Open Geospatial Consortium Inc.

Date: 2014-11-05

Reference number of this OGC®® project document: **OGC® 08-125r2**

Version: 0.8

Category: OGC® Best Practices Document

Editors: David Burggraf, Tim Wilson

OGC® KML Standard Development Best Practices

Copyright notice

Warning

This is an OGC® Best Practices Document. It is not an OGC® Standard and may not be referred to as an OGC® Standard. It is subject to change without notice. However, this document is an official position of the OGC® membership on this particular technology topic.

Document type:	Best Practices Document
Document subtype:	Policy
Document stage:	Draft
Document language:	English

Contents

i. Background

In early 2007, Google proposed to the OGC that KML be submitted to the OGC to become an OGC standard. In June 2007, KML 2.1 was approved as an OGC best practice document (OGC 07-039r1). The preamble to the document stated the reasons for KML becoming an OGC standard :

1. That there be one international standard language for expressing geographic annotation and visualization on existing or future web-based online 2D maps and 3D earth browsers .

2. That KML be aligned with international best practices and standards, thereby enabling greater uptake and interoperability of earth browser implementations.

3. That the OGC and Google will work collaboratively to insure that the KML implementer community is properly engaged in the process and that the KML community is kept informed of progress and issues.

4. That the OGC process will be used to insure proper life-cycle management of the KML candidate specification, including such issues as backwards compatibility.

In April 2008, the KML 2.2 specification document (OGC 07-147r2) together with the KML Abstract Test Suite (OGC 07-134r2) became an official two-part Implementation Standard of the OGC. In order to meet objective 4, the OGC and the KML implementation community needed to define a procedure for life cycle management of the OGC KML standard.

ii. Preface

This Best Practices Document provides guidance on the revision process for OGC KML. The intended audience is the OGC Mobile Location Services Domain Working Group (MLS DWG), current or future KML Standard Working Groups (SWG), and Technical Committee (TC) members as well as KML application developers and users. The OGC KML standard shall be revised in such a way that KML:

- Remains true to its purpose: encoding the visualization and navigation of information within a geographic context for earth browser systems;
- Is enhanced on the basis of proven extensions requested by the mass market;
- Provides general solutions that meet end user performance expectations within current software and hardware limitations, with due consideration for legacy software/hardware;
- Progresses on a regular and consistent revision cycle that assures the timely development of new applications required by the rapidly growing and changing mass market environment.

The guidance herein is based on presentations and discussions in the OGC Mass Market Geo Domain Working Group (MM DWG), which has since been renamed Mobile Location Services Domain Working Group (MLS DWG), on past earth browser development and other KML implementation practices, which proved successful to the growth and adoption of KML within the mass market community. Agreeing with the general process and principles identified, the MM DWG elected to summarize recommendations for continuing the successful evolution of the KML standard within a combined mass market and OGC framework. The policies and procedures summarized in this document are the result of those discussions.

iii. Submitting Organizations

The following organizations have submitted this Best Practices Paper to the Open Geospatial Consortium Inc.:

a) Google, Inc.

b) Galdos Systems Inc.

c) European Union Satellite Centre (EUSC)

d) Esri

iv. Submission Contact Points

All questions regarding this submission should be directed to the editor or submitters:

CONTACT	ORGANIZATION	EMAIL
David Burggraf	Individual	dsburggraf at gmail.com
Bent Hagemark	Google Inc.	bent at google.com
Michael Ashbridge	Google Inc.	mashbridge at google.com
Michael Weiss-Malik	Google Inc.	michaelwm at google.com
Sean Askay	Google Inc.	alchemist at google.com

v. Revision History

Date	Release	Author	Paragraph modified	Description
24-06-08	0.1	Tim Wilson, Bent Hagemark	All	Initial version.
26-06-08	0.2	Michael Ashbridge	5.2; 6.1; 6.2	Terminology edits.
30-06-08	0.3	Tim Wilson	All	Revision and completion of first draft.
02-07-08	0.4	Tim Wilson, Bent Hagemark, Michael Ashbridge	Preamble; Introduction	Reference to MMDWG presentation; list of KML clients referenced.
7/7/08	0.5	Carl Reed	Various	Prepare for posting to Pending for member review.
20-08-2008	0.6	David Burggraf	i-v, Intro, 4, 5, 6, 7	Further edits in preparation for posting to Pending on the OGC Portal

Date	Release	Author	Paragraph modified	Description
2012-01-30	0.7	David Burggraf	All	General updates to document revised as r2. In particular new KML versioning policy added.
2014-11-05	0.8	David Burggraf	8	Added Clause 8 OGC KML Encoding Best Practices.

vi. Changes to the OGC® Abstract Specification

The OGC® Abstract Specification does not require changes to accommodate this OGC® Best Practices Document.

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 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.

Introduction

The Free Dictionary defines mass market as, "*of, relating to, or produced for consumption in large numbers …*". In this regard, KML is a mass market standard that facilitates the visualization and navigation of information within a geographic context.

The current geo-mass market operating environment has the following characteristics:

- Consists of millions of users, most of whom are non-experts with respect to the geospatial domain;
- Using billions of existing and indexed KML files and resources;
- Within a large and growing list of earth browser applications;
- On fairly average and diverse equipment, including the expanding use of mobile devices;
- All of which is growing rapidly.

To support this environment, KML has been developed to date according to a process and principles whereby the language:

- Allows for unexpected and unintended uses;
- Supports multi-purpose constructs and mechanisms;
- Provides a core API that can be extended according to a well-defined model;
- Is extended incrementally to support new mass market applications;
- Changes formally only for those extensions that are proven through mass market adoption.

The rapid and widespread uptake of KML by the mass market attests to the benefits of this approach and advocates for its continuation within OGC KML standardization processes. As such, this document provides guidance on assuring similar success in the progression of the OGC KML standard.

OGC® KML Standard Development Best Practices

1 Scope

This OGC® Best Practices Document provides guidelines for developing the OGC KML standard in a manner that best serves and supports the KML application developer and user communities. It applies to the extension of KML by application developers and the subsequent enhancement of the KML standard by the OGC.

2 Conformance

There are no conformance clauses for this Best Practices Document.

3 Normative References

The following normative documents contain provisions that, through reference in this text, constitute provisions of this part of OGC® 08-125r2. For dated references, subsequent amendments to, or revisions of, any of these publications do not apply; however, parties to agreements based on this part of OGC® 08-125r2 are encouraged to investigate the possibility of applying the most recent editions of the normative documents indicated below. For undated references, the latest edition of the normative document referred to applies.

[1] OGC 12-007, OGC® KML 2.3

[2] OGC 14-068, OGC KML 2.3 - Abstract Test Suite

4 Terms and Symbols

4.1 Terms and Definitions

For the purposes of this document, the following terms and definitions apply.

4.1.1

KML Extension

An extension of the core KML language using the normative KML extension model and policies, to support new mass market applications.

4.1.2

KML Enhancement

A standardized KML extension integrated into the core KML language.

4.2 Acronyms (and Abbreviated Terms)

Some frequently used abbreviated terms:

ATS	Abstract Test Suite
DWG	Domain Working Group
GIS	Geographic Information System
HTTP	Hyper Text Transfer Protocol
IETF	Internet Engineering Task Force
KML	Keyhole Markup Language
MLS DWG	Mobile Location Services Domain Working Group
OGC	Open Geospatial Consortium
RFC	Request for Comments
SWG	Standards Working Group
UML	Unified Modeling Language
XML	Extensible Markup Language
XSD	XML Schema Definition

5 Conventions

5.1 UML Notation

There is no UML associated with this Best Practices Document.

5.2 XML Namespaces

All components of the KML schema are defined in the namespace with the identifier "`http://www.opengis.net/kml/2.2`". A KML namespace prefix can be assigned arbitrarily for any version of KML by any application, but in this best practice 'kml' or the default (no prefix) namespace is used to represent the KML namespace.

6 KML Development Process

6.1 Overview

KML was historically developed by cycling through the following development phases:

- **Respond** to the wisdom of the masses as to what new functionality KML should support;
- **Build** the new functionality as running code;
- **Assure** its compatibility and performance;
- **Specify** the KML language extension to support it;
- **Release** the KML extension API and application(s) built upon it;
- **Measure** its utility via usage statistics, performance metrics, and user feedback;
- **Formalize** it within the core language only if proven to be of general, popular, and effective use.

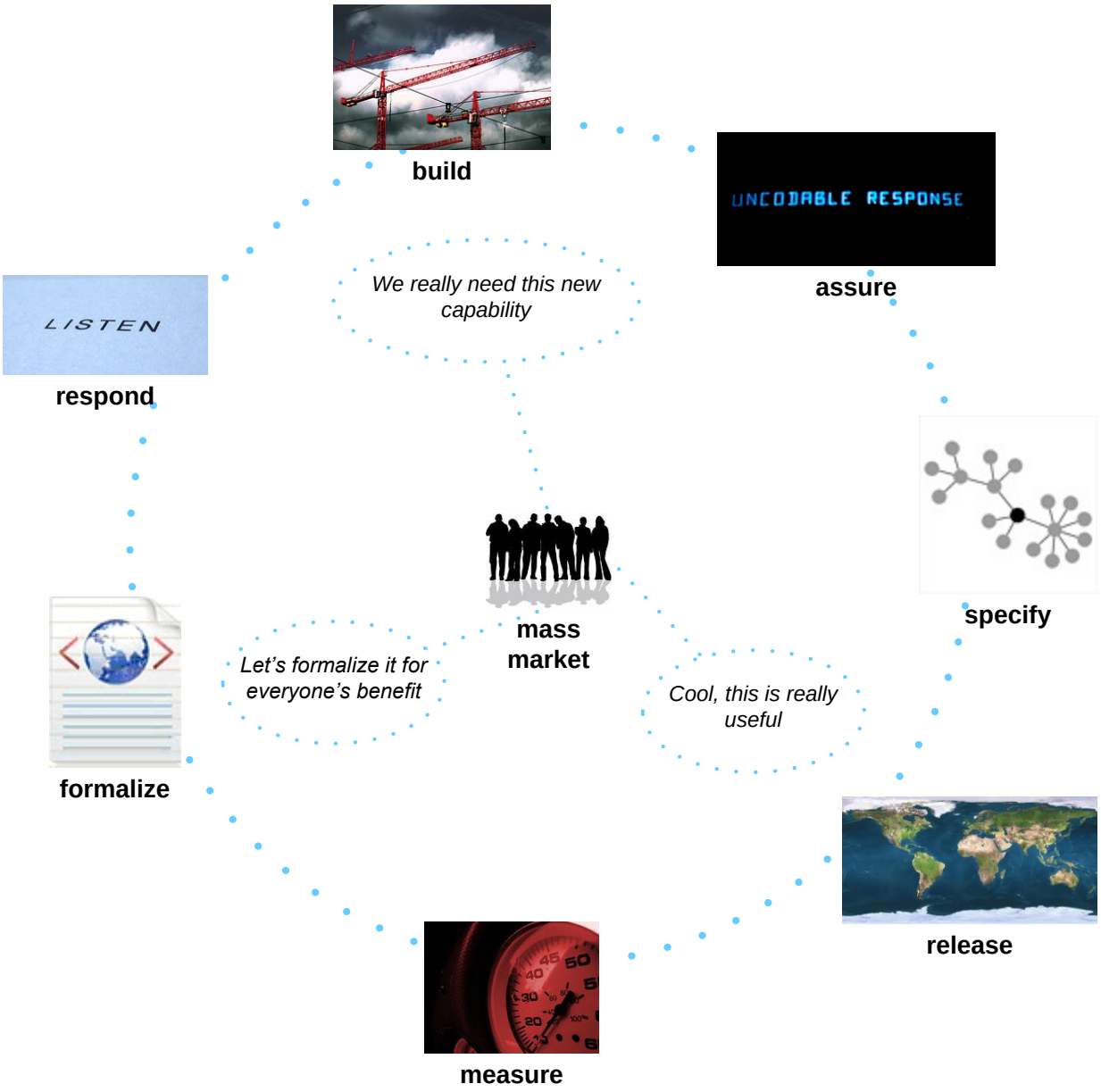

Figure 1 KML Development Phases

6.2 Development Phases

6.2.1 Respond

As with any successful social or commercial enterprise, KML has evolved according to the needs of its user base. The mass market geo community continues to express its wants and needs via bulletin boards, blogs, forums, community projects, and the OGC Mobile Location Services Domain Working Group amongst other feedback mechanisms. Developing KML in response to current and predominant mass market application requirements will continue to assure its value and relevancy.

6.2.2 Build

An overriding principle for any KML development is that the *proof is in its application*. Coding ideas early in the revision process helps to determine whether new concepts and extensions can be effectively built and integrated into existing applications while still meeting new requirements.

6.2.3 Assure

The following questions should be addressed when designing and implementing any KML extension for first minor revisions (e.g. 2.x):

- Are old implementations stable/well behaved when faced with new KML extension data? Assume that a significant percentage of existing KML implementations may never be upgraded.
- Are new implementations friendly to existing data? Assume old data exists forever and may never be changed.

In this regard, forward compatibility (see definition in 7.4.5) of KML software implementations is a paramount goal with the aim of facilitating the graceful handling of new components by old clients. In turn, the forward compatibility of software requires backward compatibility (see definition in 7.4.4) of any new KML schema with respect to older data. In practice this means KML development should continue to take responsibility for existing instance documents and old implementations for a very long time, by focusing on incremental enhancements, rather than refactoring or redesign. By maintaining this approach across minor revisions, the value of KML data and software will continue to build momentum in the mass market.

6.2.4 Specify

To support adoption and 3rd party usage, a KML extension should be well defined within an API specification document, and include sample files that demonstrate:

- normal or intended uses;

- any known edge cases and their recommended handling;
- integration with existing core KML;
- integration with the KML update mechanism (see [1], kml:Update).

Such documentation should be understandable to the average, non-expert user.

6.2.5 Release

Release the KML extension API, samples, and application(s) built upon it. Promote experimentation and seek feedback.

6.2.6 Measure

Record user feedback and usage statistics to help determine adoption rate and performance results. Pay attention to the handling of 'edge cases' i.e. combinations of extreme or omitted element values (e.g. geometry near poles or antemeridian) and revisit design, if necessary, to mitigate any unforeseen negative results.

6.2.6.1 Adoption

As mass market usage is an overriding indicator of the significance and utility of a KML extension, adoption rates should be measured and made accessible for verification.

6.2.6.2 Performance

Application performance is a predominant goal. In practice, this means KML evolution should focus on solutions that meet end user performance expectations within current software and hardware limitations.

Performance requirements should not exceed common hardware devices of non-expert users. Such devices increasingly include mobile clients.

Important considerations affecting the design, implementation, and/or standardization of any KML enhancement include:

- How does the enhancement behave on weak, limited, and/or mobile devices?
- How much texture memory does the enhancement require? How many clients right now have this much? What happens to those that still have old gear?

Performance statistics should be accessible for those exemplar applications using the new KML extension.

6.2.6.3 Edge Cases

KML enhancements should minimize the possibility of encoding any ambiguous, extreme, or meaningless values. Where this is not possible, facilitating graceful degradation within encodings and clients is encouraged.

Exemplar applications using the KML extension should test any and all known edge cases using representative sample files.

6.2.7 Formalize

Formalization of a new KML extension can occur as a last step when and where it:

- adheres to the requirements of the existing OGC KML standard;
- adheres to the best practices outlined in this document as much as possible;
- provides a satisfactory and general solution for the new functionality it provides;
- has proven itself useful through adoption within the mass market;
- would enhance the core KML language;
- is formally offered to the OGC for standardization by the owning party or parties. This includes a commitment to assign any existing intellectual property associated with the extension to the OGC.

The OGC KML standard follows a certain architecture that should persist within KML extensions and enhancements in order to maintain stable application development, facilitate the reuse of existing client code, and ease the understanding of new components.

7 OGC KML standardization

7.1 Overview

While the OGC is now the owner and forum for enhancing the KML standard, the mass market itself remains the de facto place for KML application development and therefore those KML extensions that support them.

Natural stages and roles exist within the overall progression of KML, as shown in the following diagram.

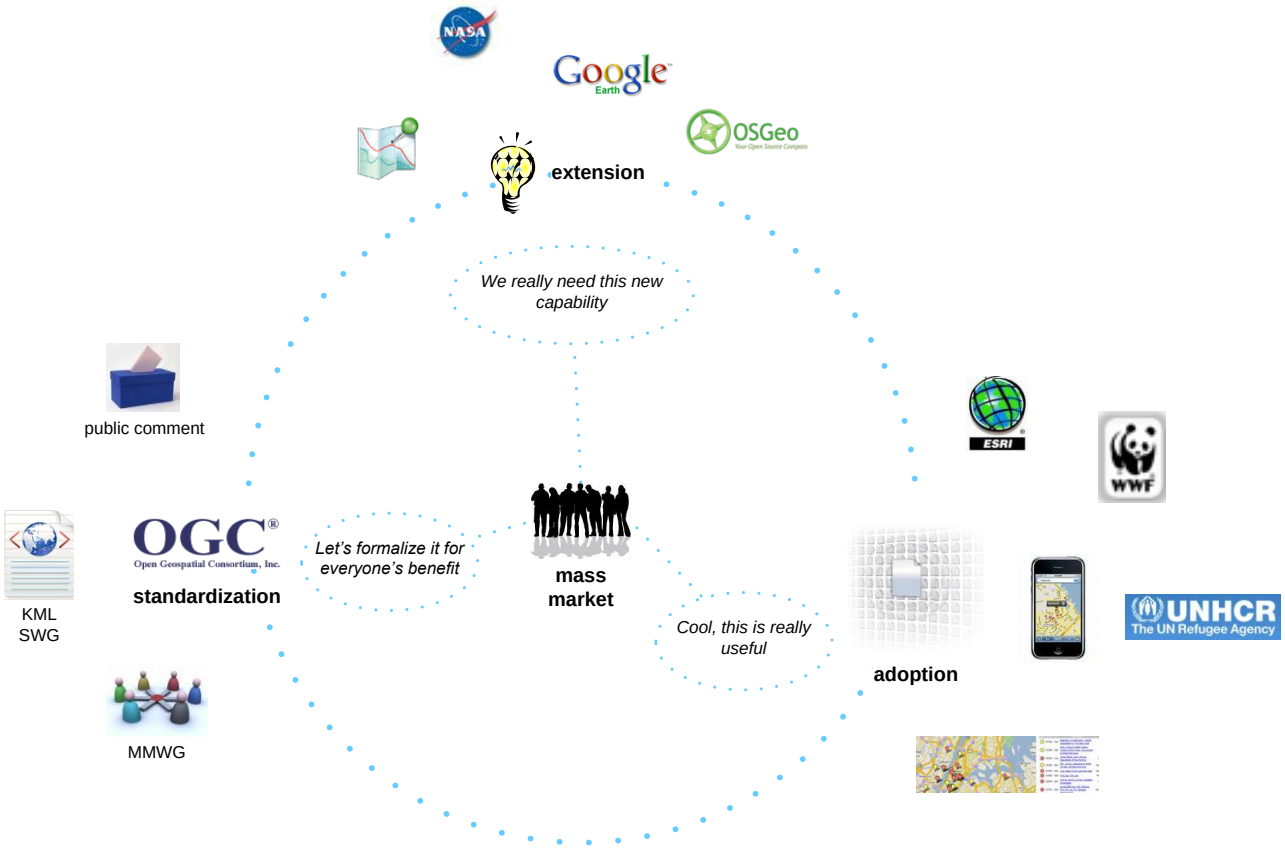

Figure 2 ·· KML Evolution

7.2 KML Progression

For simplicity of discussion, the KML development phases are summarized into three general progression stages as follows:

- **Extension** ·· This includes the **respond**, **build**, **assure**, **specify**, and **release** activities. The outcome is a KML extension API and application(s) built upon it. Such development is expected to be performed by individual vendors or organizations, although discussion amongst groups during such development is encouraged.
- **Adoption** ·· Uptake within and by the mass market of a KML extension API and originating and 3rd party applications built upon it. Performance and most critically adoption rates should be **measured** to support assessment of the extension within the standardization process.
- **Standardization** ·· **Formalize** proven KML extensions as enhancements to the core KML language. The OGC MLS DWG should provide guidance on priority enhancements, while the KML SWG conducts the technical evaluation and integration of such enhancements into the KML standard, ATS, and schema.
- A key requirement of the standardization process is that the developers of the enhancement and/or extension submit an official OGC Change Request (CR) into the OGC process. These change requests are public and may be submitted into the OGC process using the public CR submission form at: http://www.opengeospatial.org/standards/cr.
- PLEASE NOTE: No new extension or enhancement to KML will be considered for a revision to the OGC KML standard unless a formal OGC CR is submitted!

7.3 Roles

7.3.1 KML Users

The KML user community serves as the hub of mass market geo ideation, experimentation, and adoption. It includes your average Geo, not your average Geo, non-profits, academic institutions, corporations, and public agencies who are creating and/or using KML. In short, almost anyone anywhere involved with the visualization of information within a geographic context.

7.3.2 KML Application Providers

KML application providers include the larger earth browser providers, 3rd party application developers, GIS software vendors, mobile providers, and others. As they develop and provide earth browser functionality to KML users, they are best able to extend KML to meet new user requirements for earth browser technologies.

7.3.3 OGC MLS DWG

The MLS DWG, as a representative body for the mass market community, is best able to oversee the long-term development of the OGC KML standard. It represents a valuable forum for discussing feature requests and KML extensions that could satisfy them. There is likely also a role for the MLS DWG to advise on the prioritization of KML enhancements for standardization, and to arbitrate between any similar proposed enhancements.

7.3.4 OGC KML SWG

An OGC KML SWG best able to:

- establish a discrete charter with specific standardization objectives;
- assure a regular, timely and responsive KML development schedule;
- evaluate official OGC change requests and integrate proven KML extensions according to established Standardization Criteria;
- document any and all changes within a new KML standard revision (Revision notes).

To better perform these tasks, KML SWG members should preferably have mass market geo presence; participate actively in the MLS DWG; have technical expertise in KML applications and extensions; and remain actively involved in the KML SWG process.

7.4 KML SWG Process Guidelines

This section provides a synopsis of additional guidance for the processing of the OGC KML standard for formal approval as an official OGC revision.

7.4.1 Charter

A KML SWG should assure that KML remains true to its purpose: encoding the presentation and navigation of information within a geographic context.

The charter for KML SWGs should respect the KML development best practices outlined in this document.

A KML SWG should focus on incrementally enhancing the KML standard by evaluating and integrating KML extensions that have already been proven in the mass marketplace.

Only if there is sufficiently valuable and well-defined mass market needs for an application that *cannot* be accommodated by the current language primitives, shall a backwards-*in*compatible revision be contemplated. It is expected that the MLS DWG will advise on both the need and timing for any such major revision.

7.4.2 Timeline

KML should evolve in a manner that satisfies the mass market need for regular and incremental enhancements of functionality. In practice this means a KML SWG should limit its scope of work to that which can be achieved within at most an annual release cycle.

A regular and consistent KML revision cycle will help to assure commercial development of earth browser technologies and applications within the rapidly changing mass market geo environment.

7.4.3 Standardization Criteria

A KML extension should be evaluated on the basis of how well it:

- is consistent with the purpose, architecture, and requirements of KML;
- is consistent with the KML standard development best practices;
- enables new mass market application(s) that are otherwise not supported by the existing KML core primitives;
- is proven through significant and verifiable adoption within the mass market;
- provides a general solution that meets end user performance expectations within current software and hardware limitations;
- is backwards compatible with previous minor KML revisions;

- includes a change request document meeting OGC requirements, as well as a set of test files for normal, edge, and update cases.

7.4.4 Backward Compatibility of KML

The term backward compatibility for KML is defined in terms of KML instance documents as follows:

> *An instance document from a previous minor version of a schema shall validate against the new minor revision of the schema.*

The above backward compatibility definition in terms of instances implies the following backward compatibility constraints to KML schema:

1. All elements or attributes added to previously existing KML elements in the new schema version shall be declared optional.

2. New elements/types are allowed to be added to the new schema, but are not required to appear in an instance.

3. Existing elements/types in previous versions cannot be removed in in the new version (but can be labelled deprecated, meaning support may no longer exist in the next major release).

7.4.5 Forward Compatibility of KML Implementations

The definition above for backward compatibility of KML instances is defined in such a way as to enable forward compatibility of older tools and implementations.

> *A KML application is forward compatibile if it can effectively process a newer KML data instance, by ignoring the newer extensions.*

This implies that a KML instance (or portions of) generated by newer (or future) devices can still be read, viewed, etc., by an older KML application, in a sensible way with little or no additional adaptation of the code. Any new extensions encountered by the old software are gracefully ignored.

A KML SWG should assure backwards compatibility by testing against a normative set of KML test files. The normative set should include a large corpus of instances for all previous KML versions, starting with KML 2.1. The test coverage will naturally expand over time as a result of ongoing KML revisions. The KML SWG may additionally elect to test the compatibility using a reference KML parser such as the open-source libkml library.

7.4.6 KML Enhancement

When enhancing the core KML language, a KML SWG should integrate an acceptable KML extension with as little disruption as possible to existing users and applications of KML as well as the extension itself. In practice this means using the same KML extension element and attribute names and structures as much as possible.

A new KML standard revision should address necessary changes to the KML standard document, abstract test suite, and XML schema to incorporate the enhancement.

7.4.6.1 Versioning

A first minor revision (i.e. `X.Y+1`) of the KML Standard (i.e. X.Y) implies that only backwards compatible incremental enhancements have been made,; everything that validates against `X.Y` shall validate against `X.Y+n`.

A small, immediate and necessary fix to the KML Standard or XML schema shall result in a second minor (bug fix) revision of the Standard, i.e. `X.Y.Z+1`; everything that validates against `X.Y.Z` shall validate against `X.Y.Z+n`, excepting those instances that are invalid with respect to the fixes themselves. `X.Y.Z+n` shall *not* introduce any new functionality from `X.Y.Z`

A backwards-*in*compatible revision shall result in a new major version of the KML Standard, i.e. `X+1.0.0`. Major revisions are expected to rarely occur.

7.4.7 Public Comment

Draft KML enhancements must be posted for a 30 day public comment as per OGC SWG requirements. Feedback should be evaluated within the context of the standardization criteria.

7.4.8 KML Revision

A KML revision, encompassing a new KML standard document, abstract test suite, and XML schema, must receive approval by the OGC Technical Committee (TC) before it is released as the next KML standard.

8 OGC KML Encoding Best Practices

8.1 Enable Dynamic Data Typing for kml:Data Values

In KML 2.3, the `kml:value` type was changed from `xsd:string` to `xsd:anySimpleType` so that in a KML instance document, an optional `xsi:type` attribute can be used to specify one of the XML Schema primitive data types. If the `xsi:type` attribute is present, an XML Schema validator can automatically enforce that `kml:value` is consistent with the datatype specified in the `xsi:type` attribute.

8.1.1 Example

In the following sample instance the attribute `xsi:type="xs:int"` is added to one of the `kml:value` elements.

```
<kml>
  <Document>
    <name>ExtendedData+Data</name>
    <open>1</open>
    <Placemark>
      <name>Easy trail</name>
      <ExtendedData>
        <Data name="TrailHeadName">
          <value>Pi in the sky</value>
        </Data>
        <Data name="TrailLength">
          <value>3.14159</value>
        </Data>
        <!-- datatype mismatch error in the following Data value -->
        <Data name="ElevationGain">
          <value xsi:type="xs:int">10.0</value>
        </Data>
      </ExtendedData>
      <Point>
        <coordinates>-122.000,37.002</coordinates>
      </Point>
    </Placemark>
  </Document>
</kml>
```

XML Schema Validation (using Xerces-J) in Oxygen 14 reports the following error message:


```
<Document>
  <name>ExtendedData+Data</name>
  <open>1</open>
  <Placemark>
    <name>Easy trail</name>
    <ExtendedData>
      <Data name="TrailHeadName">
        <value>Pi in the sky</value>
      </Data>
      <Data name="TrailLength">
        <value>3.14159</value>
      </Data>
      <!-- datatype mismatch error in the following Data value -->
      <Data name="ElevationGain">
        <value xsi:type="xs:int">10.0</value>
      </Data>
    </ExtendedData>
    <Point>
      <coordinates>-122.000,37.002</coordinates>
    </Point>
  </Placemark>
</Document>
</kml>

E [Xerces] cvc-type.3.1.3: The value '10.0' of element 'value' is not valid.
```

8.2 Validate kml:SimpleData for Datatype Consistency

In KML, the datatype of a `kml:SimpleData` value is specified in a KML instance by the `type` attribute on `kml:SimpleField` (see [1], `kml:SimpleField`.) To verify the consistency between the `kml:SimpleData` value and the `kml:SimpleField/@type` value (i.e. a 'co-constraint') a rule-based assertion can be used. Both Schematron and XSD 1.1 allow for the creation and enforcement of rule-based assertions, but due to the 'look-down-only' scope limitation of XSD 1.1 assertions, a Schematron assertion is the more appropriate option for checking the datatype consistency of `kml:SimpleData`. Such an example is provided in Section 8.2.1.

It is also possible in KML 2.3 to use an `xsi:type` attribute to specify the datatype of `kml:SimpleData` and then use regular XML Schema validation to enforce the datatype consistency against the `xsi:type` attribute value. Unfortunately, it is not as simple as the case for `kml:Data/kml:value` (see Section 8.1), because `kml:SimpleData` is of complex type. `kml:SimpleDataType` extends from the simple ur-type `xsd:anySimpleType` by adding attributes (which is what makes it a complex type), so the `xsi:type` attribute can be provided to specify a more restricted type and enable datatype enforcement in an instance using XML Schema validation. However, unlike the case of `kml:value`, the valid values of the

xsi:type attribute are not simple datatypes, rather they are complex types that must be validly derived from kml:SimpleDataType in a schema outside the KML namespace as shown in Section 8.2.2.

8.2.1 Schematron

In this section a Schematron assertion is used to check the data type consistency between kml:SimpleData value and kml:SimpleField/@type. An XPath expression in a Schematron <assert/> can be used to test if a kml:SimpleData value is 'castable as' the type entered in kml:SimpleField/@type. The XPath expression: 'V castable as T' returns true if the value V can be cast as the datatype T and false otherwise. Schematron also allows for custom, flexible error messages to be created if an assertion fails - a sample with custom error message is encoded as follows:

```
<sch:assert test="$v castable as xs:int">
     The value of 'SimpleData' (<sch:value-of select="$v"/>) must be castable as
the corresponding  'SimpleField' attribute 'type' (<sch:value-of select="$t"/>)
</sch:assert>
```

A Schematron 'report' can also be used to flag a warning, e.g. by checking to see if kml:SimpleField/@type is one of the datatypes listed in either the OGC KML Standard or the KML 2.2 Reference guide, if not, a warning message is thrown saying something to the effect of: 'Warning! Type may not be supported ...':

```
<sch:report test="not($typeSupported)">
     Warning! Type may not be supported - the value of 'SimpleField/@type'
     (<sch:value-of select="$t"/>) is not listed in the OGC KML standard or
     reference docs.
</sch:report>
```

Now consider the following KML instance with a data type mismatch error in a kml:SimpleData value ('10.0' does not match the xsd:int datatype specified in kml:SimpleField/@type).


```
<kml>
  <Document>
    <name>ExtendedData+SchemaData</name>
    <open>1</open>
    <!-- Declare the type "TrailHeadType" with 3 fields -->
    <Schema name="TrailHeadType" id="TrailHeadTypeId">
      <SimpleField type="xsd:string" name="TrailHeadName">
        <displayName><![CDATA[<b>Trail Head Name</b>]]></displayName>
      </SimpleField>
      <SimpleField type="xsd:double" name="TrailLength">
        <displayName><![CDATA[<i>Length in miles</i>]]></displayName>
      </SimpleField>
      <SimpleField type="xsd:int" name="ElevationGain">
        <displayName><![CDATA[<i>Change in altitude</i>]]></displayName>
      </SimpleField>
    </Schema>
    <Placemark>
      <name>Easy trail</name>
      <ExtendedData>
        <SchemaData schemaUrl="#TrailHeadTypeId">
          <SimpleData name="TrailHeadName">Pi in the sky</SimpleData>
          <SimpleData name="TrailLength">3.14159</SimpleData>
          <!-- data type mismatch error in the following SimpleData value -->
          <SimpleData name="ElevationGain">10.0</SimpleData>
        </SchemaData>
      </ExtendedData>
      <Point>
        <coordinates>-122.000,37.002</coordinates>
      </Point>
    </Placemark>
    <Placemark>
      <name>Difficult trail</name>
      <ExtendedData>
        <SchemaData schemaUrl="#TrailHeadTypeId">
          <SimpleData name="TrailHeadName">Mount Everest</SimpleData>
          <SimpleData name="TrailLength">347.45</SimpleData>
          <SimpleData name="ElevationGain">10000</SimpleData>
        </SchemaData>
      </ExtendedData>
      <Point>
        <coordinates>-121.998,37.0078</coordinates>
      </Point>
    </Placemark>
  </Document>
</kml>
```

If the KML instance is validated against a Schematron schema, the error message in Oxygen 14 appears as follows:


```
<Document>
  <name>ExtendedData+SchemaData</name>
  <open>1</open>

  <!-- Declare the type "TrailHeadType" with 3 fields -->
  <Schema name="TrailHeadType" id="TrailHeadTypeId">
    <SimpleField type="xsd:string" name="TrailHeadName">
      <displayName><![CDATA[<b>Trail Head Name</b>]]></displayName>
    </SimpleField>
    <SimpleField type="xsd:double" name="TrailLength">
      <displayName><![CDATA[<i>Length in miles</i>]]></displayName>
    </SimpleField>
    <SimpleField type="xsd:int" name="ElevationGain">
      <displayName><![CDATA[<i>Change in altitude</i>]]></displayName>
    </SimpleField>
  </Schema>
  <Placemark>
    <name>Easy trail</name>
    <ExtendedData>
      <SchemaData schemaUrl="#TrailHeadTypeId">
        <SimpleData name="TrailHeadName">Pi in the sky</SimpleData>
        <SimpleData name="TrailLength">3.14159</SimpleData>
        <!-- data type mismatch error in the following SimpleData value -->
        <SimpleData name="ElevationGain">10.0</SimpleData>
      </SchemaData>
    </ExtendedData>
    <Point>
      <coordinates>-122.000,37.002</coordinates>
    </Point>
```

E [ISO Schematron (XSLT 2.0)] The value of 'SimpleData' (10.0) must be castable as the corresponding 'SimpleField' attribute 'type' (xsd:int)

8.2.2 XMLSchema

In this section, two methods are illustrated to enable an XML Schema parser to validate the datatype of a `kml:SimpleData` value: the first using the `xsi:type` attribute; and the second using element substitution. In KML 2.3, `kml:SimpleDatatype` was re-defined in a backwards compatible way to derive from `xsd:anySimpleType` instead of `xsd:string`. This allows for XML Schema data type validation using the `xsi:type` attibute, or equivalently using an element substitution - both examples are shown below. First a new schema type is created (not in the KML namespace), such as `ex:SimpleDataIntType` as a restriction of `kml:SimpleDataType`:


```
  <complexType name="SimpleDataIntType">
    <simpleContent>
      <restriction base="kml:SimpleDataType">
        <simpleType>
          <restriction base="int"/>
        </simpleType>
      </restriction>
    </simpleContent>
  </complexType>
```

Now an XML Schema validator can validate the `kml:SimpleData` value against the above data type specified by `xsi:type="ex:SimpleDataIntType"`. Consider the following instance fragment with a data type mismatch error in a `kml:SimpleData` value ('10.0' instead of '10').

```
...
    <Placemark>
      <name>Easy trail</name>
      <ExtendedData>
        <SchemaData schemaUrl="#TrailHeadTypeId">
          <SimpleData name="TrailHeadName">Pi in the sky</SimpleData>
          <SimpleData name="TrailLength">3.14159</SimpleData>
        <!-- data type mismatch error in the following SimpleData value -->
          <SimpleData  xsi:type="ex:SimpleDataIntType"
name="ElevationGain">10.0</SimpleData>
        </SchemaData>
      </ExtendedData>
      <Point>
        <coordinates>-122.000,37.002</coordinates>
      </Point>
    </Placemark>
...
```

A sample screenshot of the XML Schema validation error message in Oxygen 14 (using Saxon-EE 9.4) is as follows:

Another validator, Xerces-J, reports a similar error: *cvc-datatype-valid.1.2.1: '10.0' is not a valid value for 'integer'*.

Instead of using the `xsi:type` attribute, we could declare an element `ex:SimpleDataInt` that substitutes for `kml:SimpleData` as follows:

```
<element name="SimpleDataInt" type="ex:SimpleDataIntType"
     substitutionGroup="kml:SimpleData"/>
```

Now, instead of inserting the `xsi:type="kml:SimpleDataIntType"` attribute on `kml:SimpleData`, we can replace it with the new element `ex:SimpleDataInt`:


```
...
      <Placemark>
        <name>Easy trail</name>
        <ExtendedData>
          <SchemaData schemaUrl="#TrailHeadTypeId">
            <SimpleData name="TrailHeadName">Pi in the sky</SimpleData>
            <!-- validation test: data type mismatch errors in next two
SimpleData values -->
            <SimpleData name="TrailLength">3.14159</SimpleData>
            <SimpleDataInt name="ElevationGain">10.0</SimpleData>
          </SchemaData>
        </ExtendedData>
        <Point>
          <coordinates>-122.000,37.002</coordinates>
        </Point>
      </Placemark>
...
```

Then XML Schema validation (this time using Xerces-J in Oxygen 14) will flag a type mismatch error as it did for the `xsi:type` instance:

```
3    <Document>
4      <name>ExtendedData+SchemaData</name>
5      <open>1</open>
6
7      <!-- Declare the type "TrailHeadType" with 3 fields -->
8      <Schema name="TrailHeadType" id="TrailHeadTypeId">
9        <SimpleField type="xsd:string" name="TrailHeadName">
10         <displayName><![CDATA[<b>Trail Head Name</b>]]></displayName>
11       </SimpleField>
12       <SimpleField type="xsd:double" name="TrailLength">
13         <displayName><![CDATA[<i>Length in miles</i>]]></displayName>
14       </SimpleField>
15       <SimpleField type="xsd:decimal" name="ElevationGain">
16         <displayName><![CDATA[<i>Change in altitude</i>]]></displayName>
17       </SimpleField>
18     </Schema>
19     <Placemark>
20       <name>Easy trail</name>
21       <ExtendedData>
22         <SchemaData schemaUrl="#TrailHeadTypeId">
23           <SimpleData name="TrailHeadName">Pi in the sky</SimpleData>
24           <SimpleData name="TrailLength">3.14159</SimpleData>
25           <!-- data type mismatch error in the following SimpleData value -->
26           <ex:SimpleDataInt name="ElevationGain">10.0</ex:SimpleDataInt>
27         </SchemaData>
28       </ExtendedData>
29       <Point>
30         <coordinates>-122.000,37.002</coordinates>
31       </Point>
E [Xerces] cvc-datatype-valid.1.2.1: '10.0' is not a valid value for 'integer'.
```


Bibliography

[1] OGC 12-007, OGC® KML 2.3

