a very similar role for the Google Earth (GE) and Maps (GM) platforms. One of the primary elements of KML is `<LookAt>`, which helps the client application position the map view for the user. Unlike pure OGC service clients, GE's map content always defaults to the imagery streamed in by Google, so no state information other than `<LookAt>` is required for the user to get a map. However, a GE user may also want to capture the other KML layers displayed on their map and share that view with others. They can do this by including `<NetworkLink>`s to other KML data files along with that `<LookAt>` element that sets the view's location.

Finally, similar to the recent OWS Context experiments that include GML data inline, KML has always been able to include geographic content and style it. So due to all these similarities between the *goals, if not the syntax,* of these two XML file formats, it made sense to evaluate merging the two to create a single OGC standard for describing a map view that can consist of a mixture of inline geographic content, as well as content brought in from external services.

One could foresee significant demand from the most sophisticated geospatial users, such as the US DoD, the European Space Agency, NATO, and others for a comprehensive map document that included not only traditional OGC services, but KML as well. All features would not, of course, work in all clients. For example, one might envision a map document that included Google Earth imagery, Placemarks, WMS services, WFS services and GML data. Some, or all of these might display in GE, but in another client

maybe only the WMS services and GML data might display. The key paradigm is that, like RSS, a client should not break when it cannot understand something, but do the best possible job of using the content and services it understands. This is an extremely powerful paradigm for sharing views of geospatial information across communities.

9.1 KML integration with OWS Context

The Agile Geography group concluded that the best way to accomplish the integration of OWS Context and KML is *not* to add open web service bindings to KML, but to allow the inclusion of KML in OWS Context documents. This is a nice improvement for Context, since its use is primarily visualization, and KML is much more visually oriented than GML. Imposing SLD as the only way to style in-line GML is a high barrier, while KML has a lighter weight way of including a style alongside a feature. In practice, this simply means allowing a `<kml:Document>` tag in the `<Layer>`.

```
<OWSContext version="0.3.0" id="ows-context-OWS5-ER-9.1"
      xmlns="http://www.opengis.net/ows-context"
      xmlns:gml="http://www.opengis.net/gml"
      xmlns:kml="http://www.opengis.net/kml/2.2"
      xmlns:ogc="http://www.opengis.net/ogc"
      xmlns:ows="http://www.opengis.net/ows"
      xmlns:sld="http://www.opengis.net/sld"
      xmlns:xlink="http://www.w3.org/1999/xlink"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://www.opengis.net/ows-context
      ./owsContext.xsd">
      <General>
       <Window width="500" height="300"/>
       <ows:BoundingBox crs="urn:ogc:def:crs:EPSG:6.6:4326">
          <ows:LowerCorner>-71.148 42.259</ows:LowerCorner>
          <ows:UpperCorner>-71.001 42.439</ows:UpperCorner>
       </ows:BoundingBox>
       <ows:Title>OWS Context Document</ows:Title>
       <ows:Abstract>The OpenGIS Web Services Context
      Document</ows:Abstract>
       <ows:ServiceProvider>
          <ows:ProviderName>XYZ</ows:ProviderName>
          <ows:ServiceContact/>
       </ows:ServiceProvider>
      </General>
      <ResourceList>
       <Layer>
          <kml:Document>
             <kml:name>opengeo:archsites 1 to 100</kml:name>
             <kml:Style id="archsitesStyle">
                <kml:IconStyle>
                   <kml:color>ffffffff</kml:color>
                   <kml:colorMode>normal</kml:colorMode>
                   <kml:Icon>
                      <kml:href>
http://maps.google.com/mapfiles/kml/pal4/icon25.png
                      </kml:href>
                   </kml:Icon>
                </kml:IconStyle>
             </kml:Style>
```

```
            <kml:Placemark id="archsites.1">
                <kml:name>Signature Rock</kml:name>
                <kml:description>Signature Rock
      Description</kml:description>
                <kml:styleUrl>#archsitesStyle</kml:styleUrl>
                <kml:Point>
                   <kml:coordinates>-
      103.82681673,44.38162255</kml:coordinates>
                </kml:Point>
            </kml:Placemark>
        </kml:Document>
       </Layer>
       <Layer queryable="0" hidden="0" group="Layers">
          <ows:Title>hydro</ows:Title>
          <ows:Abstract>hydro</ows:Abstract>
          <ows:Identifier>hydro</ows:Identifier>
          <ows:OutputFormat>image/gif</ows:OutputFormat>
          <ows:OutputFormat>image/png</ows:OutputFormat>
          <ows:OutputFormat>image/jpeg</ows:OutputFormat>
          <ows:AvailableCRS>EPSG:4326</ows:AvailableCRS>
          <Server service="urn:ogc:def:serviceType:WMS" version="1.1.1"
      title="Boston on Oracle">
             <OnlineResource
      xlink:href="http://webservices.ionicsoft.com/
                   ionicweb/wfs/BOSTON_ORA"/>
          </Server>
          <sld:MinScaleDenominator>5000</sld:MinScaleDenominator>
          <sld:MaxScaleDenominator>50000</sld:MaxScaleDenominator>
          <StyleList>
             <Style current="1">
                <Name>default</Name>
                <Title>default</Title>
             </Style>
          </StyleList>
       </Layer>
      </ResourceList>
</OWSContext>
```

One should also be able to reference a remote KML file sitting on server, just like one can with GML. This would be a layer, looking something like:

```
<Layer hidden="0">
   <Server service="urn:ogc:serviceType:KML" version="2.2"
      title="KML Tasmania 2.2">
      <OnlineResource xlink:href="kml/topp_tas_ows5.kml"/>
   </Server>
</Layer>
```

9.2 OWS Context Simplification Recommendations

The group however hopes to make recommendations for the simplification of Context documents to be much more terse, containing a minimum of relevant information so as to not confuse potential implementers. It can also be improved to allow more flexibility in the definition of services, so that a new type of geospatial service does not need a schema change, but can be easily added, and simply ignored by clients who don't understand the

new service. Many members of the group were excited to go the route of adding OGC and other services to KML, mostly because it could enable a much more compact definition of those services that is easy to understand by potential implementers. The original OWS Context document did not seem to be designed with flexibility and ease of construction in mind. It just included the entire section of a WMS capabilities document, even if many of the elements were redundant or not relevant.

Further investigation into the schemas of OWS Context led the group to realize that Context documents actually *require* few elements, but most all the examples are quite verbose, demonstrating all the capabilities, leading implementers to believe it is more complex than it actually is. Much of the perceived complexity can be overcome with a number of simpler examples and implementations. Past that there are a few recommendations to eliminate several more elements that may not be necessary.

The goal should be to collapse the verbose definition of a layer into something nice and terse. From:

```
<Layer queryable="0" hidden="0" group="Layers">
      <ows:Title>hydro</ows:Title>
      <ows:Abstract>hydro</ows:Abstract>
      <ows:Identifier>hydro</ows:Identifier>
      <ows:OutputFormat>image/gif</ows:OutputFormat>
      <ows:OutputFormat>image/png</ows:OutputFormat>
      <ows:OutputFormat>image/jpeg</ows:OutputFormat>
      <ows:AvailableCRS>EPSG:4326</ows:AvailableCRS>
      <Server service="OGC:WMS" version="1.1.1"
          title="Boston on Oracle">
       <OnlineResource xlink:type="simple"
          xlink:href="http://webservices.ionicsoft.com/
                 ionicweb/wfs/BOSTON_ORA" />
      </Server>
      <sld:MinScaleDenominator>5000</sld:MinScaleDenominator>
      <sld:MaxScaleDenominator>50000</sld:MaxScaleDenominator>
      <StyleList>
       <Style current="1">
          <Name>default</Name>
          <Title>default</Title>
       </Style>
      </StyleList>
</Layer>
```

To a minimum of:

```
<Layer>
      <ows:Title>hydro</ows:Title>
      <Server service="urn:ogc:serviceType:WMS" version="1.1.1">
          <OnlineResource xlink:href="http://webservices.ionicsoft.com/
                 ionicweb/wfs/BOSTON_ORA"/>
      </Server>
</Layer>
```

An intelligent client should be able to construct an appropriate request from this information, potentially by re-querying the Capabilities document. But a naïve client

could also just use a number of reasonable defaults that most WMS services support. Most of the simplifications are possible within the current schemas, and then there are a few small changes that would make this completely possible. The assumptions and simplifications beg a bit of further explanation:

`Queryable` is currently a required attribute, but it does not need to be, and it adds a bit of confusion. It should have a default value of 'false', and if the layer is queryable then it should be included with a value of true. Note however that the recommendation below to collapse Coverage and FeatureType in to layers would need to change this element, as queryable makes little sense for WFS and WCS services, as they are by definition queryable. In that case the queryable should likely drop down to be an optional element of a WMS Service type only. The Context group could also consider just dropping the queryable element in favor of the layer equivalence, as the query operations of WFS and WCS fulfill the need to query in much more interoperable manner.

`hidden` is another attribute that is currently marked as required and could just have a reasonable default of 'false', only needing to be specified if it is true.

`group` is currently optional, so it's fine to leave it off. No schema changes would be required, though it is recommended that the Context xsd file include 'use=optional', which is the default, but other attributes spell it out explicitly.

`ows:Title, ows:Abstract, ows:Identifier` are very confusing, and none seem to be required. But it's unclear which, if any, is supposed to identify the name of the layer that the client should be requesting with a GetMap request. Perhaps a non-OWS namespace element should be used – something like layerName – so that it does not need to bring in another namespace which is potentially confusing. Implementers should have a clear recommendation on which element to use to identify the name of the layer. We chose title above, but are not sure if it is the correct one to use.

`ows:OutputFormat` is currently optional. When a service only supports an unusual format this could be useful, but in general, clients should be able to make a reasonable guess or query the capabilities document. The 'current' attribute also feels a bit odd. It seems like one should only list the current output format – the rest can be queried if need be by the client. The idea of context as the 'state' of the map seems to indicate it should just set what the client should load, not the world of options that the client can allow, reproducing the capabilities document.

`title` in the server element is not required in ows context 0.2.1, so no changes are needed there.

`xlink:type="simple"` is optional in the xlink schema, and feels very extraneous, as all the types are always simple. Use of that attribute should be eliminated in examples.

The rest of the changes, namely the elimination of Style and min/max scale denominators, are well within the current limits of the Context document. The style elements in particular are very extraneous; it requires 6 lines to tell the client that it should just use the default. Instead of writing all those lines OWS Context should just

leave it off, and only specify the style if it is different from the default. There is also no reason to list all the potential styles and to specify which is 'current'. Only the current should be listed, the others can be queried. This is not to say an intelligent client should not cache the available styles – just that it does not need to persist those to a context document. If the client just relied on the context document, then it would have no idea if there are new styles, etc.

9.3 Service Equivalence in OWS Context

Further recommendations include merging 'Layer', 'Coverage' and 'FeatureType' tags in to one. 'Layer' makes more sense, as it easily covers the others. If one wants to specify a WFS or some other tag that currently falls under FeatureType then the Server element can just be used under Layer. Past just simplifying things a bit, this also allows us to introduce the idea of equivalence of layers, as one can put multiple server elements with different service types under the same layer:

```
<Layer queryable="0" hidden="0">
     <Server service="OGC:WMS" version="1.0.0" title="OGC:WMS"
default="1">
       <OnlineResource xlink:type="simple"
xlink:href="http://localhost:8080/wms"/>
     </Server>
     <Server service="OGC:WFS" version="1.0.0" title="OGC:WFS">
       <OnlineResource xlink:type="simple"
xlink:href="http://localhost:8080/wms"/>
     </Server>
     <Name>basic</Name>
     <Title>World Basemap</Title>
     <SRS>EPSG:4326</SRS>
   </Layer>
```

This allows clients to know if a WMS layer is also available for query and editing as WFS, and indeed with other formats the client may have certain advantages of dealing with it in a certain format. The owc:ServerType element is already repeatable in OWS Context 0.2.1, so it should be possible to do this already. All that is needed is to deprecate owc:FeatureTypeType and owc:CoverageType elements, and add documentation suggesting best practices to clients on how to handle multiple service types in a single `<Layer>`.

10 KML Feature Portrayal Service

The group also did work on KML output from a Feature Portrayal Service. The first step in doing this is doing KML as an output format from a WMS. This is quite possible, and can be connected to KML clients using the Network Link element of KML. The format to specify a bounding box in a Network Link is completely compatible with a WMS bounding box. One can provide a full WMS request as the URL for a network link, leaving off the BBOX param, and using KML as the output format, and it will function properly.

One downside of this approach is that there are a number of parameters that are either constant or can be ignored by consumers of KML, but that must be provided for the WMS. One member of the group experimented with a KML 'reflector' which would automatically fill in the parameters of a WMS request so that a client could make a request like:

http://geo.openplans.org:8080/geoserver/wms/kml_reflect?layers=topp:states

The SRS of URN:OGC:DEF:CRS:EPSG:6.6:4326 is implied for KML, the width and height aren't used as its an xml output with the actual geographic values, the bounding box is supplied by the client in a network link, the style is the default, and the GetMap request is implied. The additional parameters can be supplied, and will override the defaults.

The other issue that should be noted is that a KML Network Link can receive placemarks (features plus style information) or 'ground overlays', which are basically rasters. The group found that often it is easy to overload clients with information, since there is no clear streaming protocol, though the 'super overlay' functionality looks promising. In these cases it is possible for a server to simply return a KML that just contains a ground overlay of what would be rendered by the same WMS request with a 2d output like png or jpeg. Many WMS servers could thus easily output KML by just wrapping their standard WMS responses in a KML envelope. This approach, however, often does not look as nice on advanced clients, as it does not offer nice looking pop-ups and client controlled label placement.

The same lessons from WMS output of KML also apply to Feature Portrayal Services, as they are just WMS services that reference remote services. The group has developed recommendations for improving the Component WMS specification, of which Feature Portrayal is a part. Those recommendations appear in a companion to this Engineering Report, 08-064, "Component WMS Interface".

11 KML Styling

This focus of this section is to report on how to generically transform styling rules from Symbology Encoding (SE) or Styled Layer Descriptor (SLD). The current SE and SLD specs were found lacking in one area, which is the styling of additional textual information. Label in SLD clearly maps to the KML 'Name' element, but there are elements such as description, snippet and more that have no easy corrolary. At least one implementation had a set of special templates to handle these, but these would not work for a Feature Portrayal Service, since only the SLD can be specified remotely.

The solution the group came up with was to extend SLD for these additional elements. Hence, we suggest that the elements named 'Snippet', 'FeatureDescription', and 'OtherText' are added to the sld:TextSymbolizer content model in the OGC Styled Layer Descriptor schema as shown below:

```
<xsd:element name="TextSymbolizer"
    substitutionGroup="sld:Symbolizer">
  <xsd:annotation>
    <xsd:documentation>
      A "TextSymbolizer" is used to render text labels according to
      various graphical parameters.
    </xsd:documentation>
  </xsd:annotation>
  <xsd:complexType>
    <xsd:complexContent>
      <xsd:extension base="sld:SymbolizerType">
        <xsd:sequence>
          <xsd:element ref="sld:Geometry" minOccurs="0"/>
          <xsd:element ref="sld:Label" minOccurs="0"/>
          <xsd:element ref="sld:Font" minOccurs="0"/>
          <xsd:element ref="sld:LabelPlacement" minOccurs="0"/>
          <xsd:element ref="sld:Halo" minOccurs="0"/>
          <xsd:element ref="sld:Fill" minOccurs="0"/>
          <xsd:element ref="sld:Snippet" minOccurs="0"/>
          <xsd:element ref="sld:FeatureDescription" minOccurs="0"/>
          <xsd:element ref="sld:OtherText" minOccurs="0"
    maxOccurs="unbounded"/>
        </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>
</xsd:element>

<xsd:element name="Snippet" type="sld:ParameterValueType"/>

<xsd:element name="FeatureDescription"
    type="sld:ParameterValueType"/>

<element name="OtherText">
  <complexType mixed="true">
    <complexContent mixed="true">
```

```
        <extension base="sld:ParameterValueType">
          <sequence>
            <element name="displayName" type="string" minOccurs="0"/>
            <element name="sourceURL" type="anyURI" minOccurs="0"/>
          </sequence>
          <attribute name="target" type="string" use="optional"/>
        </extension>
      </complexContent>
    </complexType>
  </element>
```

The new elements sld:Snippet and sld:FeatureDescription each have the same content model as sld:Label, i.e. type="sld:ParameterValueType". An example that uses both follows:

```
<TextSymbolizer>
      <Label>
    <ogc:PropertyName>STATE_ABBR</ogc:PropertyName>
   </Label>
      <Font>
    <CssParameter name="font-family">Times New Roman</CssParameter>
    <CssParameter name="font-style">Normal</CssParameter>
    <CssParameter name="font-size">14</CssParameter>
   </Font>
      <Snippet>
   <ogc:PropertyName>STATE_NAME</ogc:PropertyName>
   </Snippet>
      <FeatureDescription>
This is <ogc:PropertyName>STATE_NAME</ogc:PropertyName>, where there
      are <ogc:PropertyName>DRVALONE</ogc:PropertyName> people who
      drive alone
  </FeatureDescription>
</TextSymbolizer>
```

The more flexible sld:OtherText could then support the following instances:

```
...
  <sld:TextSymbolizer>
    <sld:OtherText target="kml:BalloonStyle/kml:text">
      <sld:Literal>There are 10 types of people in the world, those who
      understand binary and those that do not.
      </sld:Literal>
    </sld:OtherText>
  </sld:TextSymbolizer>
...
  <sld:TextSymbolizer>
    <sld:OtherText target="feed/title">
      <ogc:PropertyName>abc:somePropertyName</ogc:PropertyName>
    </sld:OtherText>
  </sld:TextSymbolizer>
...
  <sld:TextSymbolizer>
    <sld:OtherText target="feed/entry/title">
      <ogc:PropertyName>abc:somePropertyName</ogc:PropertyName>
```

```
    </sld:OtherText>
  </sld:TextSymbolizer>
...
  <sld:TextSymbolizer>
    <sld:OtherText target="kml:timeStamp">
      <ogc:PropertyName>abc:somePropertyName</ogc:PropertyName>
    </sld:OtherText>
  </sld:TextSymbolizer>
...
  <sld:TextSymbolizer>
    <sld:OtherText target="channel/item/description">
      <ogc:PropertyName>abc:somePropertyName</ogc:PropertyName>
    </sld:OtherText>
  </sld:TextSymbolizer>
...
  <sld:OtherText target="BalloonStyle/text">

      <ogc:PropertyName>gml:metaDataProperty/AMLSourceMetadata/aml:dept
      hUnits</ogc:PropertyName>
    <sld:displayName>aml:depthUnits</sld:displayName>

      <sld:sourceURL>http://www.ukho.gov.uk/add/servicesAMLOandA.asp</s
      ld:sourceURL>
  </sld:OtherText>
```

The OtherText element is admittedly not as standard compliant as the other two, the group deliberately left the possible target attribute vague, so as to encourage experimentation with new formats. Eventually it is anticipated that best practices will emerge around what types of OtherText are useful, and they will be turned in to more standard elements. The testbed successfully experimented with kml timeStamps, which could likely be made more generic for time stamps in rss output as well.

The one potential downside of mirroring the structure of an SLD 'Label' element is that its method of concatenation is not conducive to embedded html content, which is used in KML and other formats to style data more extensively. The Label element was conceived as a full template language, and parsers may have more trouble figuring out the ogc:PropertyName elements than a more traditional templating syntax like is present in KML: '$[holeNumber]'. The group suggests that a more friendly and powerful template language could be a nice future addition to the SLD specification. But it should also be noted that a FilterFunction can be used to read in a specific template language, in lieu of a standardized one.

Past these limitations, the rest of the transformation from feature data and SLD to KML is fairly straight forward. The examples that follow focus on GML input, but it could be input from any data format, remote or local: it is applicable to any information that can be styled with an SLD.

11.1 SLD to KML Transform

For each sld:FeatureTypeStyle:

1. Create a kml:NetworkLink with kml:name = sld:FeatureTypeName and set the Link/href to the FPS GetMap request (using sld:OnlineResource/@xlink:href)

2. Create a kml:Folder to hold Placemarks corresponding to sld:FeatureTypeName, then transform

SLD Rule/PointSymbolizer to KML Folder/Style/IconStyle

3. SLD Rule/LineSymbolizer to KML Folder/Style/LineStyle

4. SLD Rule/PolygonSymbolizer to KML Folder/Style/PolyStyle

5. SLD Rule/TextSymbolizer to KML Folder/Style/LabelStyle or Folder/Style/BalloonStyle

6. For each feature instance corresponding to the sld:FeatureTypeStyle evaluate each sld:Rule/ogc:Filter condition, if all are true then create a kml:Placemark within the kml:Folder and transform

7. gml:name (or other designated 'name' attribute) to KML Placemark/name (used in KML BalloonStyle/text)

8. gml:description (or other designated 'description' attribute or templated) to KML Placemark/description (used in KML BalloonStyle/text)

9. GML geometry property value(s) to a KML Geometry (MultiGeometry)

10. GML simple properties to Placemark/ExtendedData which may be used in KML BalloonStyle/text

11.1.1 Sample Input sld:FeatureTypeStyle

```
<sld:FeatureTypeStyle xmlns:pre1="http://www.galdosinc.com/vancouver">
 <sld:FeatureTypeName>pre1:FerryTerminal</sld:FeatureTypeName>
 <sld:Rule>
   <sld:PolygonSymbolizer>
     <sld:Geometry>
       <ogc:PropertyName>gml:extentOf</ogc:PropertyName>
     </sld:Geometry>
     <sld:Fill>
       <sld:CssParameter name="fill">
         <ogc:Literal>rgb(0,153,153)</ogc:Literal>
       </sld:CssParameter>
       <sld:CssParameter name="fi l-opacity">
         <ogc:Literal>1.0</ogc:Literal>
       </sld:CssParameter>
```

```
      </sld:Fill>
      <sld:Stroke>
        <sld:CssParameter name="stroke">
          <ogc:Literal>rgb(211,161,141)</ogc:Literal>
        </sld:CssParameter>
        <sld:CssParameter name="stroke-width">
          <ogc:Literal>0.5</ogc:Literal>
        </sld:CssParameter>
      </sld:Stroke>
    </sld:PolygonSymbolizer>
  </sld:Rule>
</sld:FeatureTypeStyle>

<sld:FeatureTypeStyle xmlns:pre1="http://www.galdosinc.com/vancouver">
  <sld:FeatureTypeName>pre1:SingleTrack</sld:FeatureTypeName>
  <sld:Rule>
    <sld:LineSymbolizer>
      <sld:Geometry>
        <ogc:PropertyName>gml:centerLineOf</ogc:PropertyName>
      </sld:Geometry>
      <sld:Stroke>
        <sld:CssParameter name="stroke-width">
          <ogc:Literal>0.5</ogc:Literal>
        </sld:CssParameter>
        <sld:CssParameter name="stroke">
          <ogc:Literal>rgb(19,21,23)</ogc:Literal>
        </sld:CssParameter>
      </sld:Stroke>
    </sld:LineSymbolizer>
  </sld:Rule>
</sld:FeatureTypeStyle>
```

11.1.2 Sample Input GML Features

```
<van:FerryTerminal gml:id="FerryTerminal4157">
<gml:description xlink:type="simple">Buildings
      ferryTerminal</gml:description>
<gml:name/>
<gml:extentOf xlink:type="simple"
      xmlns:gml="http://www.opengis.net/gml">
  <gml:Polygon gml:id="GML_PN_8277" srsName="S57:4326">
    <gml:exterior>
      <gml:LinearRing gml:id="GML_LG_8278" srsName="S57:4326">
        <gml:coordinates cs="," decimal="." ts=" ">-
       123.08396096835904,49.309304396991436 -
       123.08407111868956,49.30935828771632 -
       123.08419502660192,49.309412168313834 -
       123.08405772865231,49.30957418065733 -
       123.08405786638691,49.30965513675558 -
       123.08426439628346,49.30975393281743 -
       123.08385232963764,49.31014102323309 -
       123.08305380229827,49.30979078968083 -
       123.08343832722508,49.30938573171023 -
       123.08369991468257,49.30950247928829 -
       123.0838236241571,49.309439423693156 -
       123.08396096835904,49.309304396991436</gml:coordinates>
      </gml:LinearRing>
    </gml:exterior>
```

```
 </gml:Polygon>
</gml:extentOf>
</van:FerryTerminal>

<van:SingleTrack gml:id="SingleTrack375">
 <gml:description xlink:type="simple">RailLines
      singleTrack</gml:description>
 <gml:name/>
 <gml:centerLineOf xlink:type="simple"
      xmlns:gml="http://www.opengis.net/gml">
   <gml:LineString gml:id="GML_LG_11830" srsName="S57:4326">
     <gml:coordinates cs="," decimal="." ts=" ">-
      123.14811161343671,49.24509530905291 -
      123.14801506098512,49.244969499474955 -
      123.14721534125555,49.24401702607731 -
      123.14699420677638,49.24357654093884</gml:coordinates>
   </gml:LineString>
 </gml:centerLineOf>
</van:SingleTrack>
```

11.1.3 Sample Output KML NetworkLink

```
<NetworkLink>
 <name>FerryTerminal</name>
 <visibility>0</visibility>
 <open>0</open>
 <Link>
      <href>…fps GetMap request…</href>
   <viewRefreshMode>onRequest</viewRefreshMode>
 </Link>
</NetworkLink>

<NetworkLink>
 <name>SingleTrack</name>
 <visibility>0</visibility>
 <open>0</open>
 <Link>
   <href>…fps GetMap request…</href>
   <viewRefreshMode>onRequest</viewRefreshMode>
 </Link>
</NetworkLink>
```

11.1.4 Sample Output KML Placemarks

```
<Folder>
  <name>FerryTerminal Placemarks</name>
  <Style id="FerryTerminalPolyStyle">
   <LineStyle>
     <color>ff8da1d3</color>
     <width>0.5</width>
   </LineStyle>
   <PolyStyle>
     <color>ff999900</color>
   </PolyStyle>
 </Style>
 <Placemark>
   <name>FerryTerminal</name>
   <description>Buildings ferryTerminal</description>
   <styleUrl>#FerryTerminalPolyStyle</styleUrl>
```

```
   <Polygon>
     <outerBoundaryIs>
       <LinearRing>
         <coordinates>-123.08396096835904,49.309304396991436,0 -
      123.08407111868956,49.30935828771632,0 -
      123.08419502660192,49.309412168313834,0 -
      123.08405772865231,49.30957418065733,0 -
      123.08405786638691,49.30965513675558,0 -
      123.08426439628346,49.30975393281743,0 -
      123.08385232963764,49.31014102323309,0 -
      123.08305380229827,49.30979078968083,0 -
      123.08343832722508,49.30938573171023,0 -
      123.08369991468257,49.30950247928829,0 -
      123.0838236241571,49.309439423693156,0 -
      123.08396096835904,49.309304396991436,0</coordinates>
       </LinearRing>
     </outerBoundaryIs>
   </Polygon>
 </Placemark>
  …
</Folder>

<Folder>
  <name>SingleTrack Placemarks</name>
  <Style id="SingleTrackLineStyle">
   <LineStyle>
     <color>ff171513</color>
     <width>0.5</width>
   </LineStyle>
 </Style>
    <Placemark>
     <name>SingleTrack</name>
     <description>RailLines singleTrack</description>
     <styleUrl>#SingleTrackLineStyle</styleUrl>
       <LineString>
         <coordinates>-123.14811161343671,49.24509530905291,0 -
      123.14801506098512,49.244969499474955,0 -
      123.14721534125555,49.24401702607731,0 -
      123.14699420677638,49.24357654093884,0</coordinates>
       </LineString>
   </Placemark>
</Folder>
```

12 Usage Scenarios

12.1 Use Case 1: Mobile Phone

Use Case Description	
Name	Mobile Phone access to online geographic data store
Priority	High
Description	A user wants to access an geographic databases, such as stored points of interest (POI), friend locations, or directions, and view these on a mobile device. In addition, the user wants to push new POI's or tracks to the online store.
Precondition	The online storage must be able to output KML, preferably given a filtered query such as geographic locality, time, or keyword search. The mobile phone needs to have either connection to the geographic storage or be able to store and load a KML document on the device.
Flow of Events – Basic Path	
f)	The user selects a data source such as an online URL endpoint or file that contains a KML document of interest.
g)	The KML document contains geographic location of the various items and pertinent attributes, such as opening hours, classification, or cost.
h)	The mobile client displays the list of locations, and attributes when a location is selected. Optionally, a map could show the locations of all or some set of locations.
i)	The user can create a new POI or other geography (e.g. photo, voice note) and store this information locally to a KML document or push back to a remote server.
Flow of Events – Alternative Paths	
Step 2	*Shows the less-common paths that need to be addressed, including situations in which unusual types of processing occur. An alternative path would be taken when an uncommon condition or interaction occurs. A single use case often includes several types of alternatives. Each alternative should indicate which step, in the basic path (flow of events) is its starting point.*
Step 3.	
Step 4.	

Use Case Description	
Postcondition	

12.2 Use Case 2: Web Mapping 1

Use Case Description	
Name	Web Mapping browsing of online sources and saving of state
Priority	Medium
Description	A user wants to explore mapping data online, from both WMS Services and KML documents. After finding a few data sources and zooming in to the area they care about, and possibly adding a few placemark annotations, they need to be able to hit 'save' and receive a document that they can load later and pass to others to see the same view.
Precondition	The mapping client is completely browser driven, likely an AJAX implementation, that requires no additional components. Thus the state must be lightweight, and large KML files will be tough to handle.
Flow of Events – Basic Path	
j)	The user selects a data source such as an online WMS or file that contains a KML document of interest. Or she loads an existing Web Map Context document for a pre-populated view.
k)	The user moves around and zooms in to her area of interest, possibly adding additional WMS services and layers.
l)	The user adds new annotation points about the area she's interested in.
m)	After completing the annotation the user can hit 'save' and get a document that she can load later or hand off to a friend that shows her exact view, by including KML placemarks in line and links to WMS services.
Flow of Events – Alternative Paths	
Step 2	*Shows the less-common paths that need to be addressed, including situations in which unusual types of processing occur. An alternative path would be taken when an uncommon condition or interaction occurs. A single use case often includes several types of alternatives. Each alternative should indicate which step, in the basic path (flow of events) is its starting point.*
Step 3.	

Use Case Description	
Step 4.	
Postcondition	

12.3 Use Case 3: Web Mapping 2

Use Case Description	
Name	Developing a Web Map Client Application
Priority	Medium
Description	A web application developer wants to create a web mapping client that supports KML styling and Component WMS requests. Depending on her needs she can choose to build upon a framework (Community Mapbuilder) or use OpenLayers directly in her application. The developer will create a user interface that allows the user to create their own Component WMS layers and have these layers added to the map.
Precondition	The developer must have some HTML and Javascript experience and have some knowledge of web mapping.
Flow of Events – Basic Path	
	The developer sets up a basic map client application using OpenLayers. Once the application works, she will create an HTML form with all the parameters needed to create a Component WMS request and fills out the default values. The submit button will call a javascript function.
	Once the javascript function is called, all the parameters are gathered into a request URL. Based upon the output format that is requested, either a WMS layer (raster) or a KML layer (vector) is created and added to the map.
	The OpenLayers library automatically styles the KML (if requested) and displays the output on the map.
Flow of Events – Alternative Paths	
	The developer can choose to implement the same functionality in Mapbuilder, making use of the power of MVC and XSL processing the framework has to

Use Case Description	
	offer.
Step 3.	
Step 4.	
Postcondition	A web page showing a map with Component WMS and KML layers

Annex A
XML Schema Documents

These XML Schema Documents combine the XML schema fragments listed in various sub-clauses of this document, eliminating duplications. The KML abilities now specified in this document use this set of specified XML Schema Documents available online at http://www.ogcnetwork.net/schemas/kml/ows5/profile/.

- kml2x_core.xsd
- kml2x_globe.xsd
- kml2x_mobile.xsd
- kml2x_web.xsd

The following scripts were used to generate the above files, and are available at http://www.ogcnetwork.net/schemas/kml/ows5/xsl/

- 3d.xsl
- kml2x_core.xsl
- kml2x_globe.xsl
- kml2x_mobile.xsl
- kml2x_web.xsl
- metadata.xsl
- styling.xsl
- time.xsl