Open Applications Group – Position Paper
OAGi Versioning Position
February 6, 2006

Project Team Leader:
Michael Rowell - OAGi

Authors:
Michael Rowell – OAGi
Kurt Kanaskie – Lucent Technologies

Reviewers:
David Connelly - OAGi
Michelle Vidanes – STAR
Dave Carver - STAR
Joe Zhou – Xtensible Solutions

Document Number: 050916-1-Draft

NOTICE

The information contained in this document is subject to change without notice.

The material in this document is published by the Open Applications Group, Inc. for evaluation. Publication of this document does not represent a commitment to implement any portion of this specification in the products of the submitters.

WHILE THE INFORMATION IN THIS PUBLICATION IS BELIEVED TO BE ACCURATE, OPEN APPLICATIONS GROUP, INC. MAKES NO WARRANTY OF ANY KIND WITH REGARD TO THIS MATERIAL INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANT ABILITY AND FITNESS FOR A PARTICULAR PURPOSE. Open Applications Group, Inc. shall not be liable for errors contained herein or for incidental or consequential damages in connection with the furnishing, performance or use of this material.

For more information, contact:
Open Applications Group, Inc.
P.O. Box 4897
Marietta, Georgia 30061 USA
Telephone: 1.770.943.8364
Internet: http://www.openapplications.org

Table of Contents

OAGi Versioning

Abstract

The one constant in the world is change. Just as business, supply chains and applications change so must the standards used to connect them. These changes may result from changing conditions and changing integration requirements.

This is not new to XML or XML schema. Every application in use in the world today must have a versioning plan and versioning strategy that is used to determine what constitutes a version change.

To support integration, a business language must be defined with these changes in mind and have a versioning plan and strategy built-in. This document describes the versioning plan and strategy as it applies to the OAGi standards including OAGIS and OAGIS 9.0.

Pertinent industry best practices are identified to provide background and foundation for the OAGi approach.

1.0 Overview

If we simply recognize that the one constant in business and for that matter the world is change, there is no question as to why we need to manage changes in any artifact whether it is a document like this, an application, or a business language used for integration.

The *W3C Architecture of the World Wide Web, Volume One Recommendation 15 December 2004* describes the need for versioning as:

> *"In a perfect world, language designers would invent languages that perfectly met the requirements presented to them, the requirements would be a perfect model of the world, they would never change over time, and all implementations would be perfectly interoperable because the specifications would have no variability."*

Versioning is how organizations deal with an imperfect world.

Fundamental capabilities of versioning that must be addressed are identification and compatibility between versions. Many organizations employ the concept of major and minor version changes that is reflected in their versioning implementation. Generally, major version changes imply significant changes breaking compatibility in contrast to minor version changes that preserve interoperability.

Other approaches to versioning include date and numbering schemes that provide temporal information.

The challenge for OAGi is to determine how to support these concepts through the implementation of OAGIS in XML (e.g. Namespaces, Attributes, SchemaLocations).

2.0 VERSIONING BEST PRACTICES

There are several requirements associated with versioning in XML instance documents and XML Schema that must be supported:

1. Minimize changes that break compatibility between versions.
2. Embed version information within XML instances and XML Schemas such that it can be programmatically detected.
3. Make previous versions of the XML Schema accessible.

2.1 Version Compatibility

There are two types of version compatibility: backward compatibility and forward compatibility. XML.com article by David Orchard December 03, 2003 describes these as follows:

> "Backwards compatibility means that a new version of a receiver can be rolled out so it does not break existing senders. This means that a sender can send an old version of a message to a receiver that understands the new version and still have the message successfully processed.
>
> Forwards compatibility means that an older version of a receiver can consume newer messages and not break. Of course the older version will not implement any new behavior, but a sender can send a newer version of a message and still have the message successfully processed.
>
> In other words, backwards compatibility means that existing senders can use services that have been updated, and forwards compatibility means that newer senders can continue to use existing services.
>
> Forwards-compatible changes typically involve adding optional element(s) and/or attribute(s). The costs associated with introducing changes that are not backwards- or forwards-compatible are often very high, typically requiring deployed software to be updated to accommodate the newer version."

A key point from the excerpt above is that the cost of incompatible changes is often high due to the need to modify deployed solutions.

2.1.1 Major Versions

Major versions are those that break compatibility. Many organizations attempt to minimize costs associated with major version upgrades by collecting several changes that break backward compatibility into a single upgrade some time after the last major release that broke compatibility.

2.1.2 Minor Versions

Minor versions are those that do not break compatibility. Although technical changes to a specification are necessary to reflect version information, modifications to existing implementations are minimized and do not involve significant (costly) programmatic changes.

Minor versions are generally limited to optional content additions that are not required by the XML Schema definition. Changes that remove content are excluded since they would break compatibility.

2.1.3 OAGIS Change Management

Major version releases proposed by the project team must be validated through business drivers and approved by the OAGi Board of Directors. The Architecture Team and the Project Team are tasked with implementing the changes. The changes are then reviewed and approved by the Technical Team and the Board of Directors. These steps are further defined in the OAGi Open Development Methodology available at the Open Applications Group Web site.

OAGi strives to make all releases minor thereby maintaining compatibility among versions. However due to changing technical conditions such as the Advancement of XML and XML Schema, the Advancement of Core Components, and changing business requirements major version changes will be required.

OAGi has only made three major version changes to OAGIS based on the description of Major and Minor herein. Prior to OAGIS 8.0, OAGi considered significant content additions as criteria to increment the major release number. Based on generally accepted practices, this policy will no longer be followed.

Here are the major releases of OAGIS and the reason for the major change:

OAGIS 1.0 – Initial Release

OAGIS 2.0 – New major content area added

OAGIS 3.0 – New major content area added

OAGIS 4.0 – New major content area added

OAGIS 5.0 – New major content area added

OAGIS 6.0 – Added support for XML/DTDs.

OAGIS 7.0 – New major content area added

OAGIS 8.0 – Added full support for XML Schema

OAGIS 9.0 – Added full support for Core Components

2.2 Version Location

There are at least four options for identifying the version of a schema:

1. Use the built in internal schema version attribute provide by the schema recommendation.

2. Use a schema version attribute that the schema author defines. (This is in addition to the version attribute that the XML Schema Recommendation provides.)

3. Use the XML Schema target namespace.

4. Use the XML Schema location in the schemaLocation attribute.

Each approach has advantages and disadvantages.

2.2.1 Internal Schema Version Attribute

In this approach the user simply changes the number contained in the optional version attribute of the schema root element for every XML Schema file.

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns="http://www.openapplications.org/oagis/9"
targetNamespace="http://www.openapplications.org/oagis/9"
elementFormDefault="qualified"
attributeFormDefault="unqualified" version="9.0">
```

Advantages

- Part of the XML Schema specification, easy to use.
- Instance documents are forward compatible and do not need to change, as long as they are still valid with the new version of the schema.
- The schema embeds version information.

Disadvantage

- Parsers ignore the version number, therefore it is not enforceable.
- The XML instances do not version information so there is no way to determine which schema should be used for validation.

2.2.2 Create a Version attribute on the Root Element of the Document

In this approach, the schema author defines a version attribute that contains the version information of the schema definition on the root element. This attribute could be used two different ways.

2.2.2.1 Fixed Use

Similar to 2.2.1 above the attribute could be provided in the schema, and assigned a fixed value. For example:

```
<xs:schema xmlns="http://www.exampleSchema"
targetNamespace="http://www.exampleSchema"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified"
attributeFormDefault="unqualified">
<xs:element name="Example">
<xs:complexType>
….
<xs:attribute name="schemaVersion" type="xs:decimal"
use="required" fixed="1.0"/>
</xs:complexType>
</xs:element>
```

Advantages

- The schemaVersion attribute is now enforceable. Instances would not validate without the same version number.

Disadvantage

- The schemaVersion number in the instance must match exactly, precluding forward compatibility. An instance can only be validated by a single schema.

2.2.2.2 Instance Use

In this use, the schemaVersion attribute is populated in the XML Instance and is NOT a fixed value defined in the schema. In this case it is used in the instance to indicate the version (or versions of the schema that are compatible. This approach must be used in conjunction with a means to identify the schema file to be used.

The value of the schemaVersion attribute could be a list or use any other convention to define its use. With this approach an application could compare the schema version with the version to which the instance reports that it is compatible. For example:

```
<xs:schema xmlns="http://www.exampleSchema"
targetNamespace="http://www.exampleSchema"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified"
attributeFormDefault="unqualified"
version="1.3">
<xs:element name="Example">
<xs:complexType>
….
<xs:attribute name="schemaVersion" type="xs:decimal"
use="required"/>
</xs:complexType>
</xs:element>
```

A sample instance (declares it is compatible with version 1.2 (or 1.2 and other versions depending upon the convention used))

```
<Example schemaVersion="1.2"
xmlns="http://www.example"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.example
MyLocation\Example.xsd">
```

Advantages

- Instance documents can be validated with multiple Schemas, thereby supporting backward and forward compatibility.
- An application has an indication that the schema has changed.
- Can be used in conjunction with the schemaLocation attribute to determine if the XML Schema exists. Otherwise it can be used to indicate the repository location.

Disadvantage

- Requires an application to capture the version in which the instance declares itself valid.

2.2.3 Change the Schema's Target Namespace

In this approach, the schema's targetNamespace is changed to indicate that a new version of the schema exists. One approach is to include a schema version number in the designation of the target namespace. For example:

```
<xs:schema xmlns="http://www.exampleSchemaV1.0"
targetNamespace="http://www.exampleSchemaV1.0"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified"
attributeFormDefault="unqualified">
```

Advantages

- Applications are made aware of a change to the schema through an unrecognizable namespace.

- Requires action to assure that there are no compatibility issues with the new schema. At a minimum, the instance documents that use the schema must change to reference the new namespace.

Disadvantage

- XML Instances will not validate until they use the new targetNamespace. Minor changes require all instance documents to change.

- All schemas including this schema would also have to change.

2.2.4 Change the Location of the Schema

This approach changes the file location of the schema. This convention reflects the version information in the file name.

Advantages

- Allows the author of an instance to point into the repository of the schemas to indicate the version from which the instance was generated.

- Could be used with a relative reference in conjunction with version attribute of the root element where the content provided by the author of the instance indicates the version(s) with which it is compatible. The user can then determine which version(s) they support and point to the correct repository.

Disadvantage

- This approach forces all instances to change unless the relative approach for access to the schema repository is used as indicated above.
- The schemaLocation attribute in the instance is optional and is not authoritative even if it is present. It is a hint to help the processor to locate the schema.

2.3 Dated Version Approach

Some organizations use the year in the namespace to indicate a different repository and version. This has caused confusion when identifying the version.

Practically speaking, there is really only one way to do versioning in XML Schema which is used by almost all organizations today. That is to use a numbering system that allows for major and minor revisions as discussed previously.

While the W3C and others use the year in which a recommendation becomes a recommendation in the namespace for that standard, it is always referred to for example as XML 1.0 and not XML 1998.

3.0 OAGIS 9.0 IMPLEMENTATION

To fully comprehend OAGi's versioning it is necessary to understand the design rules used in OAGIS 9.0 They are as follows:

1. Allow extensibility – OAGIS is designed for extensibility through a documented procedural model.
2. Namespace – Allow extensibility to occur in any namespace other than the OAGIS namespace.
3. Full extensibility – All elements should allow for element extension.
4. Process Model Support – OAGIS supports use within process models.
5. Must Ignore – OAGi recommends that document receivers ignore any XML artifacts that they do not recognize.
6. Must Ignore All – Must ignore unrecognized elements and all of their descendants.
7. Must Ignore Container – Applies only to unrecognized elements.

8. Re-use namespace names – If a backwards compatibility change can be made then the old namespace name WILL BE used in conjunction with XML's extensibility model.

9. New namespaces are used to indicate a break in backward compatibility – This means that software that does not understand the new additions will beak.

10. Support constraint specification external to XML Schema – OAGIS provides constraints that a receiver and sender must understand outside of the XML Schema.

11. Be Deterministic – Extensions must be able to be determined by the XML Schema definition.

The following conventions have been established to achieve these design rules and to follow the identified best practices for versioning while maximizing the advantages and minimizing the disadvantages of the various approaches.

3.1 Version Numbering System for Major and Minor Releases

OAGIS will use a major and minor versioning number system following the pattern M.N where M represents the major release number and N represents the minor release number. The decimal point is the separator. For example OAGIS 9.0 is the first major version or release of OAGIS that incorporates UN/CEFACT Core Components in OAGIS. Doing this breaks backward compatibility between OAGIS 8.0 and OAGIS 9.0.

The second number indicates a minor release. A minor release adds optional content or loosen restrictions to OAGIS and does not break compatibility. New content typically is provided by work groups within the major release. For example OAGIS 7 had several minor releases. OAGIS 7.1, and OAGIS 7.2.

Bug fixes are another source of changes associated with minor releases such as documentation updates, schema typing, etc. A bug fix release must not break compatibility, if the change required were to break compatibility the previous element would be deprecated in favor of the new element. Examples of this can also be seen in OAGIS 7.0 and 7.2 with 7.2 having an OAGIS 7.2.1.

OAGIS 8.0 also introduced "Service Patches" releases that provide delta changes made to the schemas for bug fix releases. Instead of reissuing the entire schema set only the schema files that were updated with the changes to those files were provided in the patch. The full version made available from the OAGi web site for new downloads of OAGIS had these "Service Patches" already applied. Since these service patches are minor OAGi collects several of these before making a full repackage for a bug fix release.

OAGIS 9.0 continues to follow these same conventions.

3.2 Version Location

As it is easy to see from the Best Practices - Version Location sections above, there are options each having advantages and disadvantages. OAGi leverages these practices to maximize advantages and minimize disadvantages.

In addition to providing a means to version OAGIS, there is a need to provide a consistent means of versioning extensions to OAGIS. Extensions are made through the use of an Overlay or the UserArea.

3.2.1 In the Namespace

OAGIS 9.0 incorporates the major release number of OAGIS in the namespace. The namespace for OAGIS 9.0 is: http://www.openapplications.org/oagis/9. Each XML Schema file that is part of the OAGIS 9.0 namespace has a schema root element that looks similar to the following:

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns="http://www.openapplications.org/oagis/9"
targetNamespace="http://www.openapplications.org/oagis/9"
elementFormDefault="qualified"
attributeFormDefault="unqualified">
```

Note that the number 9 indicates the major version release. The minor version numbers are not carried in the namespace. This supports the definition that minor versions do not break compatibility.

The namespace occurs both within the schema and is referenced on all XML instances that use the schemas, including those that are created via an OAGIS Overlay.

All 9.x versions are backwards and forwards compatible as defined earlier. As minor releases introduce new optional content, technical validation and interoperability with previous implementations is only possible when new content is not included in the message. This approach facilitates change. Changes to existing implementations can be minimized to the deployment of new Schemas and changes to support additional content can be localized to new implementations.

Advantages

- Applications can detect the namespace programmatically and provide exception handling.

- Requires programmatic changes to assure that there are no compatibility issues with the major releases. At a minimum, the instance documents must change to reference the new namespace defined for the major release.

3.2.2 In the SchemaLocation

The schemaLocation used on the XML instance indicates the repository to which the given XML Instance belongs. This repository may reside locally on the source and destination, or at an agreed upon location.

The schemaLocation provided by the sender points to the version used to create the instance. The recipient may or may not have access to the source XMLSchema. If not the receiver needs to check to see what other versions the given instance can be validated against.

```
<ProcessPurchaseOrder
xmlns="http://www.openapplications.org/oagis/9"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.openapplications.org/oagis/9
http://www.openapplications.org/oagis/9.0/BODs/Developer/Pr
ocessPurchaseOrder.xsd" releaseID="9.0 9.1" versionID="10"
systemEnvironmentCode="Production" languageCode="en-US">
```

The example above shows a direct reference to the XML Schema definition for the ProcessPurchaseOrder.xsd file in the schemaLocation. In an implementation for an end user or for an application the instance must reference a URL of a repository that the XML Schema definitions are contained for the implementation. This identifies the location of the schema that can be found by both the sender and receiver and can be used for validation.

OAGIS schemas and Overlay schemas for that matter use the schemaLocation with a relative reference. Therefore it is important to maintain the file system structure in which they are provided. Below is an example of the schemaLocation used in the ProcessPurchaseOrder.xsd Schema definition:

```
<xsd:schema
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns="http://www.openapplications.org/oagis/9"
targetNamespace="http://www.openapplications.org/oagis
/9" elementFormDefault="qualified"
attributeFormDefault="unqualified">
<xsd:include
schemaLocation="../../Resources/Nouns/PurchaseOrder.xs
d"/>
```

Advantages

- Allows the author of an instance to point into the repository of the schemas to indicate the version in which the instance was generated from.

Disadvantages

- The schemaLocation attribute is not a required element, according to the W3C, and the schema parser that is being used does not have to honor the request. The schemaLocation is only a hint to the parser where the schema may be found.

The schemaLocation could be used with a relative reference in conjunction with version attribute of the root element where the content provided by the author of the instance indicates the version(s) that are compatible. The user can then determine which version(s) they support and point to the correct repository.

3.2.3 In Attributes on the Root Element Instance Usage

OAGi uses attributes on the root element to identify the OAGIS release and BOD version for the instance document. Since these are instance attributes they are not fixed by the schema and are provided by the sending application. The attributes are:

- releaseID – Indicates the OAGIS Release for the associated BOD Instance or the OAGIS Release that is the basis for extension of the Overlay.

- versionID - Indicates the version of the given BOD definition that is within the associated release. i.e., how many times this BOD definition has been revised.

The example below shows how this appears in an instance.

```
<ProcessPurchaseOrder
xmlns="http://www.openapplications.org/oagis/9"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.openapplications.org/oagis/9
../../BODs/Developer/ProcessPurchaseOrder.xsd"
releaseID="9.0" versionID="10"
systemEnvironmentCode="Production" languageCode="en-US">
```

Advantages

- Instance documents do not have to change if they remain valid with the new schema version.

- An application has programmatic indication that the schema has changed.

- Used in conjunction with the schemaLocation attribute to determine that the schemaLocation provided exists. If not it can be used to indicate the repository containing the Schema.

4.0 SUMMARY

OAGi's approach to versioning within OAGIS, an XML Schema based Canonical Business Language, meets or exceeds the general requirements of the industry. It also supports the sound design rules applied throughout OAGIS. The recognized industry best practices for versioning are applied pragmatically to maximize advantages while limiting disadvantages.

These concepts are also recognized within the software industry as best practices and facilitate versioning of software that may coincide with the version(s) of OAGIS.