Open GIS Consortium Inc.

Date: 2003-01-15

Reference number of this OpenGIS® Project Document: **OGC 03-013**

Version: 0.0.3

Category: OpenGIS® Discussion Paper

Editor: Panagiotis (Peter) A. Vretanos

Web Object Service Implementation Specification

Warning

This document is not an OGC Standard or Specification. This document presents a discussion of technology issues considered in an Interoperability Initiative of the OGC Interoperability Program. The content of this document is presented to create discussion in the geospatial information industry on this topic; the content of this document is not to be considered an adopted specification of any kind. This document does not represent the official position of the OGC nor of the OGC Technical Committee. It is subject to change without notice and may not be referred to as an OGC Standard or Specification.

Recipients of this document are invited to submit, with their comments, notification of any relevant patent rights of which they are aware and to provide supporting documentation.

Document type:	OpenGIS® Discussion Paper
Document stage:	Publicly Available
Document language:	English

File name: 03-013.doc

Contents

i. Preface

Within the OGC, the Web Feature Service (WFS) Implementation Specification has been used as the basis for defining of a number of library or data management service interfaces. These include the interface for the Stateless Catalog (now called the Web Registry Service) and the Gazetteer Specification. Each derived service modified the WFS interface and the payload that the interface operated upon slightly to satisfy its requirements. After the initial work on the Stateless Catalog, it was realized that it would be beneficial if a standard set of base types could be defined from which data management service interfaces could be derived by restriction or extension or both. This specification represents the first attempt to define such base types.

The OWS 1.2 project has a requirement to manage many different types of objects. These include styles, symbols and images. To satisfy this requirement, a repository interface is required. The intent of the Web Object Service interface is to provide a means to define this interface.

ii. Submitting organizations

The following companies submitted this specification to the OGC as a Discussion Paper:

CubeWerx Inc.
Edric Keighan
200 Rue Montcalm, Suite R-13
Hull, Quebec
Canada J8Y 3B5
ekeighan@cubewerx.com

iii. Submission contact points

All questions regarding this submission should be directed to the Architecture WG chair or to the Editor:

Panagiotis A. Vretanos
CubeWerx, Inc.
200 Rue Montcalm, Suite R-13
Hull, Quebec J8Y 3B5 CANADA
+1 416 701 1985
pvretano@cubewerx.com

Additional contributors

Darko Androsavic (Galdos System Inc.)
Craig Bruce (CubeWerx) csbruce@cubwerx.com
Shane Covington (CAST) scoving@cast.uark.edu
John Davidson johnd@imagemattersllc.com
Chris Dillard (Polexis) cdillard@polexis.com
John D. Evans (NASA) john.evans@gsfc.nasa.gov
Stephane Fellah (PCI Geomatics) Fellah@pcigeomatics.com
Steven Keens (PCI Geomatics) keens@pcigeomatics.com
Edric Keighan (CubeWerx) ekeighan@cubewerx.com
Ron Lake (Galdos Systems Inc.) rlake@galdosinc.com
Jeff Lansing (Polexis) jeff@polexis.com
Serge Margoulies (Ionic) Serge.Margoulies@ionicsoft.com
Brian May (CubeWerx) bmay@cubewerx.com
Richard Martell (Galdos Systems Inc.) rmartell@galdosinc.com
Alex Milanovic (Galados Systems Inc.)
Dimitri Monie (Ionic) dimitri.monie@ionicsoft.com
Keith Pomakis (CubeWerx) pomakis@cubewerx.com
Lou Reich (NASA) louis.i.reich@gsfc.nasa.gov
Carl Reed (Open GIS Consortium)
Bernard Snyers (Ionic) Bernard.Snyers@ionicsoft.com
Jerome Sonnet (Ionic) jerome.sonnet@ionicsoft.com
Glenn Stowe (CubeWerx) gstowe@cubwerx.com
Milan Trninic (Galdos Systems Inc.) mtrninic@galdosinc.com
John T. Vincent (Intergraph Corp.) jtvincen@intergraph.com
Arliss Whiteside (BAE Systems) Arliss.Whiteside@baesystems.com

iv. Revision history

0.0.4	Minor scubs; use IPR template- jdavidson
0.0.3	Clean-up foR ipr submission.
0.0.2	FILLED IN MOST tdb SECTIONS, ADDRESSED ROUND 1 REVIEW COMMENTS (TERM DEFINITIONS, OPERATION NAMES, ANNOTATE SCHEMAS, CLARIFY bbox, LOCKING CLARIFICATIONS, MOVED OLD clause 1 TO overview, INCLUDED IMAGE REQUIREMENTS), ADDED MORE EXAMPLES INCLUDING omf EXAMPLES
0.0.1	Initial revision.

v. Changes to the OpenGIS Abstract Specification

No further revisions to the OGC Abstract Specification are required. The revisions previously approved for Topic 12, "Service Architecture," including definitions of the terms "operation", "interface" and "service" are relevant to and sufficient for this specification. The essential operation of a web object service, as an object access and management service, is described in subclause 8.3.3 of Topic 12.

vi. Future work

- Harmonize with OMF specification.
- Include more packaging methods than just multipart MIME. By adding a parameter/attribute called **outputMIME**, a client application can specify that the WOS output be packged as a TAR/GZIP file (for example).

Foreword

Attention is drawn to the possibility that some of the elements of this standard may be the subject of patent rights. Open GIS Consortium Inc. shall not be held responsible for identifying any or all such patent rights. However, to date, no such rights have been claimed or identified.

This version of the specification cancels and replaces all previous versions.

Normative annexes

Annexes A and D are normative.

Introduction

This specification defines a set of base XML types from which object access and manegment services, such as the Web Feature Service or the Web Registry Service, may be derived. This specification further describes the behaviour of a Web Object Service, which is a unspecialized instantiation of the base types defined in this specification and represents a standard interface to an object repository.

The basic architecture is illustrated in Figure 1:

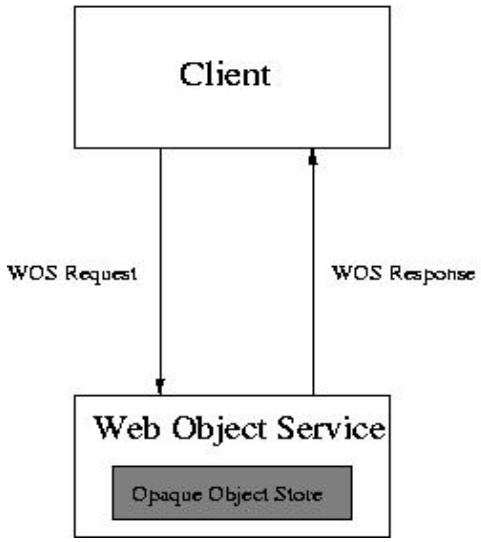

Figure 1 – Web Object Service

Web Object Service Implementation Specification

1 Scope

This specification is both an implementation specification and an Interoperability Program Report.

The specification defines a set of base XML types that define the behaviour of a Web Object Service. A Web Object Service is a generic web-based repository interface. The interface support the following operations: *GetCapabilities, DescribeObjectType, GetObjectById, GetObject, LockObject* and *Transaction.* The specification assumes that the destributed computing platform is HTTP and may define both XML (suitable for the POST method) and Keyword-Value Pair (suitable for the GET method) encodings of each operation.

A section is also included in the specification that describes several WOS implementation used in the OWS1.2 testbed to manager several object types including images, symbols, styles and metadata.

2 Conformance

Conformance with this specification shall be checked using all the relevant tests specified in Annex D (normative). The framework, concepts, and methodology for testing, and the criteria to be achieved to claim conformance are specified in ISO 19105: Geographic information — Conformance and Testing.

3 Normative references

[1] Bradner, Scott, "RFC 2119 Key words for use in RFCs to Indicate Requirement Levels," March 1997, ftp://ftp.isi.edu/in-notes/rfc2119.txt .

[2] Cox, S., Cuthbert, A., Lake, R., and Martell, R. (eds.), "OpenGIS Implementation Specification #02-009: OpenGIS® Geography Markup Language (GML) Implementation Specification, version 2.1.1", April 2002

[3] Vretanos, Panagiotis (ed.), "OpenGIS Implementation Specification #01-067: Filter Encoding Implementation Specification", May 2001

[4] Percivall, George, ed., “The OpenGIS Abstract Specification, Topic 12: OpenGIS Service Architecture”, 2002

[5] Bray, Paoli, Sperberg-McQueen, eds., "Extensible Markup Language (XML) 1.0", 2nd edition, October 2000, W3C Recommendation, http://www.w3.org/TR/2000/REC-xml.

[6] Beech, David, Maloney, Murry, Mendelson, Noah, Thompson, Harry S., “XML Schema Part 1: Structures”, May 2001, W3C Recommendation, http://www.w3c.org/TR/xmlschema-1.

[7] Bray, Hollander, Layman, eds., “Namespaces In XML”, January 1999, W3C Recommendation, http://www.w3.org/TR/2000/REC-xml-names.

[8] Clark, James, DeRose, Steve, “XML Path Language (XPATH), Version 1.0”, November 1999, W3C Recommendation, http://www.w3c.org/TR/XPath.

[9] Fielding et. al., "Hypertext Transfer Protocol – HTTP/1.1," IETF RFC 2616, June 1999, http://www.ietf.org/rfc/rfc2616.txt.

[10] Berners-Lee, T., Fielding, N., and Masinter, L., "Uniform Resource Identifiers (URI): Generic Syntax", IETF RFC 2396, http://www.ietf.org/rfc/rfc2396.txt.

[11] National Center for Supercomputing Applications, "The Common Gateway Interface," http://hoohoo.ncsa.uiuc.edu/cgi/.

[12] Freed, N. and Borenstein N., "Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies", IETF RFC 2045, November 1996, http://www.ietf.org/rfc/rfc2045.txt.

[13] Internet Assigned Numbers Authority, http://www.isi.edu/in-notes/iana/assignments/media-types/.

[14] OpenGIS Document 99-051, Catalog Interface Implementation Specification (Version 1.0)

[15] OpenGIS Document 99-049, Simple Features Specification For SQL, Revision 1.1

4 Terms and definitions

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

4.1
operation
specification of a transformation or query that an object may be called to execute [4]

4.2
interface
a named set of operations that characterize the behavior of an entity [4]

4.3
service
a distinct part of the functionality that is provided by an entity through interfaces [4]
4.4
service instance
an actual implementation of a service; service instance is synonymous with server
4.5
client
a software component that can invoke an operation from a server
4.6
request
an invocation by a client of an operation.

4.7
response
the result of an operation returned from a server to a client.

4.8
capabilities XML
service-level metadata describing the operations and content available at a service instance
4.9
spatial reference system
as defined in ISO19111

4.10
opaque
not visible, accessible or meaningful to a client application

4.11
object
an object represents an individual, identifiable item, unit, or entity, either real or abstract, with a well-defined role in the problem domain [12]

4.12
property
is a characteristic or attribute of an object

4.13
simple property
a property that does not include any sub-properties

4.13
complex property
a property that has sub-properties which may themselves be simple or complex properties

4.15
schema
the structure of an object expressed in a formal language such as XML-Schema, SQL, etc…

4.17
XML proxy
an XML representation of an object that is not easily or conveniently encoded in XML

4.18
globally unique
global uniqueness in the context of object identifiers means that now two identifiers are the same regardless of which computer they were generated on; a globally unique identifier can be created in a number of ways but is usually some combination of a few unique settings based on a specific point in time (e.g. an IP address, a MAC address, clock date/time, etc…)

5 Conventions

5.1 Normative verbs

In the clauses labeled as normative, the key words "**must**", "**must not**", "**required**", "**shall**", "**shall not**", "**should**", "**should not**", "**recommended**", "**may**", and "**optional**" in this document are to be interpreted as described in Internet RFC 2119 [1].

5.2 Abbreviated terms

CGI	Common Gateway Interface
DCP	Distributed Computing Platform
DTD	Document Type Definition
EPSG	European Petroleum Survey Group
GIS	Geographic Information System
GML	Geography Markup Language
HTTP	Hypertext Transfer Protocol
MIME	Multipurpose Internet Mail Extensions
OGC	Open GIS Consortium
OMF	OGC Messaging Framework
OWS	OGC Web Service
URL	Uniform Resource Locator
WOS	Web Object Service
WFS	Web Object Service
XML	Extensible Markup Language

5.3 Use of examples

This specification makes extensive use of XML examples. They are meant to illustrate the various aspects of a web object service discussed in this specification. While every

effort has been made to ensure that the examples are well formed and valid in many cases this goal was sacrificed for the sake of clarify. For example, many examples are formatted in a specific way to highlight a particular aspect that would render the example invalid from the perspective of an XML validation tool. Further, most examples reference fictitious servers and data.

Thus, this specification does not assert that any XML or keyword-value pair encoded example, copied from this document, will necessarily execute correctly or validate using a particular XML validation tool. Only clauses marked as *normative* should be expected to be well formed and valid XML or XML Schema documents.

6 Overview

This document serves two purposes. First, it defines a set of generic XML types from which object access and management services, such as WFS and WRS, may be derived. These generic types are defined in a schema file called *wos.xsd* and included in Annex A.

Second, this document describes an unspecialized instantiation of the types defined in *wos.xsd* to define a Web Object Service. Like the WFS and WRS, the WOS supports INSERT, UPDATE, DELETE, QUERY and DISCOVERY operations on object instances other than (but not excluding) GML [2] features. Object instances may be encoded directly inline with a WOS request message, using XML, or object instances may be referenced using other mechanisms described in this document.

6.1 Object encoding

This specification assumes that objects presented to the inteface as input are encoded using XML or that an XML proxy can be used to reference an object that is not easily or conveniently encoded in XML. For example, an HDF/EOS image is not conveniently expressable in XML. However, it is a simple matter to reference the parts of an HDF/EOS image packaged in a multipart-MIME document or stored at some remote web-accessible location. In addition to supporting references to entire objects, this specification also supports references to parts of an object by supporting references to property values of object instances that can be encoded in XML.

6.2 Object instance identification

This document assumes that every object instance that a particular WOS implementation can operate upon is uniquely identifiable. That is to say, when a WOS implementation reports an object identifier for an object instance, that object identifier is unique and can be used to repeatedly reference the same object instance (assuming it has not been deleted). An object identifier can be used wherever an object instance reference is required.

In addition, this specification mandates that the object identifer must be globally unique. A single globally-unique string would be more convenient to use in multiple contexts.

This string could be used as if it were fully opaque in many contexts, but it would be more useful if it were actually a URL or URN which could be used to directly access the object instance it identifies in the native format of the object instance.

The global object identifier must satisfy the following requirements:

1. The object identifier must be globally unique.

2. The object identifier must be a valid URL. Using a URL is helpful for applications that need only simple access to the raw object instances since no interface details need to be known. This mode of access/identification is also helpful for integration with high-level XML technologies such as RDF or XSLT, and even for debugging purposes.

3. The object identifer may be used as an opaque string or as a valid URL as the context in which it is used dictates.

4. The actual format of the URL string is entirely at the discretion of the web object service.

6.3 Operations

To support insert, update delete, query and discovery processing, the following operations are described in this specification:

GetCapabilities

A web object service must be able to describe its capabilities. Specifically, it must indicate which object types it can service and what operations are supported on each object instance.

DescribeObjectType

A web object service may be able, upon request, to describe the structure of any object type it can service. That is to say that it can generate an XML document to describe the structure of the object type. This XML document may be XML Schema but other document types are also possible such as RDF, for example.

GetObjectById

A web object service must be able to retrieve a single raw object instance using the global identifier generated by the service. The term raw is used to denote that the returned object instance must not be packged in any way inside some containment element such as the wfs:FeatureCollection used by the web feature service.

GetObject

A web object service may be able to support more complicated query processing that involves the use of a sophisticated predicate language such as the filer encoding as defined in the Filter Encoding Implementation Specification [3].

Transaction

A web object service may be able to service transaction requests. A transaction request is composed of or more operations that insert, update or delete object instances or parts of object instances from a persistent object repository.

LockObject

A web object service may be able to process a lock request on one or more instances of an object for the duration of a transaction. This ensures that serializable transactions may be supported.

Figure 2 is a simplified protocol diagram illustrating the messages that might be passed back and forth between a client application and a web object service in order to process a typical transaction request. The elements referenced in the diagram are defined in this document.

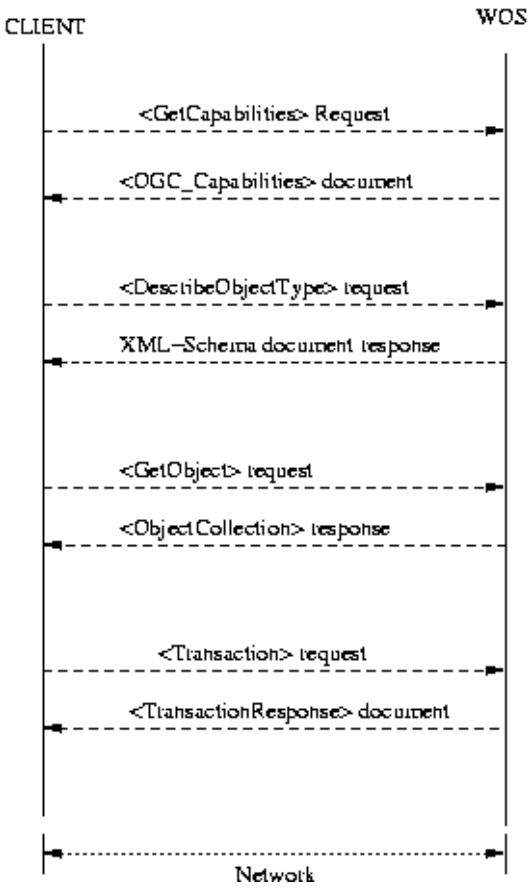

Figure 2 - Protocol diagram

Based on these operations, a variety of classes of web object servers may be implemented. The minimum requirement is a web object service that supports the **GetCapabilities**, and **GetObjectById** operations. More advanced servers many implement any or all of the other operations. In all cases, a web object service must advertise which operations it supports in its capabilities document.

7 Requirements

The following requirements, derived during OWS1.2 testbed, were considered in producing this specification:

- from the Image Handling group
    - **"Image Requirements DIPR"** (02-053r1)
        - subclauses 5.2.4 through 5.2.7
        - subclauses 5.3.2 through 5.3.5
    - **"Image Handling Requirements"** slide presentation
    - **"Operation Changes for ImageHandling**" slide presentation
- from the Style and Symbol management group
    - The ability to Fetch, Insert, Update and Delete style and symbol objects (encoded in XML or acting as XML proxies for other data types).
    - The ability to reference an object using only a unique identifier. No complex predicates are required.
    - Must be able to perform these operations on multiple objects at one time for efficiency.
    - The base HTTP method that must be supported in GET.
- additional requirements
    - The interface must be able to manipulate entire objects at once or parts of object (if that is possible). This way, the interface can manipulate a binary sound file, for example, (which is not generally divisible except with specialized software) or it can update a specific property of an object that is expressed using XML.
    - The interface should be general enough so that useful extensions of this interface can be derived by extension or restriction. Examples of

extensions to this interface would include the Web Feature Service interface and the Web Registry Service interface.

8 Basic service elements

8.1 Introduction

This section describes aspects of Web Object Service behavior (more generally, of OGC Web Service behavior) that are independent of particular operations or are common to several operations or interfaces.

8.2 Version numbering and negotiation

8.2.1 Version number form

The published specification version number contains three positive integers, separated by decimal points, in the form "x.y.z". The numbers "y" and "z" will never exceed 99. Each OWS specification is numbered independently.

8.2.2 Version changes

A particular specification's version number **shall** be changed with each revision. The number **shall** increase monotonically and **shall** comprise no more than three integers separated by decimal points, with the first integer being the most significant. There may be gaps in the numerical sequence. Some numbers may denote experimental or interim versions. Service instances and their clients need not support all defined versions, but **must** obey the negotiation rules below.

8.2.3 Appearance in requests and in service metadata

The version number appears in at least two places: in the Capabilities XML describing a service, and in the parameter list of client requests to that service. The version number used in a client's request of a particular service instance **must** be equal to a version number which that instance has declared it supports (except during negotiation as described below). A service instance may support several versions, whose values clients may discover according to the negotiation rules.

8.2.4 Version number negotiation

An OWS Client may negotiate with a Service Instance to determine a mutually agreeable specification version. Negotiation is performed using the **GetCapabilities** operation, described in clause 10, according to the following rules.

All Capabilities XML must include a protocol version number. In response to a GetCapabilities request containing a version number, an OGC Web Service **must** either respond with output that conforms to that version of the specification, **or** negotiate a mutually agreeable version if the requested version is not implemented on the server. If no version number is specified in the request, the server **must** respond with the highest version it understands and label the response accordingly.

Version number negotiation occurs as follows:

1. If the server implements the requested version number, the server **must** send that version.

2. If the client request is for an unknown version greater than the lowest version that the server understands, the server **must** send the highest version less than the requested version.

3. If the client request is for a version lower than any of those known to the server, then the server **must** send the lowest version it knows.

4. If the client does not understand the new version number sent by the server, it **may** either cease communicating with the server **or** send a new request with a new version number that the client does understand but which is less than that sent by the server (if the server had responded with a lower version).

5. If the server had responded with a higher version (because the request was for a version lower than any known to the server), and the client does not understand the proposed higher version, then the client **may** send a new request with a version number higher than that sent by the server.

The process is repeated until a mutually understood version is reached, or until the client determines that it will not or cannot communicate with that particular server.

Example 1: Server understands versions 1, 2, 4, 5 and 8. Client understands versions 1, 3, 4, 6, and 7. Client requests version 7. Server responds with version 5. Client requests version 4. Server responds with version 4, which the client understands, and the negotiation ends successfully.

Example 2: Server understands versions 4, 5 and 8. Client understands version 3. Client requests version 3. Server responds with version 4. Client does not understand that version or any higher version, so negotiation fails and client ceases communication with that server.

8.3 General HTTP request rules

8.3.1 Introduction

At present, the only distributed computing platform (DCP) explicitly supported by OGC Web Services is the World Wide Web itself, or more specifically Internet hosts implementing the Hypertext Transfer Protocol (HTTP)[9]. Thus the Online Resource of each operation supported by a service instance is located by an HTTP Uniform Resource Locator (URL). The URL may be different for each operation, or the same, at the discretion of the service provider. Each URL **must** conform to the description in [9] but is otherwise implementation-dependent; only the parameters comprising the service request itself are mandated by the OGC Web Services specifications.

HTTP supports two request methods: GET and POST. One or both of these methods may be defined for a particular OGC Web Service type and offered by a service instance, and the use of the Online Resource URL differs in each case.

In either case a WOS must be prepared to handle multipart documents as described below in order to be able to support request messages that include the encoded WOS operation and the data (possibly binary) that the operation references.

8.3.2 HTTP GET

An Online Resource URL intended for HTTP GET requests is in fact only a URL prefix to which additional parameters must be appended in order to construct a valid Operation request. A URL prefix is defined as an opaque string including the protocol, hostname, optional port number, path, a question mark '?', and, **optionally**, one or more server-specific parameters ending in an ampersand '&'. The prefix uniquely identifies the particular service instance. A client appends the necessary request parameters as name/value pairs in the form "name=value&". The resulting URL **must** be valid according to the HTTP Common Gateway Interface (CGI) standard [7], which mandates the presence of '?' before the sequence of query parameters and the '&' between each parameter. As with all CGI applications, the query URL is encoded [10] to protect special characters.

The URL prefix **must** end in either a '?' (in the absence of additional server-specific parameters) or a '&'. In practice, however, Clients **should** be prepared to add a necessary trailing '?' or '&' before appending the Operation parameters defined in this specification in order to construct a valid request URL.

Table 1 summarizes the components of an operation request URL.

Table 1 – A general OGC Web Service Request

URL COMPONENT	DESCRIPTION
http://host[:port]/path?{name[=value]&}	URL prefix of service operation. [ ] denotes 0 or 1 occurrence of an optional part; {} denotes 0 or more occurrences. The prefix is entirely at the discretion of the service provider.
name=value&	One or more standard request parameter name/value pairs defined by an OGC Web Service. The actual list of required and optional parameters is mandated for each operation by the appropriate OWS specification.

8.3.2.1 GET / Form POST request semantics

8.3.2.1.1 Introduction

This purpose of this section is to review available methods for invoking WOS operations that are encoded using keyword-value pairs. Three methods are available for transmitting keyword-value pairs from a client to a web service. They are the standard **GET** method described above, the *application/x-www-form-urlencoded* **POST** method and the *multipart/form-data* **POST** method.

8.3.2.1.2 GET request method

As described in subclause 8.3, the GET request appends the input information directly onto the URL. The keyword-value pairs are in the form of "name=value" and the pairs are separated by the "&" character. This is the familiar URL one sees in their browser. For example:

```
http://www.someplace.com/wfs/wfs.cgi?WFSVER=0.0.12&REQUEST=TRANSACTION&OPERATION=INSE
RT&PROP1=<some value>
```

With this method, a reference to a value must be an external reference (as opposed to the multipart document case where the request and the data re sent in a single request and the value references many be local to the current document).

8.3.2.1.3 KVP POST request methods

An alternate way of transmitting the keyword-value pairs is to use the **HTTP POST** method. The keyword value pairs are encoded into a document and that document is posted to a web server. This is the method, for example, in which HTML forms are transmitted from a client to a web server. Two common methods exist for encoding keyword-value pairs into a document.

8.3.2.1.3.1 "application/x-www-form-urlencoded" Encoding

This form of encoding sends the keyword-value pairs in a data body with the MIME-type "application/x-www-form-urlencoded". With this encoding, spaces are translated to "+", non-alpha characters are converted to the format "%XX" where XX are the hex digits for the ASCII representation of the translated character, and line breaks are converted to '%0D%0A". There is no practical limit to the number of keyword value pairs that can be transmitted in this manner. This is the default method for post HTML forms. An example of an urlencoded document is:

```
VERSION=0.0.1&REQUEST=TRANSACTION&OPERATION=INSERT&PROP1=%Absome%CDvalue%AB
```

8.3.2.1.3.2 "multipart/form-data" Encoding

"multipart/form-data" encoding, encodes each keyword-value pair as a subsection of the posted document with its own mini content header. For example, the keyword-value pairs SERVICE=WOS, VERSION=0.0.1, REQUEST=TRANSACTION, OPERATION=INSERT and PROR1=<some long binary value> would be encoded by the client as:

```
-----------------------WOS0123456
Content-Disposition: form-data; name="SERVICE"
Content-Type: text/plain; Content-Length=3


WOS

-----------------------WOS0123456
Content-Disposition: form-data; name="VERSION"
Content-Type: text/plain; Content-Length=5
```

```
0.0.1
------------------------WOS0123456
Content-Disposition: form-data; name="REQUEST"
Content-Type: text/plain; Content-Length=11


TRANSACTION
------------------------WOS0123456
Content-Disposition: form-data; name="OPERATION"
Content-Type: text/plain; Content-Length=16


INSERT
------------------------WOS0123456
Content-Disposition: form-data; name="PROP1"
Content-Type: xxx/binary; Content-Length=23256


… a 23256 byte long binary stream …
------------------------WOS0123456–
```

The lines with the dashes are separators generated by the client. Each client typically uses its own standard separator - what is shown here is simply an example. The separators can be any string but must begin with two dashes "—". For each part, a content header is provided that conveys all the expected information and follows all the normal rules of content headers – including the two new lines before starting the data stream. Using this encoding method, binary data can be transmitted inline with the rest of a posted document.

8.3.3 HTTP POST

An Online Resource URL intended for **HTTP POST** requests is a complete and valid URL to which clients transmit encoded requests in the body of the POST document. A WOS **must not** require additional parameters to be appended to the URL in order to construct a valid target for the operation request.

8.3.3.1 Request types

When posting XML encoded requests to a WOS, the request may be purely XML that may include references to external data. For example, the request could reference image data by includeing an *xlink* to an image somewhere on the web.

Another possibility is that the request, and the data the request references are posted as a single document. Multipart MIME documents, allow multiple (possibly different) payloads to be combined into a single document and transmitted using HTTP.

8.3.3.2 multipart MIME documents

In the context of a web wobject service, multipart MIME documents allow a client to post a WOS operation to a server that includes the XML encoded request and one or more additional parts that are the data that the request references. In this manner, both text and multimedia data can be manipulated by a web object service.

The first part of a multipart MIME document posted to a web object service **must** be the WOS operation. The other parts of the MIME document may include the data that is referenced in the request.

The following example illustrates how a client may package a WOS transaction and multimedia data into a single request:

```
------------------------WOS0123456
Content-ID: TX01
Content-Type: text/xml; Content-Length=310


<Transaction>
   <Insert>
      <NewsClip>
         <Reporter>Joe Cool</Reported>
         <Date>30-JUL-2002</Date>
         <Place>Toronto, Canada</Place>
         <Image xlink:href="IMAGE1" xlink:type="simple" />
         <AudioTrack xlink:href="AUDIOCLIP1" xlink:type="simple" />
      </NewsClip>
   </Insert>
</Transaction>
------------------------WOS0123456
Content-ID: AUDIOCLIP1
Content-Type: audio/mp3; Content-Length=565768


… 565768 bytes of MP3 data …
------------------------WOS0123456
Coontent-ID: IMAGE1
Content-Type: image/jpeg; Content-Length=1107465


… 1107465 bytes on jpeg image …
------------------------WOS0123456
```

In this example, the WOS operation is a transaction inserting a news clip into a repository. The news clip contains a JPEG image and an MP3 audio track. These multimedia data types are not encoded directly into the request but are, instead, referenced by the request. The xlink, in each case, references the appropriate *Content-ID* tag in the MIME headers so that the correct data is associated with the correct property.

Depending on the object types that a web object service plans to deal with a web object service **may** need to be prepared to deal with multipart MIME documents that includes a WOS request message and one or more extra parts that contain the data that the request references.

It should be noted that such request messages may also include references to external data, not part of the multipart MIME document, using xlinks as described in this specification.

8.4 General HTTP response rules

Upon receiving a valid request, the service **must** send a response corresponding exactly to the request as detailed in the appropriate specification. Only in the case of Version Negotiation (described above) may the server offer a differing result.

Upon receiving an invalid request, the service **must** issue a Service Exception as described in subclause 9.8.

NOTE: As a practical matter, in the WWW environment a client should be prepared to receive either a valid result, or nothing, or any other result. This is because the client may itself have formed a non-conforming request that inadvertently triggered a reply by something other than an OGC Web Service, because the Service itself may be non-conforming.

Response objects **must** be accompanied by the appropriate Multipurpose Internet Mail Extensions (MIME) [9] type for that object.

Response objects **should** be accompanied by other HTTP entity headers as appropriate and to the extent possible. In particular, the Expires and Last-Modified headers provide important information for caching; Content-Length may be used by clients to know when data transmission is complete and to efficiently allocate space for results, and Content-Encoding or Content-Transfer-Encoding may be necessary for proper interpretation of the results.

8.5 Request encoding

This document defines two methods of encoding request messages. The first method uses keyword-value pairs to encode the various parameters of a request. The second method uses XML as the encoding language. An example of a keyword value pair is:

```
"REQUEST=GetCapabilities"
```

where "REQUEST" is the keyword and "GetCapabilities" is the value. In both cases, the response to a request or exception reporting must be identical.

To conform to this specification an implementation **must** support one of these two encoding methods but **may** support both.

Table 2 correlates WOS operations and their encoding semantics as defined in this specification.

Table 2 – Operation Request Encoding

OPERATION	REQUEST ENCODING
GetCapabilities	XML & KVP[1]
DescribeObjectType	XML & KVP
GetObjectById	KVP
GetObject / GetObjectWithLock	XML & KVP

[1] KVP = Keyword-Value Pair

LockObject	XML & KVP
Transaction	XML & KVP

8.5.1 XML encoding

XML encoding means that the request message is encoded according the rules described in the Extensible Markup Language (XML) V1.0 [5] specification and the any additional rules defined in this specification.

As described above, an XML encoded request message may include data encoded directly into the message but may also reference data included with the request (as a multipart MIME document) and/or reference external data not included with the request but accessible on the web.

A WOS **must** validate such links at the time of the request, If the link is an external reference, a WOS may optionally resolve the link to pull the referenced data into its local repository. However, if a link is validated but not resolved, there is no guarantee that the link will be valid at some later time when a client attempts to access the referenced data. In such an instance, a WOS should raise an exception as describe in subclause 9.8.

8.5.2 Keyword-value pair encoding

8.5.2.1 Introduction

This section describes how to encode WOS request messages using keyword-value pairs. This means that parameters consist of name-value pairs in the form of "name=value" and the pairs are separated by the "&" character. This form of encoding is also known as URL-Encoding.

In general URL-encoding requires that certain characters, such as '<' and '>' be escaped [10] when they are not used in their intended manner. In this document, however, such characters may not escaped for the sake of clarity.

8.5.2.2 Request parameter rules

8.5.2.2.1 Parameter ordering and case

Parameter names shall not be case sensitive, but parameter values shall be case sensitive. In this document, parameter names are typically shown in uppercase for typographical clarity, not as a requirement.

Parameters in a request may be specified in any order.

An OGC Web Service must be prepared to encounter parameters that are not part of this specification. In terms of producing results per this specification, an OGC Web Service shall ignore such parameters.

8.5.2.2.2 Parameter lists

Parameters consisting of lists shall use the comma (",") as the delimiter between items in the list. In addition, multiple lists can be specified as the value of a parameter by enclosing each list in parentheses; "(", ")".

Example 1:

An example of a list of items.

```
parameter=item1,item2,item2
```

Example 2:

An example of multiple lists of items assigned to a single parameter.

```
parameter=(item11,item12,item13)(item21,item22,item23)
```

Parentheses may also be used to delimit multiple local filters when more that one object type is specified for the **OBJECTNAME** parameter. The following URL fragment shows how this may be done:

```
objectname=FEAT1,FEAT2&filter=(<Filter>… FEAT1 filter…</Filter>)(<Filter>… FEAT2
filter…</Filter>)
```

8.5.2.3 Common request parameters

8.5.2.3.1 Version parameter

The **VERSION** parameter specifies the protocol version number. The format of the version number, and version negotiation, are described in subclause 8.2.

8.5.2.3.2 Request parameter

The **REQUEST** parameter indicates which interface operation is being invoked. The value operation_name must be one of those offered by the OGC Web Service Instance.

8.5.2.3.3 Bounding box

The bounding box parameter, BBOX and the BBOXPROPERTY parameter, are included in this specification for convenience. They are a shorthand representation of the very common bounding box filter which would be expressed in much longer form using any of the predicate languages supported.

Since a general object may have more than one geometric property, the BBOXPROPERTY parameter is included to allow the client to indicate the property name of the geometric property to which the BBOX filter should be applied. If the object

being tested only has a single geometric property then the BBOXPROPERTY parameter does not need to be specified and the web object service must determine what the name of the geometric property is and act according. If the property name specified for the BBOXPROPERTY parameter is not a geometric property, then an exception should be raised.

The Bounding Box (BBOX) is a set of four comma-separated decimal, scientific notation, or integer values (if integers are provided where floating point is needed, the decimal point is assumed at the end of the number). These values specify the minimum X, minimum Y, maximum X, and maximum Y ranges, in that order, expressed in units of the SRS of the object type(s) begin queried. The four bounding box values indicate the outside edges of a rectangle, as in Figure 5; minimum X is the left edge, maximum X the right, minimum Y the bottom, and maximum Y the top.

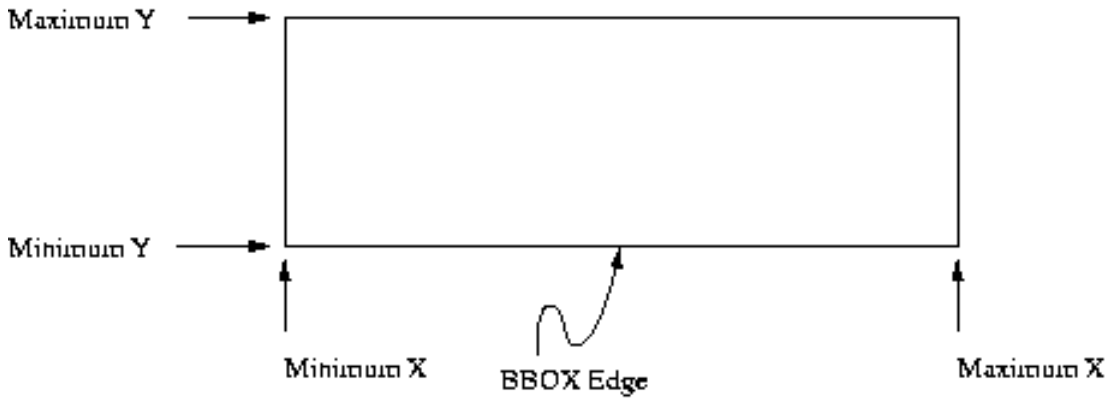

Figure 5 - Bounding Box representation

A Bounding Box should not have zero area.

If a request contains an invalid Bounding Box (e.g., one whose minimum X is greater than or equal to the maximum X, or whose minimum Y is greater than or equal to the maximum Y) the server **must** throw an exception.

If a request message contains a Bounding Box whose area does not overlap at all with the LatLongBoundingBox(s) advertised in the Capabilities XML for the requested geodata object, the server should return empty content (e.g. null object set) for that element. Any elements that are partly or entirely contained in the Bounding Box should be returned in the appropriate format.

The SRS of the bounding box **must** be the same as the SRS of the geometric property to which it is being applied. The SRS of must be advertised by a WOS in the capabilities document. If more than one object type is specified in a request.

If the Bounding Box values are not defined for the given SRS (e.g., latitudes greater than 90 degrees in EPSG:4326), the server should return empty content for areas outside the valid range of the SRS.

In the particular case of longitude, the following behavior applies regarding the anti-meridian at 180 degrees of longitude. There is a legitimate desire for maps that span the

anti-meridian (for example, a map centered on the Pacific Ocean). If Xmin is the west-most longitude in degrees and Xmax is the east-most, then the following constraint applies:

```
-180 <= Xmin < Xmax < 540
```

If more than one object name is specified in the request message then both parameters should accept lists of values as specified in subclause 8.5.2.2.2. There must be a 1:1 correspondence between the list of names assigned to the BBOXPROPERTY parameter and the lists of four coordinates assigned to the BBOX parameter.

Example

Xmin,Xmax values and the corresponding scope of the bounding box:

```
-180,180 = Earth centered at Greenwich
0,360 = Earth with Greenwich at left edge
120,250 = Pacific Ocean
```

8.5.2.3.4 Vendor-specific parameters

Keyword-value pair encoded request messages may allow for optional vendor-specific parameters (VSPs) that will enhance the results of a request. Typically, these are used for private testing of non-standard functionality prior to possible standardization. A generic client is not required or expected to make use of these VSPs.

An OGC Web Service **must** produce a valid result even if VSPs are missing or malformed (i.e., the Service shall supply a default value), or if VSPs are supplied that are not known to the Service (i.e., the Service shall ignore unknown request parameters).

An OGC Web Service may choose not to advertise some or all of its VSPs. If VSPs are included in the Capabilities XML, the VendorSpecificCapabilities element must be redefined accordingly. Additional schema documents may be imported containing the redefinition of the element.

Clients may read the vendor specific definition from the capabilities schemas and formulate requests using any VSPs advertised therein.

Vendors should choose vendor-specific parameter names with care to avoid clashes with standard parameters.

8.5.2.4 Common parameters

The following table describes parameters common to all WOS operations. Subsequent tables may redefine some of the facets of one or more of the parameters in this table. For example, the SERVICE parameter optional for most operations but is mandatory for the GetCapabilities operation.

Table 3 - Common parameters for WOS operations

URL Component	O/M[2]	DEFAULT	Description
http:/server_address/path/script	M		URL prefix of web feature service
VERSION	M	0.0.1	Request version.
SERVICE	M	WOS	Service type.
REQUEST	M		Name of WOS operation.
Additional parameters	O		As described in this document.
Vendor-specific parameters	O		Optional vendor specific parameters.

8.6

The **SERVICE** parameter specifies which of the available service types at a particular service instance is being invoked. The value WOS is used to indicate that the Web Object Service should be invoked.

The **VERSION** parameter specifies the protocol version number and allows for negotiation as described in subclause 8.2.4.

The parameter **REQUEST** must also be included and it indicates which of the web object service operations to invoke. The possible values of the **REQUEST** parameters are: *DescribeObjectType*, *LockObject*, *Transaction*, *GetObject*, *GetObjectWithLock* or *GetCapabilities*.

Additional **GET** parameters, as described in this section, shall be expressed as name-value pairs. Parameter names shall not be case sensitive. Parameter values shall be case sensitive. Parameters in a request may be specified in any order.

A WOS must be prepared to encounter parameters that are not part of the specification. These are known as vendor-specific parameters. Vendor-specific parameters allow vendors to specify additional parameters which will enhance the results of requests. A WOS must produce valid results even if the vendor-specific parameters are missing or malformed. A WOS may declare vendor-specific parameters within its capabilities XML. A WOS may choose to advertise some or all of its vendor specific parameters. Clients may read the capabilities schema and formulate requests using any vendor-specific parameters advertised therein.

[2] O = Optional, M=Mandatory

8.6.1.1 Response message

Regardless of whether a request message is encoded using keyword-value pairs or XML, the response messages in both cases **must** be identical.

8.6.1.2 Exceptions

Regardless of whether a request message is encoded using keyword-value pairs or XML, the format of the exception messages generated in response to an exception condition must be the same in both cases. Refer to subclause 9.8 for a detailed discussion of exception reporting.

8.7 Namespaces

Namespaces (17) are used to discriminate XML vocabularies from one another. For the WOS there are two normative namespace definitions, namely:

- (http://www.opengis.net/WOS) - for the WOS interface vocabulary
- (http://www.opengis.net/ogc) - for the OGC Filter vocabulary

In addition, a web object service will need to deal with one or namespaces in which the objects that it is manipulating exist. In this document the example namespace, http://www.someserver.com/myns, is used to represent the namespace in which the examples object exist.

9 Common elements

9.1 Property names

A web object service may generate any name that is meaningful to it as a property name. However, since the state of an object instance must be expressed in XML, the property names used by a web object service must also be valid element and attribute names as described in the Extensible Markup Language (XML) 1.0 [5] specification. In addition, property names may be namespace qualified as described in Namespaces in XML [7] . The following definitions are taken from sections 2 & 3 or that document:

```
[4]  NCName ::=  (Letter | '_') (NCNameChar)*
/*  An XML Name, minus the ":" */
[5]  NCNameChar ::=  Letter | Digit | '.' | '-' | '_' |
      CombiningChar | Extender
[6]  QName ::=  (Prefix ':')? LocalPart
[7]  Prefix ::=  NCName
[8]  LocalPart ::=  NCName
```

The definitions of the components Letter, Digit, CombiningChar and Extender are defined in annex B of [5].

Example

Examples of valid property names are:

Age, Temperature, _KHz, myns:INWATERA_1M.WKB_GEOM

Examples of invalid property names are:

+Domain, 123_SomeName

9.2 Referencing object properties

9.2.1 Introduction

Objects may have simple or complex properties. A simple property is a property to which a scalar value may be directly assigned. Simple properties do not have any structure associated with them. Complex properties, are properties of an object that may include nested sub-properties which themselves may be simple or complex. A problem thus arises about how such properties (simple or complex) should be referenced in the various places where property references are required (e.g. query and filter expressions). A WOS **must** use XPath [8] expressions, as defined in this document, for referencing the properties and sub-properties of an object encoded as XML elements or attributes.

9.2.2 XPath expressions

The XML Path Language [8] specification is a language for addressing parts of a XML document or in the case of this specification for referencing object properties referred to by means of XML elements or attributes.

This specification does not require a WOS implementation to support the full XPath language. In order to keep the implementation entry cost as low as possible, this specification mandates that a WOS implementation **must** support the following subset of the XPath language:

1. A WOS implementation **must** support *abbreviated relative location* paths.

2. Relative location paths are composed of one or more *steps* separated by the path separator '/'.

3. The first step of a relative location path **may** correspond to the root element of the object property being referenced **or** to the root element of the object type with the next step corresponding to the root element of the object property being referenced.

4. Each subsequent step in the path **must** be composed of the abbreviated form of the *child::* axis specifier and the name of the object property encoded as the principal node type of *element*. The abbreviated form of the *child::* axis specifier is to simply omit the specifier from the location step.

5. Each step in the path may optionally contain a predicate composed of the predicate delimiters '[' and ']' and a number indicating which child of the context

node is to be selected. This allows feature properties that may be repeated to be specifically referenced.

6. The final step in a path may optionally be composed of the abbreviated form of the *attribute::* axis specifier, '@', and the name of an object property encoded as the principal node type of *attribute*.

Example

To practically illustrate the use of XPath expressions for referencing simple and complex properties of an object (encoded as elements or attributes), consider the fictitious GML feature Person defined by the following XML Schema document:

```
<?xml version="1.0" ?>
<schema
   targetNamespace="http://www.cubewerx.com/myns"
   xmlns:myns="http://www.cubewerx.com/myns"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   xmlns="http://www.w3.org/2001/XMLSchema"
   elementFormDefault="qualified"
   version="1.0">

   <element name="Person" type="myns:PersonType" />
   <complexType name="PersonType">
      <complexContent>
         <sequence>
            <element name="LastName" nillable="true">
               <simpleType>
                  <restriction base="string">
                     <maxLength value="30"/>
                  </restriction>
               </simpleType>
            </element>
            <element name="FirstName" nillable="true">
               <simpleType>
                  <restriction base="string">
                     <maxLength value="10"/>
                  </restriction>
               </simpleType>
            </element>
            <element name="Age" type="integer" nillable="true"/>
            <element name="Sex" type="string"/>
            <element name="Spouse">
               <complexType>
                  <attribute name="sin" type="xsd:anyURI" use="required" />
               </complexType>
            </element>
            <element name="Location"
                     type="gml:PointPropertyType"
                     nillable="true"/>
            <element name="Address" type="myns:AddressType" nillable="true"/>
         </sequence>
         <attribute name="sin" type="xsd:anyURI" use="required"/>
      </complexContent>
   </complexType>

   <complexType name="AddressType">
      <sequence>
         <element name="StreetName" nillable="true">
            <simpleType>
               <restriction base="string">
                  <maxLength value="30"/>
               </restriction>
            </simpleType>
         </element>
         <element name="StreetNumber" nillable="true">
            <simpleType>
```

```
                  <restriction base="string">
                     <maxLength value="10"/>
                  </restriction>
               </simpleType>
            </element>
            <element name="City" nillable="true">
               <simpleType>
                  <restriction base="string">
                     <maxLength value="30"/>
                  </restriction>
               </simpleType>
            </element>
            <element name="Province" nillable="true">
               <simpleType>
                  <restriction base="string">
                     <maxLength value="30"/>
                  </restriction>
               </simpleType>
            </element>
            <element name="PostalCode" nillable="true">
               <simpleType>
                  <restriction base="string">
                     <maxLength value="15"/>
                  </restriction>
               </simpleType>
            </element>
            <element name="Country" nillable="true">
               <simpleType>
                  <restriction base="string">
                     <maxLength value="30"/>
                  </restriction>
               </simpleType>
            </element>
         </sequence>
      </complexType>
   </schema>
```

Note that the property Address is a complex property of type AddressType. An example instance of the feature Person might be:

```
<?xml version="1.0" ?>
<myns:Person
   sin="111222333"
   xmlns:myns="http://www.opengis.net/myns"
   xmlns:xlink="http://www.w3.org/1999/xlink"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://www.opengis.net/myns Person.xsd">
   <myns:LastName>Smith</myns:LastName>
   <myns:FirstName>Fred</myns:FirstName>
   <myns:Age>35</myns:Age>
   <myns:Sex>Male</myns:Sex>
   <myns:Spouse sin="444555666" />
   <myns:Address>
      <myns:StreetName>Main St.</myns:StreetName>
      <myns:StreetNumber>5</myns:StreetNumber>
      <myns:City>SomeCity</myns:City>
      <myns:Province>SomeProvince</myns:Province>
      <myns:PostalCode>X1X 1X1</myns:PostalCode>
      <myns:Country>Canada</myns:Country>
   </myns:Address>
</myns:Person>
```

Using XPath [10] expressions, each property of a Person feature can be referenced. Table 1 shows the XPath expressions that may be used to reference all the properties of the Person feature as well as the corresponding value of each property.

Table 3: XPath expressions and property values for Person example

XPATH EXPRESSION	ALTERNATE XPATH EXPRESSION	PROPERTY VALUE
LastName	Person/LastName	Smith
FirstName	Person/FirstName	Fred
Age	Person/Age	35
Sex	Person/Sex	Male
Source/@sin	Person/Source/@sin	444555666
Address	Person/Address	XML Fragment that represents the entire address.
Address/StreeNumber	Person/Address/StreetNumber	5
Address/StreetName	Person/Address/StreetName	Main St.
Address/City	Person/Address/City	SomeCity
Address/Province	Person/Address/Province	SomeProvince
Address/PostalCode	Person/Address/Postal_Code	X1X 1X1
Address/Country	Person/Address/Country	Canada
Person/@sin	Person/@sin	111222333

Notice that each relative location paths may begin with the root element name of the property being referenced or with the root element name of the object, Person. Each step of the path is composed of the abbreviated child:: axis specifier (i.e. the axis specifier child:: is omitted) and the name of the specified property which is of node type element.

In addition, the sin attribute on the <Person> and <Spouse> elements is referenced using the following XPath [10] expressions:

```
Person/@sin
Person/Spouse/@sin
```

In these cases the final step of the path contains the abbreviated axis specifier attribute:: (i.e. @) and the node type is attribute (i.e. sin in this case).

9.3 Object value references

A web object service must be able to accept, as input, references to objects or parts of objects. For example, consider the case where the value of an object instance or property of an object instance is a multimedia binary type such as an audio clip. The binary data may be directly encoded into a WOS operation, using base64 or hexadecimal encoding, or the value may be passed to the web object service by reference. A reference to an object instance or to a property value must include the following information:

1. A URN indicating the location of the data.

2. A MIME [9] type indicating the data model or format of the information..

The web object service schema defines the element <ObjectRef> for referencing object instances within WOS requests.

9.4 Property value references

In order to reference property values, the following attribute group, defined in the *wos.xsd* schema document, should be included on each object property definition:

```
<xsd:attributeGroup name="objRef">
   <xsd:annotation>
      <xsd:documentation>
      </xsd:documentation>
   </xsd:annotation>
   <xsd:attribute name="resolveURL"
                  type="xsd:boolean"
                  use="optional"
                  default="true" />
   <xsd:attribute name="mimeType"
                  type="xsd:string"
                  use="optional"
                  default="text/xml"/>
   <xsd:attribute name="href"
                  type="xsd:anyURI"
                  use="required" />
</xsd:attributeGroup>
```

The **resolveURL** attribute is a Boolean used to indicate how the server should handle the link. A value of **true** indicates that the server should resolve the link and store the referenced data in the repository. A value of **false** indicates that the server should validate the link but it does not have to actually retrieve the referenced resource and store it in the repository. Of course, this means that a web feature service cannot guarentee the the resource will be available when a client tries to resolve the resource at some future time.

The **mimeType** attribute is used to indicate the MIME type of the resource being pointed to.

The **href** attribute is a URI that points to the resource that is the value of the property.

9.5 Vendor specific operations

It is clear that an open interface can only support a certain common set of capabilities. The <**Native**> element is intended to allow access to vendor specific capabilities of any particular web object server.

The <**Native**> element is defined by the following XML Schema fragment:

```
<xsd:element name="Native" type="wfs:NativeType"/>
<xsd:complexType name="NativeType">
   <xsd:any />
   <xsd:attribute name="vendorId" type="xsd:string"
     use="required"/>
   <xsd:attribute name="safeToIgnore" type="xsd:boolean"
     use="required"/>
</xsd:complexType>
```

The <**Native**> element simply contains the vendor specific command or operation.

The **vendorId** attribute is used to identify the vendor that recognizes the command or operation enclosed by the <**Native**> element. The attribute is provided as a means of allowing a web object service to determine if it can deal with the command or not.

The **safeToIgnore** attribute is used to guide the actions of a web object service when the <**Native**> command or operation is not recognized. The **safeToIgnore** attribute has two possible values **True** or **False**. The values have the following meanings:

safeToIgnore=False

A value of **False** indicates that the <**Native**> element cannot be ignored and the operation that the element is associated with must fail if the web object service cannot deal with it.

safeToIgnore=True

A value of **True** indicates that the <**Native**> element can be safely ignored.

Example

This example illustrates the use of the <Native> element to enable a special feature of a SQL-based relational database. In this instance the element indicates that this is an Oracle® command and that the command can be safely ignored.

```
<Native vendorId="Oracle" safeToIgnore="True">
   ALTER SESSION ENABLE PARALLEL DML
</Native>
```

9.6 Query constraints

A query constraint or filter is used to define a set of object instances that are to be operated upon. The operating set can be comprised of one or more enumerated object instances or a set of object instances

defined by specifying spatial and non-spatial constraints on the geometric and scalar properties of a named object. This specification supports a number of predicate languages include the OGC Filter Encoding Implementation Specification [3].

9.7 Sorting

9.7.1 Introduction

Sorting is supported using the **SortBy** clause in a WOS operation. The purpose of the **SortBy** clause is to allow a client to request that the output from a WOS be sorted by specific properties and in a specific order.

9.7.2 XML Encoding

The XML encoding for a **SortBy** clause is defined be the following XML Schema fragment:

```
<xsd:element name="SortBy" type="ogc:SortByType"/>
<xsd:complexType name="SortByType">
   <xsd:sequence>
      <xsd:element ref="ogc:PropertyName" maxOccurs="unbounded"/>
   </xsd:sequence>
   <xsd:attribute name="sortOrder"
                  type="ogc:SortOrderType"
                  default="DESC"
                  use="optional"/>
</xsd:complexType>
<xsd:simpleType name="SortOrderType">
   <xsd:restriction base="xsd:string">
      <xsd:enumeration value="DESC"/>
      <xsd:enumeration value="ASC"/>
   </xsd:restriction>
</xsd:simpleType>
```

The PropertyName element is defined in the Filter Encoding Implmenentation Specification [3].

9.7.3 KVP Encoding

The keyword-value pair encoding for a sort request is accomplished using the **SORTBY** parameter. The parameter accepts a comma-separated list of property names upon which to sort. The property names may be optionally augmented with the addition of a '+A' or '+D' to indicate the sort order. Thus to perform an ascending sort on a property named *PROP1,* the value *PROP1+A* would be specified in the **SORTBY** parameter. Similarly, to perform a descending sort on the *PROP1* property, the value *PROP1+D* would be specified in the **SORTBY** parameter.

The **SORTBY** parameter is formally define as:

SORTBY=*propertyName*[+A|+D][,*propertyName*[+A|+D], …]

The default sort order is ascending or "+A".

9.7.4 Sorting semantics

The results are sorted based on the value of the first property specified in the **SortBy** clause. Each group of features with the same value for the first property is then sorted based on the value of the second property in the **SortBy** clause and so on. Null values are sorted following all others in an ascending sort order and preceding all others in a descending sort order.

9.8 Exception reporting

In the event that a web object service encounters an error while processing a request or receives an unrecognized request, it **shall** generate an XML document indicating that an error has occurred. The format of the XML error response is specified by and **must** validate against the exception response schema defined in annex A.2.

A **<ServiceExceptionReport>** element can contain one or more WOS processing exceptions. The mandatory **version** attribute is used to indicate the version of the service exception report schema; for this version of the specification, this value is fixed at 1.2.0.

Individual exception messages are contained within the **<ServiceException>** element. The optional **code** attribute may be used to associate an exception code with the accompanying message. The optional **locator** attribute may be used to indicate where an exception was encountered in the request that generated the error. A number of elements defined in this document include a **handle** attribute that can be used to associate a mnemonic name with the element. If such a **handle** exists, its value may be reported using the **locator** attribute of the **<ServiceException>** element. If the **handle** attribute is not specified, then a web object server implementation may attempt to locate the error using other means such as line numbers, etc...

Example

The following is an example of an exception report. This exception indicates that the first insert statement failed because of a missing closing XML tag in the request.

```
<?xml version="1.0" ?>
<ServiceExceptionReport
   version="1.2.0"
   xmlns="http://www.opengis.net/ogc"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://www.opengis.net/ogc ../WOS/0.0.1/OGC-exception.xsd">
   <ServiceException code="999" locator="INSERT STMT 01">
      parse error: missing closing tag for element WKB_GEOM
   </ServiceException>
</ServiceExceptionReport>
```

It should be noted that this sample output validates against the exception report schema presented above.

9.9 Common KVP parameters and XML attributes

9.9.1 Version

All KVP & XML encoded WOS operations include a parmeter and attribute called **version**. The mandatory **version** parameter/attribute is used to indicate to which version of the WOS specification the request encoding conforms and is used in version negotiation as described in subclause 8.2.4. The default value of the **version** attributed is set to "0.0.2" which corresponds to the version of this document.

9.9.2 Service

All KVP & XML encoded WOS operations include an attribute or parameter called **service**. The mandatory **service** attribute is used to indicate which of the available service types, at a particular service instance, is being invoked. When invoking a web object service, the value of the **service** attribute shall be WOS.

9.9.3 Handle attribute

The **handle** attribute is only define for XML encoded request messages. The purpose of the **handle** attribute is to allow a client application to associate a mnemonic name with a request message (or part of a request message) for error reporting purposes. If a **handle** is specified, and an exception is encountered, a Web Object Service may use the **handle** to identify the offending element. The KVP encoding does not include a **handle** parameter.

10 GetCapabilities operation

10.1 Introduction

A web object service must have the ability to describe its capabilities. The capabilities are described using an XML document. The capabilities XML document is requested using the *GetCapabilities* operation.

10.2 Request message

10.2.1 KVP encoding

Table 4: KVP Encoding for GetCapabilities Operation

URL COMPONENT	O/M	DEFAULT	DESCRIPTION
REQUEST=GetCapabilities	M		Name of request
SECTION	O		Allows an optional XPATH to be specified to retrieve subsections of

			a capabilities document.

10.2.2 XML encoding

The following XML fragment defines the XML encoding for a **GetCapabilities** operation:

```
From WOS-query.xsd:
<xsd:element name="GetCapabilities"
   type="wos:GetCapabilitiesType"/>

From wos.xsd:
<xsd:complexType name="GetCapabilitiesType">
    <xsd:attribute name="version" type="xsd:string"
   use="required"/>
    <xsd:attribute name="service" type="xsd:string"
   use="required"/>
    <xsd:attribute name="handle"  type="xsd:string"
   use="optional"/>
    <xsd:attribute name="section" type="xsd:string"
   use="optional"/>
</xsd:complexType>
```

10.2.3 Description

The optional **section** parameter/attribute is used to specify an XPATH expression for retrieving sections of a capabilities document. If the **section** parameter/attribute is not defined than the entire capabilities document should be returned.

10.3 Response message

The response XML document generated as a result of a *GetCapabilities* request is fully defined in the document OWS1.2 Service Information Model (02-055) [22].

10.4 Exception message

In the event that a web object service encounters an error servicing a **GetCapabilities** request, it shall raise an exception as described in subclause 9.8.

10.5 Examples

Example 1:

Request the entire capabilities document from a WOS.

```
KVP:
http://www.cubewerx.com/WOS.cgi?
       VERSION=0.0.1&
       SERVICE=WOS&
```

```
        REQUEST=GetCapabilities

XML:
<?xml version="1.0" ?>
<GetCapabilities
   version="0.0.1"
   service="WOS"
   xmlns="http://www.opengis.net/WOS"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://www.opengis.net/WOS ../WOS/0.0.1/WOS-basic.xsd"/>
```

In response to such a request, a web object service would generate a R7 capabilities document as defined in OWS1.2 Capabilities DIPR.

Example 2:

Request the content section of a WOS capabilities document:

```
KVP:
http://www.cubewerx.com/WOS.cgi?
       VERSION=1.0.0&
       SERVICE=WFS&
       REQUEST=GetCapabilities&
       SECTION=OGC_Capabilities/Content

XML:
<?xml version="1.0" ?>
<GetCapabilities
   version="0.0.1"
   service="WOS"
   section="OGC_Capabilities/Content"
   xmlns="http://www.opengis.net/WOS"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://www.opengis.net/WOS ../WOS/0.0.1/WOS-query.xsd"/>
```

In reponse to this request, a WOS would generate a capabilities document that only includes the Content section.

11 DescribeObjectType operation

11.1 Introduction

Object instances are characterized by their object name and by their object type. The object type defines the names and types of all the properties of an object instance. The object name, is the name assigned to the instantiation of the object type. To understand the relationsion between object type, object name and object instance, consider the following object type defined using XML-Schema:

```
<xsd:complexType name="SomeType">
   <xsd:sequence>
      <xsd:element name="Property1" type="xsd:string" />
      <xsd:element name="Property2" type="xsd:positiveInteger"
      />
   </xsd:sequence>
</xsd:complexType>
```

One or more objects of this type may be created by instantiating the type. In XML, this is done by creating elements of type *SomeType*:

```
<xsd:element name="Object1" type="SomeType" />
<xsd:element name="Object2" type="SomeType" />
```

In this example, we have created two objects named, *Object1* and *Object2* of type *SomeType*. To create instances of *Object1* or *Object2* we simply assign values to each property. Example instances of *Object1* and *Object2* might be:

```
<Object1>
   <Property1>A short string</Property1>
   <Property2>10</Property2>
</Object1>

<Object2>
   <Property1>To be or not to be?  That is the
      question.</Property1>
   <Property2>20</Property2>
</Object2>
```

The function of the **DescribeObjectType** operation is to generate a schema description of name and type of objects serviced by a WOS implementation. The purpose of the generated schema is to allow a client to determine the names and types or all the properties of each named object and to allow input and output to be validate againt that schema description using appropriate tools.

The default schema description language is XML-Schema. However, other languages, such as DTD or RDF, are possible as long as they are advertised in the capabilities document. The supported schema description languages would be specified in the capabilities document as binding templates as described in [22].

11.2 Request Message

11.2.1 KVP encoding

Table 4 – DescribeObjectType encoding

URL Component	O/M	DEFAULT	Description
REQUEST=DescribeObjectType	M		Name of request.
OBJECTNAME	O		A comma separated list of object types to describe. If no value is specified that is to be interpreted as all object types.
OUTPUTFORMAT	O	XMLSCHEMA	The output format to use to describe the object. XMLSCHEMA must be supported. Other output formats, such as DTD or RDF are possible.

11.2.2 XML encoding

The XML encoding of a **DescribeObjectType** request is defined by the following XML Schema fragments:

```
From WOS-query.xsd:
<xsd:element name="DescribeObjectType" type="wos:DescribeObjectTypeType"/>

From wos.xsd:
<xsd:complexType name="DescribeObjectTypeType">
   <xsd:sequence>
      <xsd:element name="ObjectName" type="xsd:QName"
                   minOccurs="0" maxOccurs="unbounded"/>
   </xsd:sequence>
   <xsd:attribute name="version" type="xsd:string" use="required"/>
   <xsd:attribute name="service" type="xsd:string" use="required"/>
   <xsd:attribute name="handle"  type="xsd:string" use="optional"/>
   <xsd:attribute name="outputFormat"
                  type="xsd:string" use="optional" default="XMLSCHEMA"/>
</xsd:complexType>
```

11.2.3 Description

The **DescribeObjectType** operation accepts the name of one or more objects that are to be described. If no object names are specified in the request, then all objects that a WOS can service should be described.

The **outputFormat** parameter/attribute, is used to indicate the schema description language that should be used to describe object schemas. The only mandatory output format in response to a **DescribeObjectType** operation is XML Schema denoted by the value **XMLSCHEMA** for the **outputFormat** parameter/attribute. Other vendor specific formats are also possible but they must be advertised on the capabilities document [clause 12].

11.3 Response message

11.3.1 Message format

In response to a keyword-value pair encoded or XML encoded **DescribeObjectType** request, where the value of the **outputFormat** parameter/attribute has been set to **XMLSCHEMA**, a WOS implementation must be able to present an XML Schema [6] document that defines the schema of the objects listed in the request. The document(s) presented by the **DescribeObjectType** request may be used to validate object instances generated by the WOS in the form of object collections on output or object instances specified as input for transaction operations.

Schema descriptions using other schema description languages, such as DTD or RDF, are also possible as long as such capabilities are declared in the capabilities document.

11.3.2 Supporting multiple namespaces

An XML Schema[6] document can only describe elements that belong to a single namespace. This means that a Web Object Service cannot describe objects from multiple

namespaces in a single XML Schema document. To overcome this limitation a WOS may generate an XML Schema document that is a "wrapper" schema that imports the schemas of the objects from the various namespaces in the request. For example, consider the following request:

```
<?xml version="1.0" ?>
<DescribeObjectType
   version="0.0.1"
   service="WOS"
   xmlns="http://www.opengis.net/WOS"
   xmlns:ns01="http://www.server01.com/ns01"
   xmlns:ns02="http://www.server02.com/ns02"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://www.opengis.net/WOS ../WOS/0.0.1/WOS-basic.xsd">
   <ObjectName>ns01:STYLE</ObjectName>
   <ObjectName>ns02:SYMBOL</ObjectName>
</DescribeObjectType>
```

A WOS may generate the following response to this request:

```
<?xml version="1.0" ?>
<schema
   xmlns="http://www.w3.org/2001/XMLSchema"
   elementFormDefault="qualified"
   attributeFormDefault="unqualified">

   <import namespace="http://www.server01.com/ns01"
           schemaLocation="http://www.myserver.com/WOS.cgi?
                           request=DescribeObjectType&amp;objectName=ns01:STYLE"/>

   <import namespace="http://www.server02.com/ns02"
           schemaLocation="http://www.yourserver.com/WOS.cgi?
                           request=DescribeObjectType&amp;objectName=ns02:SYMBOL"/>

</schema>
```

In this example, the WOS is using a **DescribeObjectType** request to obtain the schemas of the objects in the various namespaces. This is simply an example and other methods of obtaining the schemas may be implemented (for example referencing static schema documents).

11.4 Exception message

In the event that a web object service encounters an error while processing a **DescribeObjectType** request message , it shall raise an exception as described subclause 9.8.

11.5 Examples

Example 1

The following example requests the XML Schema description of the object named FeatureStyle.

```
KVP:
http://www.cubewerx.com/WOS.cgi?
        VERSION=0.0.1&
        SERVICE=WFS&
        REQUEST=DescribeObjectType&
```

```
        OBJECTNAME=FeatureStyle

XML:
<?xml version="1.0" encoding="UTF-8"?>
<DescribeObjectType>
   <ObjectName>FeatureStyle</ObjectName>
</DescribeObjectType>

Response:
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema targetNamespace="http://www.opengis.net/sld"
            xmlns:sld="http://www.opengis.net/sld"
            xmlns:ogc="http://www.opengis.net/ogc"
            xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            elementFormDefault="qualified">
  <xsd:include schemaLocation="Symbol.xsd"/>
  <xsd:import namespace="http://www.opengis.net/ogc"
              schemaLocation="../../filter/1.0.0/Filter.xsd"/>

<!--
     ********************************************************
     *************** -->  <xsd:annotation>
    <xsd:documentation>
      SLD FEATURE STYLE version 0.7.3 (2002-08-08)
    </xsd:documentation>
  </xsd:annotation>

  <xsd:element name="FeatureStyle">
  <xsd:annotation>
    <xsd:documentation>
      A FeatureStyle contains styling information specific to
     one
      feature type.  This is the SLD level that separates the
     'layer'
      handling from the 'feature' handling.
    </xsd:documentation>
    </xsd:annotation>
    <xsd:complexType>
      <xsd:choice>
        <xsd:sequence>
          <xsd:element ref="sld:Name" minOccurs="0"/>
          <xsd:element ref="sld:Description" minOccurs="0"/>
          <xsd:element ref="sld:FeatureTypeName"
     minOccurs="0"/>
          <xsd:element ref="sld:SemanticTypeIdentifier"
     minOccurs="0"
                       maxOccurs="unbounded"/>
          <xsd:element ref="sld:Rule" maxOccurs="unbounded"/>
        </xsd:sequence>
        <xsd:element ref="sld:OnlineResource"/>
      </xsd:choice>
      <xsd:attribute name="version" type="sld:VersionType"/>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="SemanticTypeIdentifier"
     type="xsd:string"/>

  <xsd:element name="Rule">
```

```
      <xsd:annotation>
        <xsd:documentation>
          A Rule is used to attach property/scale conditions to
        and group
          the individual symbols used for rendering.
        </xsd:documentation>
      </xsd:annotation>
      <xsd:complexType>
        <xsd:sequence>
          <xsd:element ref="sld:Name" minOccurs="0"/>
          <xsd:element ref="sld:Description" minOccurs="0"/>
          <xsd:element ref="sld:LegendGraphic" minOccurs="0"/>
          <xsd:choice minOccurs="0">
            <xsd:element ref="ogc:Filter"/>
            <xsd:element ref="sld:ElseFilter"/>
          </xsd:choice>
          <xsd:element ref="sld:MinScaleDenominator"
        minOccurs="0"/>
          <xsd:element ref="sld:MaxScaleDenominator"
        minOccurs="0"/>
          <xsd:choice minOccurs="0" maxOccurs="unbounded">
            <xsd:element ref="sld:Symbol"/>
          </xsd:choice>
        </xsd:sequence>
      </xsd:complexType>
    </xsd:element>
    <xsd:element name="LegendGraphic">
      <xsd:complexType>
        <xsd:sequence>
          <xsd:element ref="sld:Graphic"/>
        </xsd:sequence>
      </xsd:complexType>
    </xsd:element>
    <xsd:element name="ElseFilter">
      <xsd:complexType/>
    </xsd:element>
    <xsd:element name="MinScaleDenominator" type="xsd:double"/>
    <xsd:element name="MaxScaleDenominator" type="xsd:double"/>
  </xsd:schema>
```

Example 2

This example describes a fictitious object named *Person.* The *Person* object is of type *PeopleType* which includes a complex property named *Address*.

In response to the **DescribeObjectType** request:

```
<?xml version="1.0" ?>
<DescribeObjectType
   version="0.0.1"
   service="WOS"
   outputFormat="XMLSCHEMA"
   xmlns="http://www.opengis.net/WOS"
   xmlns:myns="http://www.myserver.com/myns"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://www.opengis.net/WOS ../WOS/0.0.1/WOS-basic.xsd">
   <ObjectName>myns:People</ObjectName>
</DescribeObjectType>
```

a web object service might generate an XML Schema document that looks like:

```
<?xml version="1.0" ?>
<xsd:schema
   targetNamespace="http://www.cubewerx.com/myns"
   xmlns:myns="http://www.cubewerx.com/myns"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   xmlns="http://www.w3.org/2001/XMLSchema"
   elementFormDefault="qualified" version="0.1">

   <xsd:element name="Person"
                type="myns:PersonType" />

   <xsd:complexType name="PersonType">
      <xsd:complexContent>
         <xsd:sequence>
            <xsd:element name="LastName" nillable="true">
               <xsd:simpleType>
                  <xsd:restriction base="string">
                     <xsd:maxLength value="30"/>
                  </xsd:restriction>
               </xsd:simpleType>
            </xsd:element>
         <xsd:element name="FirstName" nillable="true">
               <xsd:simpleType>
                  <xsd:restriction base="string">
                     <xsd:maxLength value="10"/>
                  </xsd:restriction>
               </xsd:simpleType>
            </xsd:element>
            <xsd:element name="Age"
                         type="integer"
                         nillable="true"/>
            <xsd:element name="Sex"
                         type="string"/>
            <xsd:element name="Address"
                         type="myns:AddressType"
                         nillable="true"/>
         </xsd:sequence>
       </xsd:complexContent>
   </xsd:complexType>

   <xsd:complexType name="AddressType">
      <xsd:sequence>
         <xsd:element name="StreetName" nillable="true">
            <xsd:simpleType>
               <xsd:restriction base="string">
                  <xsd:maxLength value="30"/>
               </xsd:restriction>
            </xsd:simpleType>
         </xsd:element>
         <xsd:element name="StreetNumber" nillable="true">
            <xsd:simpleType>
               <xsd:restriction base="string">
                  <xsd:maxLength value="10"/>
               </xsd:restriction>
            </xsd:simpleType>
         </xsd:element>
         <xsd:element name="City" nillable="true">
            <xsd:simpleType>
               <xsd:restriction base="string">
                  <xsd:maxLength value="30"/>
               </xsd:restriction>
            </xsd:simpleType>
         </xsd:element>
         <xsd:element name="Province" nillable="true">
            <xsd:simpleType>
               <xsd:restriction base="string">
                  <xsd:maxLength value="30"/>
               </xsd:restriction>
            </xsd:simpleType>
         </xsd:element>
```

```
            <xsd:element name="PostalCode" nillable="true">
               <xsd:simpleType>
                  <xsd:restriction base="string">
                     <xsd:maxLength value="15"/>
                  </xsd:restriction>
               </xsd:simpleType>
            </xsd:element>
            <xsd:element name="Country" nillable="true">
               <xsd:simpleType>
                  <xsd:restriction base="string">
                     <xsd:maxLength value="30"/>
                  </xsd:restriction>
               </xsd:simpleType>
            </xsd:element>
         </xsd:sequence>
      </xsd:complexType>
   </xsd:schema>
```

A sample instance document that validates against this schema might be:

```
<?xml version="1.0" ?>
<wos:ObjectCollection
xmlns="http://www.cubewerx.com/myns"
xmlns:myns="http://www.cubewerx.com/myns"
xmlns:wos="http://www.opengis.net/WOS"
xmlns:gml="http://www.opengis.net/gml"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.opengis.net/WOS ../WOS/0.0.1/WOS-basic.xsd
                    http://www.cubewerx.com/myns ex10.xsd">
   <wos:objectInstance>
      <Person>
         <myns:LastName>Smith</myns:LastName>
         <myns:FirstName>Fred</myns:FirstName>
         <myns:Age>35</myns:Age>
         <myns:Sex>Male</myns:Sex>
         <myns:Location>
            <gml:Point><gml:coordinates>15,15</gml:coordinates></gml:Point>
         </myns:Location>
         <myns:Address>
            <myns:StreetName>Main St.</myns:StreetName>
            <myns:StreetNumber>5</myns:StreetNumber>
            <myns:City>SomeCity</myns:City>
            <myns:Province>SomeProvince</myns:Province>
            <myns:PostalCode>X1X 1X1</myns:PostalCode>
            <myns:Country>Canada</myns:Country>
         </myns:Address>
      </Person>
   </wos:objectInstance>
</wos:ObjectCollection>
```

12 GetObjectById operation

12.1 Introduction

The **GetObjectById** operation is an abstract operation in that its encoding is not specifically defined by this specification. Its existence is the result of the requirements stated in subclause 6.2 concerning global identifiers for objects. A WOS must able to ingest its own global object identifiers, expressed as URLs, and return the associated object instances.

12.2 Request message

The encoding of the global object identifier that a WOS generates for an object instance is an implementation specific detail and is not specified by this document. The global object identifier, must however, satisfy the requirements stated in clause 6.2.

12.3 Response message

A valid global object identifer must resolve to a valid instance of the object associated with the identifier without any extra packaging such as that used for collections.

12.4 Exception message

In the event that a web object service encounters an error servicing a DescribeObjectType request, it shall raise an exception as described subclause 9.8.

12.5 Examples

Example 1:

The global object identifier:

```
http://www.some_sms_server.com/sms.cgi/objectid=A1F7B
```

might resolve to the following style instance:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<FeatureTypeStyle
   xsi:schemaLocation="http://www.opengis.net/sld FeatureTypeStyle.xsd"
   xmlns="http://www.opengis.net/sld"
   xmlns:ogc="http://www.opengis.net/ogc"
   xmlns:xlink="http://www.w3.org/1999/xlink"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

   <FeatureTypeName>OCEANSEA_1M:Foundation</FeatureTypeName>
   <Rule>
      <Name>main</Name>
      <PolygonSymbolizer>
         <Geometry>
            <ogc:PropertyName>GEOMETRY</ogc:PropertyName>
         </Geometry>
         <Fill>
            <CssParameter name="fill">#96C3F5</CssParameter>
         </Fill>
      </PolygonSymbolizer>
   </Rule>
</FeatureTypeStyle>
```

Example 2:

The global object identifier:

```
http://www.cubewerx.com/sms.cgi?request=GetObjectById&objectid=SYMBOL.2456
```

might resolve to the following object symbol instance:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<PolygonSymbolizer
   xsi:schemaLocation="http://www.opengis.net/sld Symbolizer.xsd"
```

```
      xmlns="http://www.opengis.net/sld"
      xmlns:ogc="http://www.opengis.net/ogc"
      xmlns:xlink="http://www.w3.org/1999/xlink"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
      <Geometry>
         <ogc:PropertyName>GEOMETRY</ogc:PropertyName>
      </Geometry>
      <Fill>
         <CssParameter name="fill">#96C3F5</CssParameter>
      </Fill>
   </PolygonSymbolizer>
```

It should be noted, that the encoding of this object identifier is totally at the discression of the web object service. This form of the **GetObjectById** operation is not normative. The **GetObjectById** operation has no normative encoding.

13 GetObject operation

13.1 Introduction

The **GetObject/GetObjectWithLock** operation allows retrieval of object instances from an object repository.

The encoding of the **GetObjectWithLock** operation is identical to the encoding of the **GetObject** operation. The only difference is that the **GetObjectWithLock** request attempts to lock all object instances identified by the operation.

13.2 Request message

13.2.1 KVP Encoding

Table 4 - GetObject & GetObjectWithLock encoding

URL Component	O/M	DEFAULT	Description
REQUEST=[GetObject | GetObjectWithLock]	M		The name of the WOS operation.
PROPERTYNAME (**mutually exclusive** with the PROPERTYSET parameter)	O		A list of properties may be specified for each object type that is being queried. Refer to subclause [8.5.2.2.2] on how to form lists of parameters. A "*" character can be used to indicate that all properties should be retrieved. There is a 1:1 mapping between each element in an objectID or OBJECTNAME list and the PROPERTYNAME list. The absense of a value also indicates that all properties should be fetched.
PROPERTYSET	O		Specifies a named set of properties that is known to the server rather than enumerating

(**mutually exclusive** with the PROPERTYNAME parameter)			each property name one-by-one.
OBJECTVERSION=[ALL | N]	O		If versioning is supported, the OBJECTVERSION parameter directs the WOS on which object version to fetch.. A value of ALL indicates to fetch all versions of an object. An integer value fetches the Nth version of an object. No value indicates that the latest version of the object should be fetched.
MAXOBJECTS=N	O		A positive integer indicating the maximum number of objects that the WOS should return in response to a query. If no value is specified then all result instances should be presented.
LOCKACTION=[ALL|SOME]	O	ALL	Defines how a GetFeatureWithLock operation should try to acquire object instance locks. A value of ALL means that a WOS should get locks on all the target object instances or fail. A value of SOME means that a WOS should ry to get as many locks as possible.
OBJECTNAME (**Optional** if OBJECTID is specified.)	M		A list of object type names to query.
OBJECTID (**Mutually exclusive** with FILTER and BBOX)	O		An enumerated list of object instances to fetch identified by their object identifiers.
FILTER (**Prerequisite**:OBJECTNAME) (**Mutually exclusive** with OBJECTID and BBOX)	O		A FILTER SPECIFICATION DESCRIBES A SET OF OBJECTS TO OPERATE UPON. THE PREDICATE LANGUAGE THAT IS BEING USED IS SPECIFIED USING THE FILTERLANGUAGE PARAMETER. POSSIBLE VALUES INCLUDE OGCFILTER, CQLTEXT, SFSQL AND SQLMM. IF THE FILTER PARAMETER IS USED, ONE FILTER MUST BE SPECIFIED FOR EACH OBJECT TYPE LISTED IN THE OBJECTNAME PARAMETER. INDIVIDUAL FILTERS ENCODED IN THE FILTER PARAMETER ARE ENCLOSED IN PARENTHESES "(" AND ")".

FILTERLANGUAGE (**Prerequisite:** FILTER)	O	OGCFILTER	DEFINES THE PREDICATE LANGUAGE USED IN THE FILTER PARAMETER. THE POSSIBLE VALUES ARE OGCFILTER (FOR THE OGC FILTER ENCODING SPECIFICATION, CQLTEXT (FOR THE COMMON QUERY LANGUAGE), SFSQL (FOR SIMPLE FEATURE SQL), SQLMM (FOR ISO SQL MM).
BBOX (**Prerequisite**:OBJECTNAME) (**Mutually exclusive** with OBJECTID and FILTER.)	O		In lieu of an objected or FILTER, a client may specify a bounding box as described in subclause 8.5.2.3.3.
SORTBY	O		Allows the client to specify a sort by clause for a query. Multiple SORTBY clauses, for multiple queries, can be specified by using parentheses to enclose the individual clauses.

13.2.2 XML Encoding

The XML encoding of a **GetObject/GetObjectWithLock** request is defined by the following XML Schema fragments:

```
From WOS-query.xsd
<xsd:element name="GetObject">
   <xsd:complexType>
      <xsd:complexContent>
         <xsd:extension base="wos:GetObjectType">
            <xsd:attribute name="outputFormat"
                           type="xsd:string"
                           use="optional" default="XML" />
            <xsd:attribute name="archiver"
                           type="ows:knownArchivers"
                           use="optional"/>
            <xsd:attribute name="compressor"
                           type="ows:knownCompressors"
                           use="optional"/>
         </xsd:extension>
      </xsd:complexContent>
   </xsd:complexType>
</xsd:element>


From WOS-transaction.xsd:
<xsd:element name="GetObjectWithLock">
   <xsd:complexType>
      <xsd:complexContent>
         <xsd:extension base="wos:GetObjectWithLockType">
            <xsd:attribute name="outputFormat"
```

```
                              type="xsd:string"
                              use="optional" default="XML" />
            </xsd:extension>
         </xsd:complexContent>
      </xsd:complexType>
   </xsd:element>

   From wos.xsd:
      <xsd:complexType name="GetObjectType">
         <xsd:sequence>
            <xsd:element ref="wos:Query" maxOccurs="unbounded"/>
         </xsd:sequence>
         <xsd:attribute name="version" type="xsd:string"
         use="required" />
         <xsd:attribute name="service" type="xsd:string"
         use="required" />
         <xsd:attribute name="handle"  type="xsd:string"
         use="optional" />
      </xsd:complexType>
      <xsd:complexType name="GetObjectWithLockType">
         <xsd:sequence>
            <xsd:element ref="wos:Query" maxOccurs="unbounded"/>
         </xsd:sequence>
         <xsd:attribute name="version" type="xsd:string"
         use="required" />
         <xsd:attribute name="service" type="xsd:string"
         use="required" />
         <xsd:attribute name="handle"  type="xsd:string"
         use="optional" />
         <xsd:attribute name="lockAction" type="wos:AllSomeType"
         use="optional"/>
      </xsd:complexType>
      <xsd:element name="Query">
         <xsd:complexType>
            <xsd:complexContent>
               <xsd:extension base="wos:QueryType"/>
            </xsd:complexContent>
         </xsd:complexType>
      </xsd:element>
      <xsd:complexType name="QueryType">
         <xsd:sequence>
            <xsd:choice>
               <xsd:element name="wos:PropertySet"
                            type="wos:PropertySetType"
                            minOccurs="0" maxOccurs="1"/>
               <xsd:element ref="ogc:PropertyName"
                            minOccurs="0" maxOccurs="unbounded"/>
            </xsd:choice>
            <xsd:element name="QueryConstraint"
                         type="QueryConstraintType"
                         minOccurs="0" maxOccurs="1"/>
            <xsd:element ref="ogc:SortBy"
                         minOccurs="0" maxOccurs="1"/>
         </xsd:sequence>
         <xsd:attribute name="handle"
                        type="xsd:string" use="optional"/>
         <xsd:attribute name="objectName"
```

```
                    type="wos:ObjectNameListType"
    use="required"/>
     <xsd:attribute name="objectVersion"
                    type="xsd:string" use="optional"/>
</xsd:complexType>
<xsd:complexType name="QueryConstraintType">
    <xsd:choice>
       <xsd:element ref="ogc:Filter" />
       <xsd:element name="CqlText" type="xsd:string" />
       <xsd:element name="SfSql"   type="xsd:string" />
       <xsd:element name="SqlMM"   type="xsd:string" />
    </xsd:choice>
</xsd:complexType>
<xsd:simpleType name="ObjectNameListType">
    <xsd:list itemType="xsd:QName"/>
</xsd:simpleType>
<xsd:complexType name="PropertySetType">
    <xsd:attribute name="setName" type="xsd:string"
    use="optional"/>
</xsd:complexType>
```

13.2.3 Description

A **GetObject** operation allows one or more queries to be executed in order to retrieve object instances from the repository.

In the XML encoding, the <**GetObject**> element contains one or more <**Query**> elements each of which in turn contain the description of a query. The results of all queries contained in a **GetObject** request are concatenated to produce the result set.

Each individual query packaged in a **GetObject** request is defined using the <**Query**> element. The <**Query**> element specifies the name of the object to query, what properties to retrieve and what constraints (spatial and non-spatial) to apply to those properties.

The **objectName** attribute or **OBJECTNAME** property is used to indicate the name of the object class to be queried.

The **outputFormat** attribute or **OUTPUTFORMAT** parameter defines the format to use to generate the result set. The default value is **XML** indicating that XML [2] shall be used. Vendor specific formats (including non-XML and binary formats), declared in the capabilities document are also possible.

The **objectName** attribute is used to indicate the name of the object to be queried.

The **objectVersion** parameter/attribute is included in order to accommodate systems that support object versioning. A value of **ALL** indicates that all versions of an object should be fetched. Otherwise an integer, *n*, can be specified to return the n^{th} version of an object. The version numbers start at '1' which is the oldest version. If a version value larger than the largest version number is specified then the latest version is return. The default action

shall be for the query to return the latest version. Systems that do not support versioning can ignore the parameter and return the only version that they have.

The **<PropertyName>** element is used to enumerate the object properties that should be selected during a query and whose values should be included in the response to a **GetObject** request. A client application can determine the properties of an object by making a **DescribeObjectType** request before composing a **GetObject** request. The **DescribeObjectType** operation, described in clause 11, will generate an XML Schema document defining the schema of the object and the client can then select the properties to be fetched. In addition, the client can determine which object properties are mandatory and must be fetched in order for the WOS to be able to generate an instance of the object that will validate againt the generated schema. In the event that a WOS encounters a query that does not select all mandatory properties of an object, the WOS will internally augment the property name list to include all necessary property names. A WOS client must thus be prepared to deal with a situation where it receives more property values than it requests.

If no **<PropertyName>** elements are specified for the XML encoding, then all object properties should be fetched.

The <**PropertySetName**> element is used to reference a named subset of object properties. This element may be used to create views of on object type. An example may be found in the registry/catalogue domain. In that domain there are the concepts of brief, summary and full views on the data. Associated with each names is a set of properties. The property set name **BRIEF**, only fetches a small number of properties of a registry object. The set name **SUMMARY** is used to fetch a larger number of properties of a registry object properties giving a more detailed view of the object instances. Finally, the set name **FULL** included even more properties (although it may not be all the properties) to give a yet more detailed view of the data.

The <**QueryConstraint**> element can be used to define constraints on a query. Both spatial and/or non-spatial constraints can be specified depending on the predicate language used. If no <**QueryConstraint**> element is contained within the <**Query**> element, then the query is unconstrained and all object instances should be retrieved.

The <**QueryConstaint**> element may contain the elements <**Filter**>, <**CqlText**>, <**SfSql**> and <**SqlMM**>. The <**Filter**> element indicates that the predicate language being used is the OGC Filter Encoding Specification[3]. The <**CqlText**> element is used to indicate that the Common Query Language, defined in [14] , is the predicate language being used. The element <**SfSql**> is used to indicates that the SQL dialect described in the Simple Features for SQL Specification [15] is the predicate language. Finally, the element <**SqlMM**> indcates that the ISO SQL MM dialect is being used as the predicate language.

EDITOR'S NOTE:
This idea of supporting multiple query languages comes for the stateless catalog specification. I personally think it is a bad idea since it hinders interoperability. I think we should pick one predicate language and be done with it. My

recommendation is the Filter Encoding Specification since it encodes predicates in XML (making it easy to parse the predicate with existing XML parsers) and is the common predicate language among all OGC web services

.

The WOS includes the concept of a named sub-set of properties of an object. The **PROPERTYSET** parameter or the <**PropertySet**> element can be used to reference a set of named properties, instead of specifying a list of property name values for the **PROPERTYNAME** parameter or a set of **<PropertyName>** elements.

The <**SortBy**> element for XML encoding may be used to specify that the results of a query be sorted by specific properties and in a specific order.

The encoding of the **GetObjectWithLock** operation is identical to the **GetObject** operation except that it indicates to a web object service to attempt to lock the objects that are selected; presumably to update the objects.

13.3 Response message

The response to a **GetObject** operation is a collection of object instances that satisfy the query constraints defined in the operation. For the default output format of XML, the results are packaged in an **<ObjectCollection>** element which is a generic container of object instances. The **<ObjectCollection>** element is defined by the following XML-Schema fragment:

```
<xsd:element name="ObjectCollection"
             type="wos:ObjectCollectionType" />

<!--
   ============================================================
   ===
      RESPONSE TYPES

   ============================================================
   === -->
<xsd:complexType name="ObjectCollectionType">
   <xsd:sequence>
      <xsd:element name="ObjectInstance"
                   type="wos:ObjectInstanceType"
                   maxOccurs="unbounded" />
   </xsd:sequence>
   <xsd:attribute name="lockId" type="xsd:string"
  user="optional" />
</xsd:complexType>
<xsd:complexType name="ObjectInstanceType">
   <xsd:sequence>
      <xsd:choice>
         <xsd:any namespace="##targetNamespace"/>
         <xsd:element ref="wos:ObjectRef" />
      </xsd:choice>
   </xsd:sequence>
   <xsd:attribute name="oid" type="xsd:anyURI" use="required"
   />
</xsd:complexType>
```

An **<ObjectCollection>** element contains one or more **<ObjectInstance>** elements. Each **<ObjectInstance>** element contains an instance or a reference to an instance of an object in the result set of the query. Each **<ObjectInstance>** element also includes an **oid** attribute that encodes the unique object identifier for each object instance.

For the **<GetObjectWithLock>** operation, a WOS must generate a result that includes the lock identifier. The lock identifier is encoded using the **lockId** attribute that is defined on the <**ObjectCollection**> element. The following XML fragment illustrates how to include the **lockId** attribute in the response to the operation:

```
<wos:ObjectCollection lockId="00A01"… >
…
</wos:ObjectCollection>
```

The ellipses are meant to represent all the other components included in the **GetObjectWithLock** response which are identical to the components included in the **GetObject** response.

13.3.1 Response validation

The response format is controlled by the **outputFormat** attribute for XML encoded request message or the **OUTPUTFORMAT** parameter for KVP encoded request messages. In both cases, the default value of **XML** shall indicate that a WOS must generate an XML document of the result set whose object instances **must** validate against an XML Schema document generated by the **DescribeObjectType** operation [see clause 11].

The XML document generated by a WOS implementation, in response to a query where the output format is **XML**, must reference an appropriate XML Schema document so that the output can be validated. This can be accomplished using the **schemaLocation** attribute, as defined in [6], to provide hints as to the physical location of one or more schema documents which may be used for local validation and schema-validity assessment. The **schemaLocation** attribute value contains pairs of values. The first member of each pair is the namespace for which the second member is the hint describing where to find to an appropriate schema document. The physical location of the schema documents is specified using a URI [10].

The following XML fragment shows the use of the **schemaLocation** attribute on the root element indicating the location of the an XML Schema document that can be used for validation:

```
<?xml version="1.0" ?>
<wos:ObjectCollection
   xmlns="http://www.opengis.net/myns"
   xmlns:myns="http://www.opengis.net/myns"
   xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
   xsi:schemaLocation="http://www.opengis.net/myns
                       http://www.cubewerx.com/WOS.cgi?
                       request=DescribeObjectType&objectname=SYMBOL"> …
```

In this instance, the schema document corresponding to the *myns* namespace is dynamically generated by making a **DescribeObjectType** request, back to the server that

generated the output, requesting the schema. This **DescribeObjectType** operation [see clause 11] requests the schema of the object SYMBOL in the *myns* namespace.

It is up to each WOS implementation to arrange that the XML output that it generates makes the appropriate **schemaLocation** reference(s) such that the output can be validated.

13.3.2 Non-XML responses

In many instance, the objects that a web object service manipulates may not be easily or conveniently encoded in XML. In such instances, the WOS response may include object references to the object instances. The actual object data may be packaged with the response in a multi-part mime document or exist in some accessible location on the web.

13.4 Exception message

In the event that a web object service encounters an error servicing a **GetObject** request, it shall raise an exception as described in sebclause 9.8.

13.5 Examples

This section contains numerous examples of the **GetObject** request. Some examples include sample output.

Example 1

The following example searches a feature style repository for all instances of styles that may be applied to the feature INWATERA_1M.

```
KVP:
http://www.cubewerx.com/sms.cgi?request=GetObject&objectName=F
      eatureStyle&

      Filter=<ogc:Filter><ogc:PropertyIsEqualTo>

      <ogc:PropertyName>FeatureStyle/FeatureTypeName

      </ogc:PropertyName><ogc:Literal>INWATERA_1M

      </ogc:Literal></ogc:PropertyIsEqualTo></ogc:Filter>

XML:
<?xml version="1.0"?>
<wos:GetObject>
   <wos:Query objectName=”FeatureStyle”>
      <ogc:Filter>
         <ogc:PropertyIsEqualTo>

      <ogc:PropertyName>FeatureStyle/FeatureTypeName</ogc:Prop
      ertyName>
            <ogc:Literal>INWATERA_1M</ogc:Literal>
         </ogc:PropertyIsEqualTo>
```

```
      </ogc:Filter>
   </wos:Query>
</wos:GetObject>

Response:
<?xml version="1.0" ?>
<wos:ObjectCollection>
   <wos:ObjectInstance
      oid="http://www.cubewerx.com/sms.cgi?objectid=100">
       <FeatureStyle version="0.7.3"

      xsi:schemaLocation="http://www.opengis.net/sld
      FeatureStyle.xsd"
                    xmlns="http://www.opengis.net/sld"
                    xmlns:ogc="http://www.opengis.net/ogc"
                    xmlns:xlink="http://www.w3.org/1999/xlink"

      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
          <FeatureTypeName>INWATERA_1M</FeatureTypeName>
          <Rule>
             <Name>main</Name>
                <PolygonSymbol>
                   <Geometry>

      <ogc:PropertyName>GEOMETRY</ogc:PropertyName>
                   </Geometry>
                   <Fill>
                      <SvgParameter
      name="fill">#96C3F5</SvgParameter>
                   </Fill>
                </PolygonSymbol>
          </Rule>
       </FeatureStyle>
   </wos:ObjectInstance>
   <wos:ObjectInstance
      oid="http://www.cubewerx.com/sms.cgi?objectid=101">
       <FeatureStyle version="0.7.3"

      xsi:schemaLocation="http://www.opengis.net/sld
      FeatureStyle.xsd"
                    xmlns="http://www.opengis.net/sld"
                    xmlns:ogc="http://www.opengis.net/ogc"
                    xmlns:xlink="http://www.w3.org/1999/xlink"

      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
          <FeatureTypeName>INWATERA_1M</FeatureTypeName>
          <Rule>
             <Name>AlternameStyle</Name>
                <PolygonSymbol>
                   <Geometry>

      <ogc:PropertyName>GEOMETRY</ogc:PropertyName>
                   </Geometry>
                   <Fill>
                      <SvgParameter
      name="fill">#96C3F5</SvgParameter>
                   </Fill>
                </PolygonSymbol>
```

```
         </Rule>
      </FeatureStyle>
   </wos:ObjectInstance>
</wos:ObjectCollection>
```

Example 2

This is meant to be a simple image archive example for a images collected by a fictitious satellite SAT5. Objects stored in the archive have only two properties. There is a COLLECTION_DATE property indicating when the image was collected. There is also the IMAGE property which stores or references the actual pixels of the image. Depending on the imeplementation, the image pixles may be stored in-line in the repository or at some offline location.

The following **GetObject** request message will retrieve all images collected during the month of August:

KVP:

```
http://www.someserver.com/WOS.cgi?
       service=WOS&
       version=0.0.2&
       request=GetObject&
       objectname=SAT5_IMAGES&

      filter=<ogc:Filter><ogc:PropertyIsBetween><ogc:PropertyN
      ame>

      COLLECTION_DATE</ogc:PropertyName><ogc:LowerBoundary>
              01-AUG-
      2002</ogc:LowerBoundary><ogc:UpperBoundary>
              30-AUG-
      2002<ogc:UpperBoundary></ogc:PropertyIsBetween>
              </ogc:Filter>
```

XML:

```
<?xml version="1.0" ?>
<wos:GetObject
   service="WOS"
   version="0.0.2
   xmlns:myns="http://www.cubewerx.com/myns"
   xmlns:wos="http://www.opengis.net/wos"
   xmlns:ogc="http://www.opengis.net/ogc"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://www.opengis.net/wos
      ../WOS/0.0.1/WOS-basic.xsd">
   <wos:Query objectName="myns:SAT5_IMAGES">
      <wos:PropertyName>Image</wos:PropertyName>
      <ogc:Filter>
         <ogc:PropertyIsBetween>
            <ogc:PropertyName>COLLECTION_DATE</PropertyName>
            <ogc:LowerBoundary>01-AUG-2002</ogc:LowerBoundary>
            <ogc:UpperBoundary>30-AUG-2002</ogc:LowerBoundary>
         </ogc:PropertyIsBetween>
      </ogc:Filter>
   </wos:Query>
```

```
</wos:GetObject>

Response:
Content-type: multipart/mixed; boundary="wos12345"

-- wos12345
Content-type: text/xml
Content-Length: 99999
Date: 29-AUG-2002

<wos:ObjectCollection>
   <wos:ObjectInstance
      oid="http://www.cubewerx.com/wos.cgi?objectid=10">
       <SAT5_IMAGES>
          <COLLECTION_DATE>03-AUG-2002</COLLECTION_DATE>
          <IMAGE wos:mimetype="image/geotiff" wos:href="10" />
       </SAT5_IMAGES>
   </wos:ObjectInstance>
   <wos:ObjectInstance
      oid="http://www.cubewerx.com/wos.cgi?objectid=11">
       <SAT5_IMAGES>
          <COLLECTION_DATE>29-AUG-2002</COLLECTION_DATE>
          <IMAGE wos:mimetype="image/geotiff" wos:hfre="11" />
       </SAT5_IMAGES>
   </wos:ObjectInstance>
</wos:ObjectCollection>

-- wos12345
Content-ID: 10
Content-type: image/geotiff
Content-Length: 190458347
Date: 30-AUG-2002

... 198458347 bytes of geotiff image ...

-- wos12345
Content-ID: 20
Content-type: image/geotiff
Content-Length: 5786987680
Date: 30-AUG-2002

... 5786987680 bytes of geotiff image ...
```

The response in this example is a multipart MIME document. The first part of the document is the standard web object server response encoded in XML. The other parts of the response are the images files in their native format. The XML response references the included image files using the **objectRef** attribute group defined in clause 9.4 and the *Content-ID* component of the MIME headers.

14 Transaction operation

14.1 Introduction

The **Transaction** operation is used to describe data transformation operations that are to be applied to web accessible object instances. A web object service may process a **Transaction** operation directly or possibly translate it into the language of a target datastore to which it is connected and then have the datastore execute the transaction. When the transaction has been completed, a web object service will generate an XML response document indicating the completion status of the transaction.

The **Transaction** operation is optional and a WOS implementation does not need to support it to conform to this specification. If the **Transaction** operation is supported then this fact must be advertised on the capabilities document as described in clause 10.

14.2 Request message

14.2.1 KVP Encoding

Table 5 - Transaction encoding

URL Component	O/M	DEFAULT	Description
REQUEST=Transaction	M		The name of the WOS operation.
OPERATION=[INSERT | UPDATE | DELETE]	M		Transaction operation to execute.
OBJECTNAME	M		The name of the object or objects to operate on.

OBJECTMIME OBJECT (parameter may only be specified for INSERT operation)	O O	text/xml	These parameter are always specified together. The OBJECTMIME parameter is the MIME type of the OBJECT parameter. The default is text/xml. However, other formats are possible. This OBJECT parameter is used to specify the value of a new object instance. The value may be the actual text of an XML fragment that defines the value for the object. Multiple objects may be specified by delimiting each object with parentheses. The value of the OBJECT parameter may also be a URL pointing to an object in which case the WOS must verify that the link is valid and optional may retrieve the object and store it in its local repository based on the value of RESOLVEURL prameter.
PROERTYNAME PROPERTYMIME PROPERTYVALUE	O O O		These parameters are always specified together. In contrast to the "OBJECT" parameters which allow whole objects to be manipulated, the PROPERTYNAME parameter is used to specify the name of an object property whose value is to be modified. The name of the property must be an Xpath expression as specified in this document. The PROPERTYMIME parameter is the MIME type of the PROPERTYVALUE parameter. There is no default meanin that the value specified should be taken as is. Otherwise the value of the PROPERTYVALUE parameter should be interpreted as indicated by the MIME type. The value of a property may be explicitly specified using the PROPERTYVALUE parameter. Alternatively, the PROPERTYVALUE may be a URL reference in which case, the PROPERTYMIME parameter must be specified indicating how the value

(**mutually exclusive** with the OBJECT parameter) (parameters may only be specified with INSERT or UPDATE operation)			should be interpreted. If the value of the PROPERTYVALUE parameter is URL reference then the WOS must validate the link at the time of the request and may optionally retrieve the content and store it in a local repository as specified by the RESOLVEURL parameter.
RESOLVEURL	O		If the value of a parameter is a URL, this parameter indicates how a WOS should handle the reference. A value of TRUE indicates that the WOS should retrieve the resource and store it in its local repository. A value of FALSE indicates that the WOS should simply maintain the reference as a string object. In either case, the WOS must verify that the reference is valid and active at the time it checks it.
RELEASEACTION=[ALL|SOME]	O	TRUE	A value of ALL indicates that all object locks should be released when a transaction terminates. A value of SOME indicates that only those records that are modified should be released. The remaining locks are maintained
OBJECTID (**Mutually exclusive** with FILTER and BBOX) (parameter may only be specified for UDPATE and DELETE operations)	O		A list of object identifiers upon which the specified operation shall be applied.
FILTER (**Prerequisite**:OBJECTNAME) (**Mutually exclusive** with OBJECTID and BBOX)	O		A FILTER SPECIFICATION DESCRIBES A SET OF OBJECTS TO OPERATE UPON. THE PREDICATE LANGUAGE THAT IS BEING USED IS SPECIFIED USING THE FILTERLANGUAGE PARAMETER. POSSIBLE VALUES INCLUDE OGCFILTER, CQLTEXT, SFSQL AND SQLMM. IF THE FILTER PARAMETER IS USED, ONE FILTER MUST BE SPECIFIED FOR EACH OBJECT TYPE LISTED IN THE OBJECTNAME PARAMETER. INDIVIDUAL FILTERS ENCODED IN THE FILTER PARAMETER ARE ENCLOSED IN PARENTHESES "("

			AND “)”.
FILTERLANGUAGE (**Prerequisite:** FILTER)	O	OGCFILTER	DEFINES THE PREDICATE LANGUAGE USED IN THE FILTER PARAMETER. THE POSSIBLE VALUES ARE OGCFILTER (FOR THE OGC FILTER ENCODING SPECIFICATION, CQLTEXT (FOR THE COMMON QUERY LANGUAGE), SFSQL (FOR SIMPLE FEATURE SQL), SQLMM (FOR ISO SQL MM).
BBOX (**Prerequisite**: OBJECTNAME, REQUEST=UPDATE or REQUEST=DELETE) (**Mutually exclusive** with FILTER and OBJECTID)	O		In lieu of an objectID or FILTER, a client may specify a bounding box as described in subclause 8.5.2.3.3.

14.2.2 XML encoding

The XML encoding of a **Transaction** request is defined by the following XML Schema fragments:

```
From WOS-transaction.xsd:
<xsd:element name="Transaction"
             type="wos:TransactionType"/>
<xsd:element name="TransactionResponse"
             type="wos:TransactionResponseType"/>

From wos.xsd:
<xsd:complexType name="TransactionType">
   <xsd:sequence>
      <xsd:element ref="wos:LockId" minOccurs="0" maxOccurs="1" />
      <xsd:choice minOccurs="0" maxOccurs="unbounded">
         <xsd:element ref="wos:Insert" minOccurs="0" maxOccurs="unbounded"/>
         <xsd:element ref="wos:Update" minOccurs="0" maxOccurs="unbounded"/>
         <xsd:element ref="wos:Delete" minOccurs="0" maxOccurs="unbounded"/>
         <xsd:element ref="wos:Native" minOccurs="0" maxOccurs="unbounded"/>
      </xsd:choice>
   </xsd:sequence>
   <xsd:attribute name="version" type="xsd:string" use="required" />
   <xsd:attribute name="service" type="xsd:string" use="required" />
   <xsd:attribute name="handle"  type="xsd:string" use="optional" />
   <xsd:attribute name="releaseAction"
                  type="wos:AllSomeType" use="optional"/>
</xsd:complexType>
<xsd:element name="LockId" type="xsd:string"/>
<xsd:element name="Insert" type="wos:InsertElementType"/>
<xsd:element name="Update" type="wos:UpdateElementType"/>
<xsd:element name="Delete" type="wos:DeleteElementType"/>
<xsd:complexType name="InsertElementType">
   <xsd:sequence>
      <xsd:choice>
         <xsd:any namespace="##targetNamespace"/>
         <xsd:element ref="wos:ObjectRef" />
      </xsd:choice>
```

```
      </xsd:sequence>
      <xsd:attribute name="handle" type="xsd:string" use="optional"/>
   </xsd:complexType>
   <xsd:complexType name="UpdateElementType">
      <xsd:sequence>
         <xsd:element ref="wos:Property"
                      maxOccurs="unbounded"/>
         <xsd:element name="QueryConstraint"
                      type="QueryConstraintType"
                      minOccurs="0" maxOccurs="1"/>
      </xsd:sequence>
      <xsd:attribute name="handle" type="xsd:string" use="optional"/>
      <xsd:attribute name="objectName" type="xsd:QName" use="required" />
   </xsd:complexType>
   <xsd:complexType name="DeleteElementType">
      <xsd:sequence>
         <xsd:element name="QueryConstraint"
                      type="QueryConstraintType"
                      minOccurs="0" maxOccurs="1" />
      </xsd:sequence>
      <xsd:attribute name="handle" type="xsd:string" use="optional"/>
      <xsd:attribute name="objectName" type="xsd:QName" use="required" />
   </xsd:complexType>
   <xsd:element name="Property" type="wos:PropertyType"/>
   <xsd:complexType name="PropertyType">
      <xsd:sequence>
         <xsd:element name="Name" type="xsd:string"/>
         <xsd:choice minOccurs="1" maxOccurs="1">
            <xsd:element name="Value"/>
            <xsd:element name="ValueRef" type="wos:ValueRefType"/>
         </xsd:choice>
      </xsd:sequence>
   </xsd:complexType>
   <xsd:complexType name="ValueRefType">
      <xsd:attribute name="resolveURL"
                     type="xsd:boolean" use="optional" default="true" />
      <xsd:attribute name="mimeType"
                     type="xsd:string" use="optional" default="text/xml" />
      <xsd:attributeGroup ref="wos:objRef"/>
   </xsd:complexType>
   <xsd:element name="ObjectRef" type="wos:ObjectRefType"/>
   <xsd:complexType name="ObjectRefType">
      <xsd:attribute name="resolveURL"
                     type="xsd:boolean" use="optional" default="true" />
      <xsd:attribute name="mimeType"
                     type="xsd:string" use="optional" default="text/xml" />
      <xsd:attributeGroup ref="wos:objRef"/>
   </xsd:complexType>
   <xsd:attributeGroup name="objRef">
      <xsd:annotation>
         <xsd:documentation>
         </xsd:documentation>
      </xsd:annotation>
      <xsd:attribute name="resolveURL"
                     type="xsd:boolean"
                     use="optional"
                     default="true" />
      <xsd:attribute name="mimeType"
                     type="xsd:string"
                     use="optional"
                     default="text/xml"/>
      <xsd:attribute name="href"
                     type="xsd:anyURI"
                     use="required" />
   </xsd:attributeGroup>
```

14.2.3 Description

As described in subclause 9.9.3, the **handle** attribute can be used to assign a mnemonic name to the element with which it is associated. Its intended use, is to make error reporting more meaningful for a client application. In the event that an error is

encountered, the **handle** attribute can be used by a web object service to locate the error when generating an exception report. In the event that no **handle** is specified, a web object service may attempt to report the location of the exception relative to the current **Transaction** request using line numbers of some other convenient mechanism.

Assuming that a WOS implementation supports the optional **LockObject** and/or **GetObjectWithLock** operations, the **releaseAction** attribute is used to control how locked objects are treated, when a transaction request is completed. A value of **ALL** indicates that the locks on all object instances locked using the specified **<LockId>** should be released when the transaction completes, regardless of whether or not a particular object instance in the locked set was actually operated upon. A value of **SOME** indicates that only the locks on object instances modified by the transaction should be released. The other, unmodified, object instances should remain locked using the same **<LockId>** so that subsequent transactions can operate on those object instances. In the event that the **releaseAction** is set to **SOME**, and an expiry period was specified on the <**LockObject**> or <**GetObjectWithLock**> elements using the **expiry** attribute, the expiry counter must be reset to zero after each transaction unless all object instances in the locked set have been operated upon. The default **releaseAction** value is **ALL.**

As an example, if a client application locks 20 object instances and then submits a transaction request that only operates on 10 of those locked object instances, a **releaseAction** of **SOME** would mean that the 10 remaining unaltered object instances should remain locked when the transaction terminates. Subsequent transaction operations can then be submitted by the client application, using the same lock identifier to modify the remaining 10 object instances.

14.2.3.1 <Transaction> element

A **<Transaction>** element may contain zero or more **<Insert>**, **<Update>**, or **<Delete>** elements that describe operations to create, modify or destroy object instances. An empty **<Transaction>** request is valid but not very useful.

The optional **<LockId>** element is used to specify that the transaction will be applied to previously locked set of object instances. Clause 15 presents a full description of an object locking mechanism. If the WOS does not support object locking, then the **<LockId>** element can be ignored. If a WOS does support locking and an invalid lock identifier is specified in the transaction, then the transaction shall fail and the web object service shall report the error as described in subclause 9.8.

At the end of a transaction, the web object service shall apply transaction semantics appropriate to the particular system used to persistently store objects. For example if the datastore is a SQL based RDBMS, then a *commit* will be executed at the end of the transaction (or a *rollback* should the transaction fail). Any locks maintained by the web object service for the duration of the transaction shall be released according to the value of the **releaseAction** attribute described above.

The **<Native>** element is defined in subclause 9.5.

14.2.3.2 <Insert> element

The **<Insert>** element is used to create new object instances. The <**Insert**> element either contains an object instance encoded in XML that defines the initial state of the object instance or it may contain a reference to an object instance (specified using the <**ObjectRef**> element) to be inserted into the repository.

In addition, as described in subclause 9.4, object instances encoded in XML may include property value references using the **objRef** attribute group**.**

If the initial state of an object to be created is expressed using XML, it must validate relative to an application schema generated by the **DescribeObjectType** operation [see clause 11].

Multiple **<Insert>** elements can be enclosed in a single **Transaction** request and multiple object instances can be created using a single **<Insert>** element.

In response to an **<Insert>** operation, a web object service shall generate a list of newly generated object identifiers assigned to the new object instances. The object identifiers must be presented in the order in which the **<Insert>** operations were encountered in the **Transaction** request.

14.2.3.3 <Update> element

The **<Update>** element describes one update operation that is to be applied to an object instance or set of object instances of a single object type. Multiple **<Update>** operations can be contained in a single **Transaction** request.

An **<Update>** element contains one or more <**Property>** elements. A **<Property>** element, contains a **<Name>** element that contains the name of the object property to be modified and an optional **<Value>** or **<ValueRef>** element that contains the replacement value for the named object property. The omission of the **<Value>** or **<ValueRef>** element means that the property should be assigned a NULL value. In the event that the property is not nillable, and a NULL value is assigned to the property, a WOS **must** raise an exception.

The scope of the **<Update>** element is constrained by using the **<QueryConstraint>** element. The **<QueryConstraint>** element can be used to limit the scope of an update operation to an enumerated set of objects or a set of objects defined using spatial and non-spatial constraints.

14.2.3.4 <Delete> element

The **<Delete>** element is used to indicate that one or more object instances should be deleted. The scope of a delete operation is constrained by using the **<QueryConstraint>** element which is described in clause 9.6**.**

Example

14.3 Response message

In response to a transaction request, a web object service shall generate an XML document indicating the termination status of the transaction. In addition, if the transaction request message includes **<Insert>** elements, then the web object service must report the object identifiers of all newly created object instances. In the event that the transaction fails to execute, the web object service shall also indicate this in the response message.

The XML encoding of the WOS transaction response is defined by the following XML Schema fragment:

```
<xsd:complexType name="TransactionResponseType">
   <xsd:sequence>
     <xsd:element name="InsertResult"
                  type="wos:InsertResultType"
                  minOccurs="0" maxOccurs="unbounded"/>
     <xsd:element name="TransactionResult"
                  type="wos:TransactionResultType"/>
   </xsd:sequence>
   <xsd:attribute name="version" type="xsd:string" use="required" />
</xsd:complexType>
<xsd:complexType name="TransactionResultType">
   <xsd:sequence>
     <xsd:element name="Status" type="wos:StatusType"/>
     <xsd:element name="Locator" type="xsd:string" minOccurs="0"/>
     <xsd:element name="Message" type="xsd:string" minOccurs="0"/>
   </xsd:sequence>
   <xsd:attribute name="handle"  type="xsd:string" use="optional" />
</xsd:complexType>
<xsd:complexType name="InsertResultType">
   <xsd:sequence>
     <xsd:element ref="ogc:ObjectId" maxOccurs="unbounded"/>
   </xsd:sequence>
   <xsd:attribute name="handle"  type="xsd:string" use="optional" />
</xsd:complexType>
<xsd:complexType name="StatusType">
   <xsd:choice>
      <xsd:element ref="wos:SUCCESS"/>
      <xsd:element ref="wos:FAILED"/>
      <xsd:element ref="wos:PARTIAL"/>
   </xsd:choice>
</xsd:complexType>
<xsd:element name="SUCCESS" type="wos:EmptyType"/>
<xsd:element name="FAILED" type="wos:EmptyType"/>
<xsd:element name="PARTIAL" type="wos:EmptyType"/>
```

The **<WOS_TransactionResponse>** element contains zero or more **<InsertResult>** elements and a **<TransactionResult>** element.

The **<InsertResult>** element contains one or more object identifiers of newly created object instancess. One **<InsertResult>** element is reported per **<Insert>** element in the request. The insert results are reported in the order in which the **<Insert>** operations were contained in the **<Transaction>** element. Additionally, they can be correlated using the **handle** attribute if it was specified.

The overall result of a transaction request is specified using the **<TransactionResult>** element. The **<TransactionResult>** element must contain a **<Status>** element and may contain **<Locator>** and **<Message>** elements.

The **<Status>** element is used to indicate the completion status of the transaction. Transactions can terminate with a status of:

SUCCESS	The transaction was successfully completed.
FAILED	An exception was encountered while processing one or more elements contained in a **Transaction** request.
PARTIAL	The transaction partially succeeded and the data may be in an inconsistent state. For systems that do not support atomic transactions, this outcome is a distinct possibility.

In the event that a transaction request fails, the **<Locator>** element can be used to indicate which part of the transaction failed. If the element at which the failure occured is labeled using a **handle** attribute then a web object service may report its value to locate the failure. Otherwise, a web object service may try to identify the failure relative to the beginning of the transaction request possibly using line numbers or some other convenient mechanism.

The **<Message>** element is used to report any error messages.

Example

Consider a transaction request (labeled "TX01") that creates a number of new object instances. The object instances are created using three **<Insert>** elements labeled "STMT1", "STMT2" and "STMT3". A typical response to such a request might be:

```
<?xml version="1.0" ?>
<wos:TransactionResponse
   version="0.0.1"
   xmlns:wos="http://www.opengis.net/WOS"
   xmlns:ogc="http://www.opengis.net/ogc"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://www.opengis.net/WOS ../WOS/0.0.1/WOS-transaction.xsd">
   <wos:InsertResult handle="STMT1">
      <ogc:ObjectId oid="http://www.someserver.com/SOMEOBJECT.4567"/>
      <ogc:ObjectId oid="http://www.someserver.com/SOMEOBJECT.4568"/>
      <ogc:ObjectId oid="http://www.someserver.com/SOMEOBJECT.4569"/>
   </wos:InsertResult>
   <wos:InsertResult handle="STMT2">
      <ogc:ObjectId oid="http://www.simeserver.com/OBJECT1.4569"/>
   </wos:InsertResult>
   <wos:InsertResult handle="STMT3">
      <ogc:ObjectId oid="http://www.someserver.com/OBJECT2.389345"/>
   </wos:InsertResult>
   <wos:TransactionResult handle="TX01">
      <wos:Status>
         <wos:SUCCESS/>
      </wos:Status>
   </wos:TransactionResult>
</wos:TransactionResponse>
```

14.4 Exception message

In the event that a web object service encounters an error servicing a **Transaction** request message, it shall raise an exception as described in clause 9.8.

In the event that a web object service encounters an error while processing a particular element contained in a **Transaction** request, the web object service shall queue the result and report the failure in a **<WOS_TransactionResponse>** element [sec. 11.3].

Example

In this example, the second statement of a **Transaction** request has failed. The WOS might generate a response such as:

```
<?xml version="1.0" ?>
<WOS_TransactionResponse
   version="0.0.1"
   xmlns:wos="http://www.opengis.net/WOS"
   xmlns:ogc="http://www.opengis.net/ogc"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://www.opengis.net/WOS ../WOS/0.0.1/WOS-transaction.xsd">

   <TransactionResult handle="TX01">
      <Status><FAILED/></Status>
      <Locator>STMT2</Locator>
      <Message>ORA-00942: table or view does not exist</Message>
   </TransactionResult>
</WOS_TransactionResponse>
```

14.5 Examples

Example 1

This example inserts a style into a repository.

```
KVP:
http://www.someserver.com/sms.cgi?service=WOS&version=0.0.2&
   request=Transaction&operation=Insert&
   objectname=FeatureStyle&object=<FeatureStyle
      version="0.7.3"
   xsi:schemaLocation="http://www.opengis.net/sld
      FeatureStyle.xsd"
   xmlns="http://www.opengis.net/sld"
      xmlns:ogc="http://www.opengis.net/ogc"
   xmlns:xlink="http://www.w3.org/1999/xlink"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

      <FeatureTypeName>OCEANSEA_1M:Foundation</FeatureTypeName
      ><Rule>

      <Name>main</Name><PolygonSymbol><Geometry><ogc:PropertyN
      ame>
   GEOMETRY</ogc:PropertyName></Geometry><Fill><SvgParameter
      name="fill">

      #96C3F5</SvgParameter></Fill></PolygonSymbol></Rule><wos
      :FeatureStyle>

XML:
<wos:Transaction version="0.0.2" service="WOS">
   <wos:Insert>
      <FeatureStyle version="0.7.3"
```

```
      xsi:schemaLocation="http://www.opengis.net/sld
      FeatureStyle.xsd"
                    xmlns="http://www.opengis.net/sld"
                    xmlns:ogc="http://www.opengis.net/ogc"
                    xmlns:xlink="http://www.w3.org/1999/xlink"

      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

      <FeatureTypeName>OCEANSEA_1M:Foundation</FeatureTypeName
      >
          <Rule>
             <Name>main</Name>
             <PolygonSymbol>
                <Geometry>

      <ogc:PropertyName>GEOMETRY</ogc:PropertyName>
                </Geometry>
                <Fill>
                   <SvgParameter
      name="fill">#96C3F5</SvgParameter>
                </Fill>
             </PolygonSymbol>
          </Rule>
       <wos:FeatureStyle>
    </wos:Insert>
</wos:Transaction>
```

Response:

```
<?xml version="1.0" ?>
<wos:TransactionResponse
   version="0.0.2"
   xmlns:wos="http://www.opengis.net/WOS"
   xmlns:ogc="http://www.opengis.net/ogc"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://www.opengis.net/WOS ../WOS/0.0.1/WOS-transaction.xsd">
   <wos:InsertResult>
      <ogc:ObjectId oid="http://www.someserver.com/sms.cgi?odi=4567"/>
   </wos:InsertResult>
   <wos:TransactionResult>
      <wos:Status>
         <wos:SUCCESS/>
      </wos:Status>
   </wos:TransactionResult>
</wos:TransactionResponse>
```

Example 2

The following example deletes the style inserted in example 1.

KVP:

```
http://www.someserver.com/sms.cgi?service=WOS&version=0.0.2&re
      quest=Transaction&

      operation=Delete&objected="http://www.someserver.com/sms
      .cgi?oid=4567"
```

XML:

```
<?xml version="1.0"?>
<wos:Transaction service="WOS" version="0.0.2">
   <wos:Delete>
```

```
      <ogc:Filter>
         <ObjectId
        oid="http://www.someserver.com/sms.cgi?oid=4567" />
      </ogc:Filter>
   </wos:Delete>
</wos:Transaction>

Response:
<?xml version="1.0" ?>
<wos:TransactionResponse
   version="0.0.1"
   xmlns:wos="http://www.opengis.net/WOS"
   xmlns:ogc="http://www.opengis.net/ogc"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://www.opengis.net/WOS
      ../WOS/0.0.1/WOS-transaction.xsd">
   <wos:TransactionResult>
      <wos:Status>
         <wos:SUCCESS/>
      </wos:Status>
   </wos:TransactionResult>
</wos:TransactionResponse>
```

Example 3

This XML encoded example inserts multiple related resources into an image archive. There are 9 bands, and an HDF file. The example uses the following files:

```
L71014032_B10.L1G
Band 1
L71014032_B20.L1G
Band 2
L71014032_B30.L1G
Band 3
L71014032_B40.L1G
Band 4
L71014032_B50.L1G
Band 5
L71014032_B61.L1G
Band 6
L72014032_B62.L1G
Band 7
L72014032_B70.L1G
Band 8
L72014032_B80.L1G
Band 9
L71014032_HDF.L1G
```

The following multipart MIME document inserts the parts into an image archive:

```
Content-type: multipart/mixed; boundary="wos12345"

-- wos12345
Content-ID: Insert-Transaction
Content-Type: text/xml

<?xml version="1.0" ?>
```

```
<wos:Transaction>
   <wos:Insert handle="L71014032_B10">
      <LandsatImage>
         <GroupId>L71014032</GroupId>
         <Band>1</Band>
         <Image wos:mimeType="image/HDF_EOS_PART"
     wos:href="L71014032_B10.L1G" />
      <LandsatImage>
   </wos:Insert>
   <wos:Insert handle="L71014032_B20">
      <LandsatImage>
         <GroupId>L71014032</GroupId>
         <Band>2</Band>
         <Image wos:mimeType="image/HDF_EOS_PART"
     wos:href="L71014032_B20.L1G" />
      <LandsatImage>
   </wos:Insert>
   <wos:Insert handle="L71014032_B30">
      <LandsatImage>
         <Groupd>L71014032</GroupId>
         <Band>3</Band>
         <Image wos:mimeType="image/HDF_EOS_PART"
     wos:href="L71014032_B30.L1G" />
      <LandsatImage>
   </wos:Insert>
   <wos:Insert handle="L71014032_B40">
      <LandsatImage>
         <GroupId>L71014032</GroupId>
         <Band>4</Band>
         <Image wos:mimeType="image/HDF_EOS_PART"
     wos:href="L71014032_B40.L1G" />
      <LandsatImage>
   </wos:Insert>
   <wos:Insert handle="L71014032_B50">
      <LandsatImage>
         <GroupId>L71014032</GroupId>
         <Band>5</Band>
         <Image wos:mimeType="image/HDF_EOS_PART"
     wos:href="L71014032_B50.L1G" />
      <LandsatImage>
   </wos:Insert>
   <wos:Insert handle="L71014032_B61">
      <LandsatImage>
         <GroupId>L71014032</GroupId>
         <Band>61</Band>
         <Image wos:mimeType="image/HDF_EOS_PART"
     wos:href="L71014032_B61.L1G" />
      <LandsatImage>
   </wos:Insert>
   <wos:Insert handle="L71014032_B62">
      <LandsatImage>
         <GroupId>L71014032</GroupId>
         <Band>62</Band>
         <Image wos:mimeType="image/HDF_EOS_PART"
     wos:href="L71014032_B62.L1G" />
      <LandsatImage>
   </wos:Insert>
   <wos:Insert handle="L71014032_B70">
```

```
      <LandsatImage>
         <GroupId>L71014032</GroupId>
         <Band>7</Band>
         <Image wos:mimeType="image/HDF_EOS_PART"
      wos:href="L71014032_B70.L1G" />
      <LandsatImage>
   </wos:Insert>
   <wos:Insert handle="L71014032_B80">
      <LandsatImage>
         <GroupId>L71014032</GroupId>
         <Band>8</Band>
         <Image wos:mimeType="image/HDF_EOS_PART"
      wos:href="L71014032_B80.L1G" />
      <LandsatImage>
   </wos:Insert>
   <wos:Insert handle="L71014032_HDF">
      <LandsatImage>
         <GroupId>L71014032</GroupId>
         <Band>HDF</Band>
         <Image wos:mimeType="image/HDF_EOS_PART"
      wos:href="L71014032_HDF.L1G" />
      <LandsatImage>
   </wos:Insert>
</wos:Transaction>
--wos12345
Content-ID: L71014032_B10.L1G
Content-Type: image/HDF_EOS_PART
Content-Length: 99999

... 99999 bytes of image data ...
--wos12345
Content-ID: L71014032_B20.L1G
Content-Type: image/HDF_EOS_PART
Content-Length: 99999

... 99999 bytes of image data ...
--wos12345
Content-ID: L71014032_B30.L1G
Content-Type: image/HDF_EOS_PART
Content-Length: 99999

... 99999 bytes of image data ...
--wos12345
Content-ID: L71014032_B40.L1G
Content-Type: image/HDF_EOS_PART
Content-Length: 99999

... 99999 bytes of image data ...
--wos12345
Content-ID: L71014032_B50.L1G
Content-Type: image/HDF_EOS_PART
Content-Length: 99999

... 99999 bytes of image data ...
--wos12345
Content-ID: L71014032_B61.L1G
Content-Type: image/HDF_EOS_PART
Content-Length: 99999
```

```
... 99999 bytes of image data ...
--wos12345
Content-ID: L71014032_B62.L1G
Content-Type: image/HDF_EOS_PART

... 99999 bytes of image data ...
--wos12345
Content-ID: L71014032_B70.L1G
Content-Type: image/HDF_EOS_PART

... 99999 bytes of image data ...
--wos12345
Content-ID: L71014032_B80.L1G
Content-Type: image/HDF_EOS_PART
Content-Length: 99999

... 99999 bytes of image data ...
--wos12345
Content-ID: L71014032_HDF.L1G
Content-Type: image
Content-Length: 99999

   ... 99999 bytes of image data ...
--wos12345
```

Response:

```
<?xml version="1.0" ?>
<wos:TransactionResponse
   version="0.0.2"
   xmlns:wos="http://www.opengis.net/WOS"
   xmlns:ogc="http://www.opengis.net/ogc"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://www.opengis.net/WOS
      ../WOS/0.0.1/WOS-transaction.xsd">
   <wos:InsertResult>
      <ogc:ObjectId handle="L71014032_B10"

     oid="http://www.someserver.com/sms.cgi?odi=4567"/>
      <ogc:ObjectId handle="L71014032_B20"

     oid="http://www.someserver.com/sms.cgi?odi=4568"/>
      <ogc:ObjectId handle="L71014032_B30"

     oid="http://www.someserver.com/sms.cgi?odi=4569"/>
      <ogc:ObjectId handle="L71014032_B40"

     oid="http://www.someserver.com/sms.cgi?odi=4570"/>
      <ogc:ObjectId handle="L71014032_B50"

     oid="http://www.someserver.com/sms.cgi?odi=4571"/>
      <ogc:ObjectId handle="L71014032_B61"

     oid="http://www.someserver.com/sms.cgi?odi=4572"/>
      <ogc:ObjectId handle="L71014032_B62"

     oid="http://www.someserver.com/sms.cgi?odi=4573"/>
      <ogc:ObjectId handle="L71014032_B70"
```

```
        oid="http://www.someserver.com/sms.cgi?odi=4574"/>
         <ogc:ObjectId handle="L71014032_B80"

        oid="http://www.someserver.com/sms.cgi?odi=4575"/>
         <ogc:ObjectId handle="L71014032_HDF"

        oid="http://www.someserver.com/sms.cgi?odi=4576"/>
    </wos:InsertResult>
    <wos:TransactionResult>
       <wos:Status>
          <wos:SUCCESS/>
       </wos:Status>
    </wos:TransactionResult>
</wos:TransactionResponse>
```

Example 4

This sample HTTP POST, shows the same transaction used in Example 1 but packaged into an OMF message. The example maps the OMF schema to SOAP with attachements using HTTP and multipart MIME.

```
POST /servlet/handler HTTP/1.1
Host: www.example.com
soapAction: "omf"
Content-type: multipart/related; MimeBoundary="wos12345";
      type="text/xml";

start=message_header
-- wos12345
Content-ID: message_header
Content-Type: text/xml

<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope xmlns:xlink="http://www.w3.org/1999/xlink"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
     xmlns:omf="http://www.opengis.net/schema/omf.xsd"

      xsi:schemaLocation="http://schemas.xmlsoap.org/soap/enve
      lope/
            http://www.opengis.net/schema/omf.xsd">
   <soap:Header>
      <omf:Header omf:version="1.0">
         <omf:From>
            <omf:PartyId>urn:duns:123456789</omf:PartyId>
         </omf:From>
         <omf:To>
            <omf:PartyId>urn:duns:912345678</omf:PartyId>
         </omf:To>
         </omf:Action>
            <omf:Service>urn:ogc:services:wcs</omf:Service>
            <omf:Process>Transaction</omf:Process>
         </omf:Action>
         <omf:MessageId>20001209-133003-
      28572@example.com</omf:MessageId>
         <omf:Timestamp>2001-02-15T11:12:12</omf:Timestamp>
```

```
      </omf:Header>
   </soap:Header>
   <soap:Body>
      <omf:Manifest>
         <omf:Reference omf:id="item00"
                        xlink:href="Insert-Transaction"
                        xlink:role="WOS Request Message"
     xlink:type="simple">
         </omf:Reference>
         <omf:Reference omf:id="item01"
                        xlink:href="L71014032_B10.L1G"
                        xlink:role="Data" xlink:type="simple">
         </omf:Reference>
         <omf:Reference omf:id="item02"
                        xlink:href="L71014032_B20.L1G"
                        xlink:role="Data" xlink:type="simple">
         </omf:Reference>
         <omf:Reference omf:id="item03"
                        xlink:href="L71014032_B30.L1G"
                        xlink:role="Data" xlink:type="simple">
         </omf:Reference>
         <omf:Reference omf:id="item04"
                        xlink:href="L71014032_B40.L1G"
                        xlink:role="Data" xlink:type="simple">
         </omf:Reference>
         <omf:Reference omf:id="item05"
                        xlink:href="L71014032_B50.L1G"
                        xlink:role="Data" xlink:type="simple">
         </omf:Reference>
         <omf:Reference omf:id="item06"
                        xlink:href="L71014032_B61.L1G"
                        xlink:role="Data" xlink:type="simple">
         </omf:Reference>
         <omf:Reference omf:id="item07"
                        xlink:href="L71014032_B62.L1G"
                        xlink:role="Data" xlink:type="simple">
         </omf:Reference>
         <omf:Reference omf:id="item08"
                        xlink:href="L71014032_B70.L1G"
                        xlink:role="Data" xlink:type="simple">
         </omf:Reference>
         <omf:Reference omf:id="item09"
                        xlink:href="L71014032_B80.L1G"
                        xlink:role="Data" xlink:type="simple">
         </omf:Reference>
         <omf:Reference omf:id="item10"
                        xlink:href="L71014032_HDF.L1G"
                        xlink:role="Data" xlink:type="simple">
         </omf:Reference>
         <omf:Reference omf:id="item11"
                        xlink:href="L71014032_MTL.L1G"
                        xlink:role="Data" xlink:type="simple">
            <omf:Description xml:lang="en-US">Native Landsat
                         metadata</omf:Description>
         </omf:Reference>
      </omf:Manifest>
   </soap:Body>
</soap:Envelope>
```

```
-- wos12345
Content-ID: Insert-Transaction
Content-Type: text/xml

<?xml version="1.0" ?>
<wos:Transaction>
   <wos:Insert>
      <LandsatImage>
         <GroupId>L71014032</GroupId>
         <Band>1</Band>
         <Image wos:mimeType="image/HDF_EOS_PART"
      wos:href="L71014032_B10.L1G" />
      <LandsatImage>
      <LandsatImage>
         <GroupId>L71014032</GroupId>
         <Band>2</Band>
         <Image wos:mimeType="image/HDF_EOS_PART"
      wos:href="L71014032_B20.L1G" />
      <LandsatImage>
      <LandsatImage>
         <Groupd>L71014032</GroupId>
         <Band>3</Band>
         <Image wos:mimeType="image/HDF_EOS_PART"
      wos:href="L71014032_B30.L1G" />
      <LandsatImage>
      <LandsatImage>
         <GroupId>L71014032</GroupId>
         <Band>4</Band>
         <Image wos:mimeType="image/HDF_EOS_PART"
      wos:href="L71014032_B40.L1G" />
      <LandsatImage>
      <LandsatImage>
         <GroupId>L71014032</GroupId>
         <Band>5</Band>
         <Image wos:mimeType="image/HDF_EOS_PART"
      wos:href="L71014032_B50.L1G" />
      <LandsatImage>
      <LandsatImage>
         <GroupId>L71014032</GroupId>
         <Band>61</Band>
         <Image wos:mimeType="image/HDF_EOS_PART"
      wos:href="L71014032_B61.L1G" />
      <LandsatImage>
      <LandsatImage>
         <GroupId>L71014032</GroupId>
         <Band>62</Band>
         <Image wos:mimeType="image/HDF_EOS_PART"
      wos:href="L71014032_B62.L1G" />
      <LandsatImage>
      <LandsatImage>
         <GroupId>L71014032</GroupId>
         <Band>7</Band>
         <Image wos:mimeType="image/HDF_EOS_PART"
      wos:href="L71014032_B70.L1G" />
      <LandsatImage>
      <LandsatImage>
         <GroupId>L71014032</GroupId>
```

```
            <Band>8</Band>
            <Image wos:mimeType="image/HDF_EOS_PART"
         wos:href="L71014032_B80.L1G" />
          <LandsatImage>
          <LandsatImage>
            <GroupId>L71014032</GroupId>
            <Band>HDF</Band>
            <Image wos:mimeType="image/HDF_EOS_PART"
         wos:href="L71014032_HDF.L1G" />
          <LandsatImage>
      </wos:Insert>
</wos:Transaction>
--wos12345
Content-ID: L71014032_B10.L1G
Content-Type: image/HDF_EOS_PART
Content-Length: 99999

... Binary image data not shown ...
--wos12345
Content-ID: L71014032_B20.L1G
Content-Type: image/HDF_EOS_PART
Content-Length: 99999

... Binary image data not shown ...
--wos12345
Content-ID: L71014032_B30.L1G
Content-Type: image/HDF_EOS_PART
Content-Length: 99999

... Binary image data not shown ...
--wos12345
Content-ID: L71014032_B40.L1G
Content-Type: image/HDF_EOS_PART
Content-Length: 99999

... Binary image data not shown ...
--wos12345
Content-ID: L71014032_B50.L1G
Content-Type: image/HDF_EOS_PART
Content-Length: 99999

... Binary image data not shown ...
--wos12345
Content-ID: L71014032_B61.L1G
Content-Type: image/HDF_EOS_PART
Content-Length: 99999

... Binary image data not shown ...
--wos12345
Content-ID: L71014032_B62.L1G
Content-Type: image/HDF_EOS_PART

... Binary image data not shown ...
--wos12345
Content-ID: L71014032_B70.L1G
Content-Type: image/HDF_EOS_PART

... Binary image data not shown ...
```

```
--wos12345
Content-ID: L71014032_B80.L1G
Content-Type: image/HDF_EOS_PART
Content-Length: 99999

... Binary image data not shown ...
--wos12345
Content-ID: L71014032_HDF.L1G
Content-Type: image
Content-Length: 99999

   ... Binary image data not shown ...
--wos12345
```

The response would be similar to that shown in example 3.

15 LockObject operation

15.1 Introduction

Web connections are inherently stateless. As a consequence of this, the semantics of serializable transactions are not preserved. To understand the issue consider an UPDATE operation.

The client fetches an object instance. The object is then modified on the client side, and submitted back to the database, via a **Transaction** request for update. Serializability is lost since there is nothing to guarantee that while the object was being modified on the client side, another client did not come along and update that same object in the database.

One way to ensure serializability is to require that changes to data be done in a mutually exclusive manner; that is while one transaction is changing a data item, no other transaction can modify the same data item. This can be accomplished by using locks that control write access to the data.

The purpose of the **LockObject** operation is to expose *a long term object locking* mechanism to ensure data consistency. The lock is considered long term because network latency would make object locks last relatively longer than native commercial database locks.

Locking an object instance blocks all **Transaction**, **GetObjectWithLock**, and **LockObject** operations on that object except for those operations issued by the lock owner. The **GetCapabilities**, **DescribeObjectType**, **GetObjectById** and **GetObject** operations are not affected by locking.

The **LockObject** operation is optional and does not need to be implemented for a WOS implementation to conform to this specification. If a WOS implements the **LockObject** operation, this fact must be advertised in the capabilities document [sec. 12].

15.2 Request

15.2.1 Keyword-value pair encoding

Table 6 – LockObject encoding

URL Component	O/M	DEFAULT	Description
REQUEST=LockObject	M		The name of the request.
OBJECTNAME (**Optional** if OBJECTID is specified.)	M		Names or one or more object types whose object instances are to be locked.
EXPIRY	O		The number of minutes the lock should persist before being cleared. If no value is specified then the lock should persist indefinitely.
LOCKACTION=[ALL | SOME]	O		Specify how the lock should be acquired. ALL indicates to try to get all object locks otherwise fail. SOME indicates to try to get as many object locks as possible.
OBJECTID (**Mutually exclusive** with FILTER and BBOX.)	O		An enumerated list of object instance identifiers indicating which object instances to lock.
FILTER (**Prerequisite**:OBJECTNAME) (**Mutually exclusive** with OBJECTID and BBOX)	O		A FILTER SPECIFICATION DESCRIBES A SET OF OBJECTS TO OPERATE UPON. THE PREDICATE LANGUAGE THAT IS BEING USED IS SPECIFIED USING THE FILTERLANGUAGE PARAMETER. POSSIBLE VALUES INCLUDE OGCFILTER, CQLTEXT, SFSQL AND SQLMM. IF THE FILTER PARAMETER IS USED, ONE FILTER MUST BE SPECIFIED FOR EACH OBJECT TYPE LISTED IN THE OBJECTNAME PARAMETER. INDIVIDUAL FILTERS ENCODED IN THE FILTER PARAMETER ARE ENCLOSED IN PARENTHESES “(“ AND “)”.
BBOX (**Prerequisite**: OBJECTNAME) (**Mutually exclusive** with OBJECTID and FILTER.)	O		IN LIEU OF AN OBJECTID OR FILTER, A CLIENT MAY SPECIFY A BOUNDING BOX AS DESCRIBED IN CLAUSE 8.5.2.3.3.

15.2.2

15.2.3 XML encoding

```
From WOS-transaction.xsd:
<xsd:element name="LockObject"
             type="wos:LockObjectType"/>
<xsd:element name="LockObjectResponse"
             type="wos:LockObjectResponseType"/>

From wos.xsd:
<xsd:complexType name="LockObjectType">
   <xsd:sequence>
      <xsd:element name="Expiry"
                   type="xsd:positiveInteger" minOccurs="0"/>
      <xsd:element name="Lock"
                   type="wos:LockType" maxOccurs="unbounded"/>
   </xsd:sequence>
   <xsd:attribute name="version" type="xsd:string" use="required" />
   <xsd:attribute name="service" type="xsd:string" use="required" />
   <xsd:attribute name="handle"  type="xsd:string" use="optional" />
</xsd:complexType>
<xsd:complexType name="LockType">
   <xsd:sequence>
      <xsd:element name="QueryConstraint"
                   type="QueryConstraintType"
                   minOccurs="0" maxOccurs="1"/>
   </xsd:sequence>
   <xsd:attribute name="handle" type="string" use="optional" />
   <xsd:attribute name="objectName" type="xsd:QName" use="required" />
   <xsd:attribute name="lockAction" type="wos:AllSomeType" use="optional"/>
</xsd:complexType>
```

15.2.4 Description

A **LockObject** operation may define one or more lock operations that attempt to lock instances of the specified object.

The **expiry** attribute is used to set a limit on how long a web object service should hold a lock on object instances in the event that a transaction is never issued that will release the lock. The *expiry* limit is specified in minutes. Once the specified number of minutes have elapsed a web object service may release the lock if it exists. Any further transactions issued against that lock using a lock identifier generated by the service will fail. This specification does not constrain how long a lock should be held if the **expiry** attribute is not specified. However, it would be prudent for a web object service implementation to include methods to detect and release locks that have been maintained for a long period of time without any transactions being executed to release them.

A **<QueryConstraint>** element is used to define query constraints that identify the set of object instances of the specified object to be locked.

The optional **lockAction** parameter is used to control how object locks are acquired. A lock action of **ALL** indicates that a web object service should try to acquire a lock on all requested object instances; if all object instances cannot be locked then the operation should fail and no object instances should remain locked. If the lock action is set to **SOME**, then a web object service shall attempt to lock as many of the requested object instances as it can. The default lock action shall be **ALL**. Clause 15.2.5 presents a state machine for the **LockObject** operation.

15.2.5 State machine notation from UML

The approach to dynamic modeling uses that given by the UML Reference Manual. The main technique is the state machine view. A summary of the UML notation for state diagrams is shown in Figure 4.

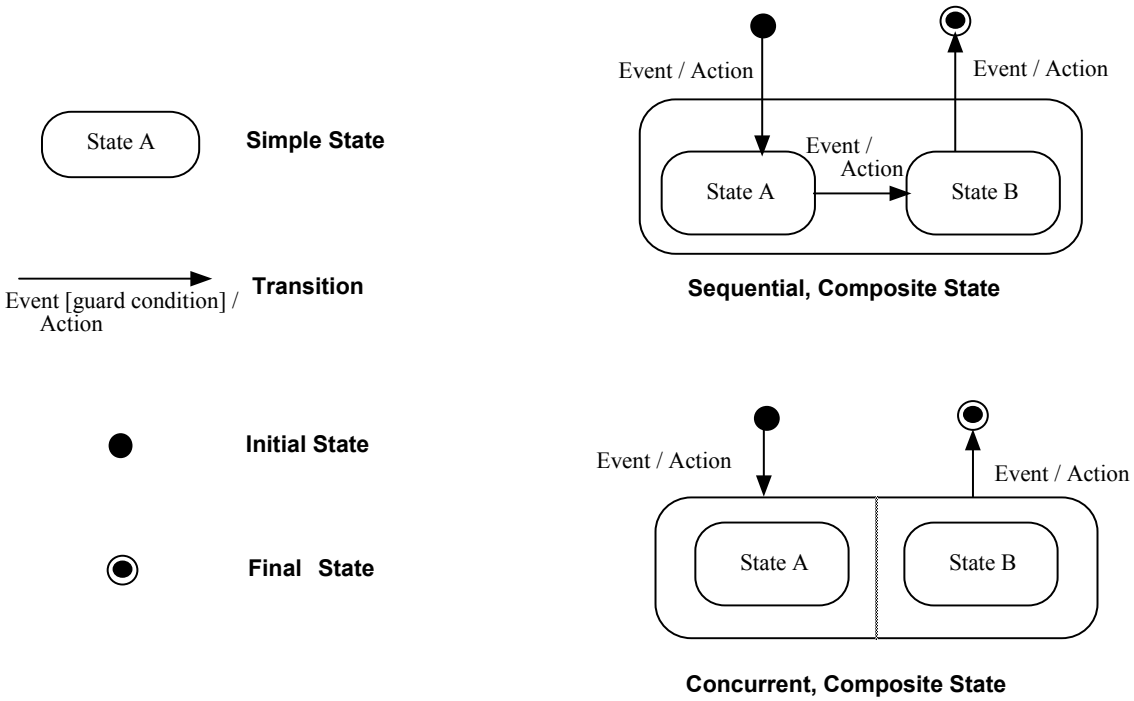

Figure 4 – Summary of UML state diagram notation

15.2.6 State machine for WOS locking

This section defines the state machine for the Lock State for a server that provides the Web Object Service interface. The state diagram shows the allowed transitions between the states. All other state transitions are disallowed and are consider errors if exhibited by a server.

A physical server may support more than one lock. Each of the locks are independent when viewed from the service defined by the WOS specification.

In the state model below, a transition is typically triggered by a request. Following the messaging model, a WOS operation is paired with a WOS Response. Note that a request-response pair cannot be started while it is active. The request may be cancelled by a HTTP-level command.

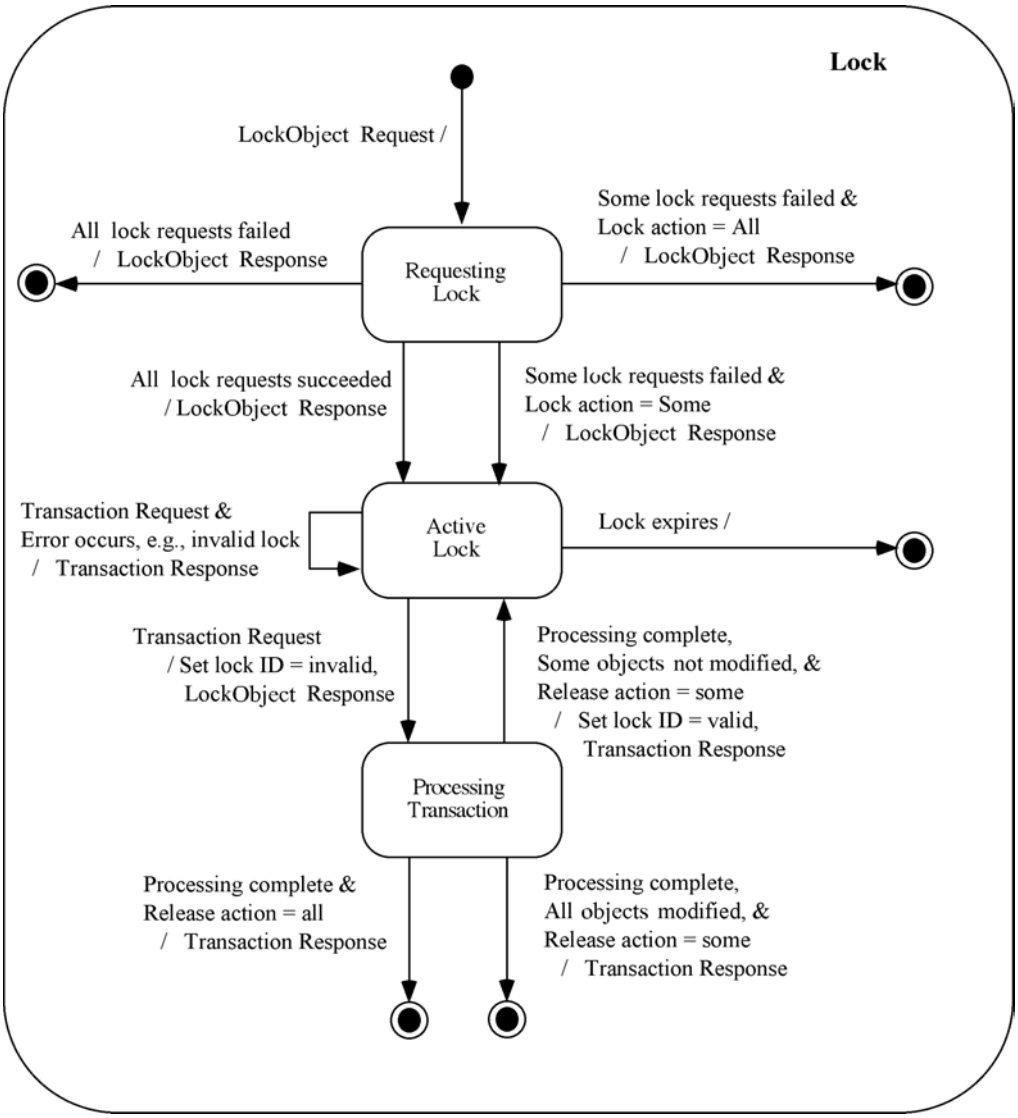

Figure 5 –State diagram for a WOS lock

15.3 Response

The XML encoding of the response to a **LockObject** request is defined by the following XML Schema fragment:

```
From wos.xsd:
<xsd:complexType name="LockObjectResponseType">
  <xsd:sequence>
    <xsd:element ref="wos:LockId"/>
    <xsd:element name="ObjectsLocked"
                 type="wos:ObjectsLockedType" minOccurs="0"/>
    <xsd:element name="ObjectsNotLocked"
                 type="wos:ObjectsNotLockedType" minOccurs="0"/>
  </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="ObjectsLockedType">
  <xsd:sequence maxOccurs="unbounded">
    <xsd:element ref="ogc:ObjectId"/>
  </xsd:sequence>
</xsd:complexType>
```

```
<xsd:complexType name="ObjectsNotLockedType">
  <xsd:sequence maxOccurs="unbounded">
    <xsd:element ref="ogc:ObjectId"/>
  </xsd:sequence>
</xsd:complexType>
```

In response to an keyword-value pair encoded or XML encoded **LockObject** request, a web object service shall generate an XML document containing a lock identifier that a client application can use in subsequent WOS operations to operate upon the set of locked object instances. The response can also contain the optional element <**ObjectsLocked**> and <**ObjectsNotLocked**> depending on the value of the **lockAction** attribute.

If the lock action is specified as **SOME**, then the <**WOS_LockObjectResponse**> element must contain the <**ObjectsLocked**> and <**ObjectNotLocked**> elements.. The **<ObjectsLocked>** element shall list the object identifiers of all the object instances that were locked by the **LockObject** request. The **<ObjectsNotLocked>** element shall contain a list of object identifiers for the object instances that could not be locked by the web object service (possibly because they were already locked by someone else).

No assumption is made about the format of the lock identifier. The only requirement is that it can be expressed in the character set of the transaction request.

15.4 Exceptions

If a WOS does not implement the **LockObject** operation then it should generate an exception, indicating that the operation is not supported, if such a request is encountered.

In the event that a web object service does support the **LockObject** operation and encounters an error servicing the request, it shall raise an exception as described in clause 9.8.

15.5 Examples

Example 1

Lock a set of enumerated objects. The WOS is, in this case, directed to try and lock as many objects as it can.

```
KVP:
http://www.someserver.com/wos.cgi?service=WOS&version=0.0.2&request=LockObject&
   objectid= http://www.someserver.com/wos.cgi?objectid=S1013,
             http://www.someserver.com/wos.cgi?objectid=S1014,
             http://www.someserver.com/ows.cgi?objectid=S1015,
             http://www.someserver.com/ows.cgi?objectid=S1016,
             http://www.someserver.com/wos.cgi?objectid=S1017

XML:
<?xml version="1.0" ?>
<LockObject
   version="0.0.1"
   service="WOS"
   lockAction="SOME"
   xmlns="http://www.opengis.net/WOS"
   xmlns:myns="http://www.cubewerx.com/myns"
   xmlns:ogc="http://www.opengis.net/ogc"
```

```
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://www.opengis.net/WOS ../WOS/0.0.1/WOS-transaction.xsd">
   <Lock objectName="myns:SYMBOLS">
      <ogc:Filter>
         <ogc:ObjectId oid="http://www.someserver.com/wos.cgi?objectid=S1013"/>
         <ogc:ObjectId oid="http://www.someserver.com/wos.cgi?objectid=S1014"/>
         <ogc:ObjectId oid="http://www.someserver.com/ows.cgi?objectid=S1015"/>
         <ogc:ObjectId oid="http://www.someserver.com/ows.cgi?objectid=S1016"/>
         <ogc:ObjectId oid="http://www.someserver.com/wos.cgi?objectid=S1017"/>
      </ogc:Filter>
   </Lock>
</LockObject>

Response:
<?xml version="1.0" ?>
<WOS_LockObjectResponse
   xmlns="http://www.opengis.net/WOS"
   xmlns:ogc="http://www.opengis.net/ogc"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://www.opengis.net/WOS ../WOS/0.0.1/WOS-transaction.xsd">
   <LockId>1</LockId>
   <ObjectsLocked>
      <ogc:ObjectId oid="http://www.someserver.com/wos.cgi?objectid=S1013"/>
      <ogc:ObjectId oid="http://www.someserver.com/wos.cgi?objectid=S1014"/>
      <ogc:ObjectId oid="http://www.someserver.com/ows.cgi?objectid=S1016"/>
      <ogc:ObjectId oid="http://www.someserver.com/wos.cgi?objectid=S1017"/>
   </ObjectsLocked>
   <ObjectsNotLocked>
      <ogc:ObjectId oid="http://www.someserver.com/ows.cgi?objectid=S1015"/>
   </ObjectsNotLocked>
</WOS_LockObjectResponse>
```

Example 2

Lock all the object instances of the STYLES object.

```
KVP:
http://www.someserver.com/wos.cgi?service=WOS&version=0.0.2&request=LockObject&
   objectname=STYLES

XML:
<?xml version="1.0" ?>
<LockObject
   version="0.0.1"
   service="WOS"
   xmlns="http://www.opengis.net/WOS"
   xmlns:ogc="http://www.opengis.net/ogc"
   xmlns:myns="http://www.cubewerx.com/myns"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://www.opengis.net/WOS ../WOS/0.0.1/WOS-transaction.xsd">
   <Lock objectName="myns:STYLES"/>
</LockObject>

Response:
<?xml version="1.0" ?>
<WOS_LockObjectResponse
   xmlns="http://www.opengis.net/WOS"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://www.opengis.net/WOS ../WOS/0.0.1/WOS-transaction.xsd">
   <LockId>2</LockId>
</WOS_LockObjectResponse>
```

16 Interoperability Report

16.1 Implementations

Table 7 lists the organizations that implemented Web Object Servers during the OWS1.2 testbed.

Table 7 – Web Object Service Implementations

Organization	Server URL	Description
Polexis Inc.	http://ogc.polexis.com/regrep/repository	Presently only supports Transaction/Insert, KVP-encoded Transaction/Delete, GetObjectByID, and GetCapabilities. This repository places no restrictions on submitted content.
CAST	http://kirk.cast.uark.edu:9080/wos	Version 0.0.2 Supports Transaction/Insert (KVP and XMLPOST), Transaction/Delete (KVP and XMLPOST), DescribeObjectType? (KVP), and GetObjectByID?.
CubeWerx Inc.	http://demo.cubewerx.com/ows12/wos/cwwos.cgi	The CubeWerx WOS supports the GetCapabilities, DescribeObjectType, GetObjectById, GetObject and Transaction operation for Styles and Symbols in support of the SMS. In addition, the server can act as a "dumb" XML document repository.

16.2 Technology Integration Experiments

The following technology integration experiment reports involving Web Object Servers were generated during the OWS1.2 testbed:

1. The CAST repository client (http://kirk.cast.uark.edu:9080/sms) successfully connected to the CubeWerx Styles and Symbols WOS repository.

2. The Polexis integrated client was able to store and retrieve SLD document to and from the Polexis web object service.

16.3 OGC Messaging Framework

The OGC Messaging Framework is fully described in the document titled "DIRP: OGC Messaging Framework" [23]. Essentially, the OMF is a standard framework for encoding messages that are passed between clients and servers. Although no formal effort was made to harmonize the WOS with the OMF, the two specification are quite compatible and complimentary. Example 4 in section 14.5 illustrates how the OMF may be used to encode a WOS transaction request that references a number of payloads to be inserted into the repository.

An OMF module was implemented for the CubeWerx WOS and several technology integration experiments were attempted between CubeWerx and PCI. However, due to time constraints, a completely successful T.I.E. was not achived.

Figure 6 illustrates the relationship between the a client application, an OMF processing module and a WOS. OMF compliant messages, generated at the client, are passed to the OMF module (bubble 1). The OMF module processes the message and generates a WOS request that is passed to the WOS (bubble 2) for processing. The WOS processes the request and generates a standard WOS response which is passed back to the OMF module (bubble 4). The OMF module refactors the WOS response message into an OMF compliant message and sends it back to the client application.

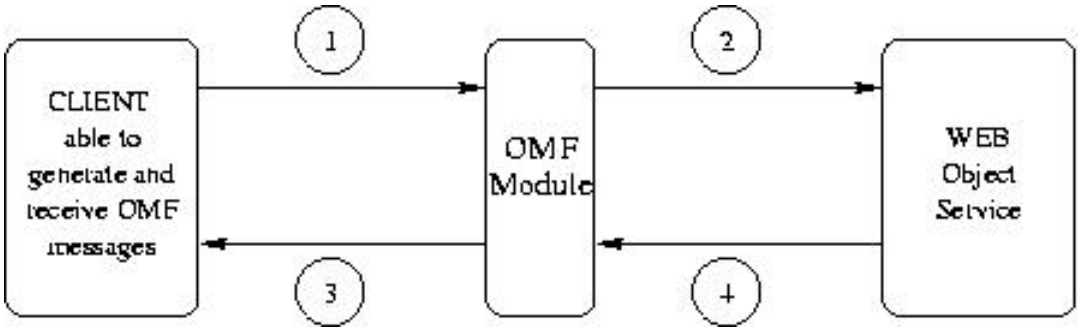

Figure 6 – Processing OMF messages in the CubeWerx WOS

It should be noted that although the OMF processing module is shown as a separate entity in figure 6, in the CubeWerx implementation it is in fact an integral part of the WOS. The CubeWerx WOS inspects the incoming messages and classifies them based on the root element. It it recognizes the incoming message as an OMF compliant message, it processes it using the built-in OMF module and converts it into a WOS request which it handles in the normal way. Standard WOS requests simply pass through the OMF module unchanged and are processed in the normal manner.

ANNEX A – XML Schema definitions (Normative)

Introduction

This annex contains the normative XML Schema definitions of the WOS operations, capabilities document, and exceptions.

wos.xsd

The file *wos.xsd* defines the set of base types from which other services such as the WOS, WFS or WRS may be derived.

```
<?xml version="1.0"?>
<xsd:schema
   targetNamespace="http://www.opengis.net/wos"
   xmlns:wos="http://www.opengis.net/wos"
   xmlns:ogc="http://www.opengis.net/ogc"
   xmlns:gml="http://www.opengis.net/gml"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   elementFormDefault="qualified">

   <xsd:annotation>
      <xsd:documentation>
         INTRODUCTION:
         This file defines a set of base clases from which data
         service interfaces, such as the WFS or WRS interfaces
         may be derived.  For example, the essential operatons of
      a
         query only WFS could be defined by the following schema
         fragment using this file:

         COMMON ATTRIBUTES:

         Many of the types and elements defined in this file
      include
         common attributes.  These are described here to avoid
      repeating
         the same  description.

         version:
         The version attribute is used to indicate which version
      of
         the specification an operation conforms to.  It is a
      mandatory
         parameter for all operations except the GetCapabilities
      operation.

         service:
         The service attribute is used to indicate which service
      type
         should be invoked at a server that implements multiple
      services
         at the same URL.
```

```
        handle:
        The handle attribute allows a client to assign a
    menmonic
        name to a WOS operation for error reporting purposes.
    If
        an error occurs processing the operation, a WOS may
    report
        the handle to identify the failed operation.

        objectName:
        The objectName attribute is used to indicate the name of
    the
        object type being manipulated.
     </xsd:documentation>
  </xsd:annotation>

  <!--
    ============================================================
    ===
       Includes and Imports

    ============================================================
    === -->
  <xsd:import namespace="http://www.opengis.net/gml"
              schemaLocation="../gml21/feature.xsd"/>
  <xsd:import namespace="http://www.opengis.net/ogc"
              schemaLocation="filter/filter.xsd"/>

  <!--
    ============================================================
    ===
       GETCAPABILITIES TYPE

    ============================================================
    === -->
  <xsd:complexType name="GetCapabilitiesType">
     <xsd:annotation>
        <xsd:documentation>
           This type defines the GetCapabilities operation.

           NOTE:  This definition will likely be superceded by
    the work
                  being done in OWS1.2 to extend the
    GetCapabilities
                  operation.
        </xsd:documentation>
     </xsd:annotation>
     <xsd:attribute name="version" type="xsd:string"
    use="optional">
        <xsd:annotation>
           <xsd:documentation>
              The version number on the GetCapabilities
    operation
              indirectly indicates the version of the
    capabilities
              document that the server will generate since the
    version
```

```
            of the capabilities document is tied to version of
   the
            operation.

            If the version parameter is not specified at all,
   the server
            should execute the highest version of the
   operation that it
            supports.

            Version Negotiation:

            A client app requests that a particular version of
   the
            GetCapabilities operation be executed (since the
   client
            knows how to process the response that the
   requested
            version will generate).  If the server does not
   support
            the requested version, it must execute the nearest
   version
            of the operation that is less than the requested
   version.
            The client may then choose to request another
   version or
            terminate the interaction if no agreeable version
   can
            be found.
         </xsd:documentation>
      </xsd:annotation>
    </xsd:attribute>
    <xsd:attribute name="service" type="xsd:string"
   use="required"/>
    <xsd:attribute name="handle"  type="xsd:string"
   use="optional"/>
</xsd:complexType>

<!--
   ============================================================
   ===
      DESCRIBEOBJECT TYPE

   ============================================================
   === -->
<xsd:complexType name="DescribeObjectType">
    <xsd:annotation>
       <xsd:documentation>
          The DescribeObjectType type, defines an operation
   that
          allows a client to obtain a schema document
   describing
          the type of the named object.
       </xsd:documentation>
    </xsd:annotation>
    <xsd:sequence>
       <xsd:element name="ObjectName" type="xsd:QName"
                    minOccurs="0" maxOccurs="unbounded">
```

```
            <xsd:annotation>
               <xsd:documentation>
                  The ObjectName element is used to specify the
   name of the
                  object to be described.  Multiple objects may
   be described
                  by specifying multiple ObjectName elements.

                  If no ObjectName elements are specified, then a
   WOS should
                  describe the types of all objects that it
   manages.
               </xsd:documentation>
            </xsd:annotation>
         </xsd:element>
       </xsd:sequence>
       <xsd:attribute name="version" type="xsd:string"
      use="required" />
       <xsd:attribute name="service" type="xsd:string"
      use="required" />
       <xsd:attribute name="handle"  type="xsd:string"
      use="optional" />
       <xsd:attribute name="outputFormat" type="xsd:string"
      use="optional"
                      default="XMLSCHEMA">
         <xsd:annotation>
            <xsd:documentation>
               The outputFormat attribute indicates the type of
   document
               that the server should generate in response to a
   DescribeObject
               request.  The default value, XMLSCHEMA, means that
   an
               XML-Schema document should be generated.  Other
   values
               are also possible as long as they are advertised
   in the
               capabilities document.
            </xsd:documentation>
         </xsd:annotation>
       </xsd:attribute>
   </xsd:complexType>

   <!--
      ============================================================
      ===
         GETOBJECT & GETOBJECTWITHLOCK TYPE

      ============================================================
      === -->
   <xsd:complexType name="GetObjectType">
      <xsd:annotation>
         <xsd:documentation>
            The GetObjectType type defines an operation that
   allows
            object instances to be retrived from an object
   repository.
```

```
            This operation may contain one or more Query elements
      that
            define one or more queries to be executed on the
      repository.

            The response to this operation must be defined when
      the
            operation is instantiated in an interface.  For
      example,
            in the WFS, the reponse to a GetFeature operation
      (which
            is an extension of the type GetObjectType) is defined
      as
            GML2.
         </xsd:documentation>
       </xsd:annotation>
       <xsd:sequence>
          <xsd:element ref="wos:Query" maxOccurs="unbounded"/>
       </xsd:sequence>
       <xsd:attribute name="version" type="xsd:string"
      use="required" />
       <xsd:attribute name="service" type="xsd:string"
      use="required" />
       <xsd:attribute name="handle"  type="xsd:string"
      use="optional" />
   </xsd:complexType>
   <xsd:complexType name="GetObjectWithLockType">
      <xsd:annotation>
         <xsd:documentation>
            This type is similar to the GetObject type except
      that
            it indicates that a WOS should attempt to lock the
      object
            instances identified by the operation.
         </xsd:documentation>
       </xsd:annotation>
       <xsd:sequence>
          <xsd:element ref="wos:Query" maxOccurs="unbounded"/>
       </xsd:sequence>
       <xsd:attribute name="version" type="xsd:string"
      use="required" />
       <xsd:attribute name="service" type="xsd:string"
      use="required" />
       <xsd:attribute name="handle"  type="xsd:string"
      use="optional" />
       <xsd:attribute name="lockAction" type="wos:AllSomeType"
      use="optional">
          <xsd:annotation>
             <xsd:documentation>
                The lockAction attribute controls how a WOS tries
      to
                acquire locks on objects identified by the
      operation.  A
                value of ALL means that the WOS must be able to
      lock all
                object instances identified by an operation.  If
      the server
```

```
                is unable to lock all object instances (because
      some are
                already locked) the operation should fail.  A
      value
                of SOME means that a WOS should try to lock as
      many
                object instances as possible.
             </xsd:documentation>
          </xsd:annotation>
       </xsd:attribute>
   </xsd:complexType>
   <xsd:element name="Query">
       <xsd:annotation>
          <xsd:documentation>
             The Query element contains the description of a
      single query
             to be executed on an object repository for the
      purpose
             of retrieving object instances of the specified
      object type.
          </xsd:documentation>
       </xsd:annotation>
       <xsd:complexType>
          <xsd:complexContent>
             <xsd:extension base="wos:QueryType"/>
          </xsd:complexContent>
       </xsd:complexType>
   </xsd:element>
   <xsd:complexType name="QueryType">
       <xsd:annotation>
          <xsd:documentation>
             The QueryType type defines the Query element which is
             used to describe a query on an object repository.

             A query is specified by indicating:

                1) Which object instances to retrieve by defining
                   a query constraint using the QueryConstraint
                   element.

                2) What properties of an object type to retrieve.
                   If no properties are specified in the request
                   then this should be interpreted as fetching
                   all the properties or getting the entire object
                   instance.

                3) In what order the results should be presented
      using
                   the SortBy element.

             Each of these parts is optional.  If none are
      specified
             then that should be interpreted as meaning that all
      object
             instances of the specified object type should be
      retrieved.
          </xsd:documentation>
       </xsd:annotation>
```

```
<xsd:sequence>
   <xsd:choice>
      <xsd:element name="wos:PropertySet"
                   type="wos:PropertySetType"
                   minOccurs="0" maxOccurs="1">
         <xsd:annotation>
            <xsd:documentation>
               The PropertySet element is used to indicate
that a
               well known, named, set of object properties
should
               be retrieved.
            </xsd:documentation>
         </xsd:annotation>
      </xsd:element>
      <xsd:element ref="ogc:PropertyName"
                   minOccurs="0" maxOccurs="unbounded">
         <xsd:annotation>
            <xsd:documentation>
               The PropertName element allows a client to
enumerate
               the object properties that a query should
retrieve.
               Each property name is enclosed in a single
PropertyName
               element.
            </xsd:documentation>
         </xsd:annotation>
      </xsd:element>
   </xsd:choice>
   <xsd:element name="QueryConstraint"
                type="QueryConstraintType"
                minOccurs="0" maxOccurs="1">
      <xsd:annotation>
         <xsd:documentation>
            The QueryConstraint element contains a query
predicate
            that constrains the query to a specific set of
object
            instances that satisfy the predicate.
         </xsd:documentation>
      </xsd:annotation>
   </xsd:element>

   <xsd:element ref="ogc:SortBy"
                minOccurs="0" maxOccurs="1">
      <xsd:annotation>
         <xsd:documentation>
            The SortBy element contains elements that
define
            in what order the query results should be
presented.
         </xsd:documentation>
      </xsd:annotation>
   </xsd:element>
</xsd:sequence>
<xsd:attribute name="handle"
               type="xsd:string" use="optional"/>
```

```
    <xsd:attribute name="objectName"
                   type="wos:ObjectNameListType"
    use="required">
       <xsd:annotation>
          <xsd:documentation>
             The objecName attribute is used to indicate the
    name
             of the object type being queried.

             This attribute is actually a list of object type
    names
             meaning that joins can be supported if the
    predicate
             language allows joins to be specified.
          </xsd:documentation>
       </xsd:annotation>
    </xsd:attribute>
    <xsd:attribute name="objectVersion"
                   type="xsd:string" use="optional">
       <xsd:annotation>
          <xsd:documentation>
             If a WOS supports object versioning, then this
    attribute
             may be used to specify which version of an object
    to
             retrieve.  The value is a postive non-zero integer
    which
             indicates which version should be retrieved.  The
    number 1
             is used to indicate the oldest available version.
    The default
             action is to retrieve that more recent version.
    If a version
             number greater than the largest version number is
    specified
             then the most recent version should be presented.
          </xsd:documentation>
       </xsd:annotation>
    </xsd:attribute>
</xsd:complexType>
<xsd:complexType name="QueryConstraintType">
    <xsd:annotation>
       <xsd:documentation>
          The QueryConstraintType type allows a query predicate
    clause
          to be encoded using one or several supported
    predicate languages.
          The predicate languages supported include the OGC
    Filter
          specification, the OGC Common Query Language, Simple
    Feature SQL,
          and SQLMM from ISO.
       </xsd:documentation>
    </xsd:annotation>
    <xsd:choice>
       <xsd:element ref="ogc:Filter" />
       <xsd:element name="CqlText" type="xsd:string" />
       <xsd:element name="SfSql"   type="xsd:string" />
```

```
      <xsd:element name="SqlMM"   type="xsd:string" />
   </xsd:choice>
</xsd:complexType>
<xsd:simpleType name="ObjectNameListType">
   <xsd:list itemType="xsd:QName"/>
</xsd:simpleType>
<xsd:complexType name="PropertySetType">
   <xsd:attribute name="setName" type="xsd:string"
  use="optional"/>
</xsd:complexType>

<!--
   ================================================================
   ===
      LOCKOBJECT TYPE

   ================================================================
   === -->
<xsd:complexType name="LockObjectType">
   <xsd:annotation>
      <xsd:documentation>
         The LockObjectType type is used to define an
  operation
         that locks object instances.  The root element of the
         operation is actually a container of one or more Lock
         elements that define locks on individual objects
  classes.
      </xsd:documentation>
   </xsd:annotation>
   <xsd:sequence>
      <xsd:element name="Expiry"
                   type="xsd:positiveInteger" minOccurs="0">
         <xsd:annotation>
            <xsd:documentation>
               An object instance remains locked until a
  transaction
               has been executed on that instances OR until
  the value
               of this attribute, specified in minutes, has
  elapsed.
               Once the expiry period on a lock has elapsed,
  the lock
               is automatically released.
            </xsd:documentation>
         </xsd:annotation>
      </xsd:element>
      <xsd:element name="Lock"
                   type="wos:LockType" maxOccurs="unbounded"/>
   </xsd:sequence>
   <xsd:attribute name="version" type="xsd:string"
  use="required" />
   <xsd:attribute name="service" type="xsd:string"
  use="required" />
   <xsd:attribute name="handle"  type="xsd:string"
  use="optional" />
</xsd:complexType>
<xsd:complexType name="LockType">
   <xsd:annotation>
```

```
      <xsd:documentation>
         The LockType type is used to define the Lock element
         which describes a locking operation to be performed
         on object instances.
      </xsd:documentation>
   </xsd:annotation>
   <xsd:sequence>
      <xsd:element name="QueryConstraint"
                   type="QueryConstraintType"
                   minOccurs="0" maxOccurs="1">
         <xsd:annotation>
            <xsd:documentation>
               The QueryConstraint element is used to specify
   a query
               predicate that identifies the set of object
   instances
               that are to be locked.

               The objects to be locked can be enumerated
   individually
               by object identifier or identified as the set
   of objects
               instances that satisfy the specified predicate.
            </xsd:documentation>
         </xsd:annotation>
      </xsd:element>
   </xsd:sequence>
   <xsd:attribute name="handle" type="string" use="optional"
   />
   <xsd:attribute name="objectName" type="xsd:QName"
   use="required" />
   <xsd:attribute name="lockAction" type="wos:AllSomeType"
   use="optional">
      <xsd:annotation>
         <xsd:documentation>
            The lockAction attribute indicates how a WOS
   should attempt
            to acquire locks on the request object instances.
   A value of
            ALL means that the WOS must either lock all object
   instances
            identified otherwise the operation should fail.  A
   value of
            SOME, indicates that a WOS should attempt to lock
   as many
            object instances as it can.  In this case, the
   response
            document for this operation indicates which object
   instances
            were locked and which were not locked.
         </xsd:documentation>
      </xsd:annotation>
   </xsd:attribute>
</xsd:complexType>

<!--
   ============================================================
   ===
```

```
      TRANSACTION TYPE

   ============================================================
   === -->
<xsd:complexType name="TransactionType">
   <xsd:annotation>
      <xsd:documentation>
         The TransactionType type, defines an operation that
   allows
         object instances to be inserted or created, updated
   or deleted
         from an object repository.

         A transaction operation may contain LockId element
   that contains
         a previously obtained identifier associated with a
   locked set of
         object instances.  Only object instances locked under
   the specified
         LockId can be operated on by the transaction if the
   LockId element
         is specified.  If it is not specified, then any
   object instance
         can be operated upon.

         It may also contain zero or more Insert, Update or
   Delete element
         that create, change or remove object instances from
   the object
         repository.
      </xsd:documentation>
   </xsd:annotation>
   <xsd:sequence>
      <xsd:element ref="wos:LockId" minOccurs="0"
   maxOccurs="1" />
      <xsd:choice minOccurs="0" maxOccurs="unbounded">
         <xsd:element ref="wos:Insert" minOccurs="0"
   maxOccurs="unbounded"/>
         <xsd:element ref="wos:Update" minOccurs="0"
   maxOccurs="unbounded"/>
         <xsd:element ref="wos:Delete" minOccurs="0"
   maxOccurs="unbounded"/>
         <xsd:element ref="wos:Native" minOccurs="0"
   maxOccurs="unbounded"/>
      </xsd:choice>
   </xsd:sequence>
   <xsd:attribute name="version" type="xsd:string"
   use="required" />
   <xsd:attribute name="service" type="xsd:string"
   use="required" />
   <xsd:attribute name="handle"  type="xsd:string"
   use="optional" />
   <xsd:attribute name="releaseAction"
                  type="wos:AllSomeType" use="optional">
      <xsd:annotation>
         <xsd:documentation>
            The releaseAction attribute control how locks are
   released
```

```
            when a transaction operation terminates.

            A value of ALL indicates that all locks on object
   instances
            locked with the specified LockId should be
   released.

            A value of SOME indicates that only the locks on
   object
            instances actually operated on by the transaction
   should be
            removed.  Objects not operated on by the
   transaction should
            remain locked.  The LockId is valid as long as one
   or more
            object instances remain locked.  Subsequent
   transactions can
            be executed on the locked instances using the same
   LockId.
         </xsd:documentation>
      </xsd:annotation>
   </xsd:attribute>
</xsd:complexType>
<xsd:element name="LockId" type="xsd:string">
   <xsd:annotation>
      <xsd:documentation>
         When a client app locks object instances using a Lock
   or
         GetObjectWithLock operation, the server generates a
   unique
         lockid that is passed back to the client.  The client
   can
         then use that LockId to reference the locked objects
   in
         subsequent transactions.
      </xsd:documentation>
   </xsd:annotation>
</xsd:element>
<xsd:element name="Insert" type="wos:InsertElementType"/>
<xsd:element name="Update" type="wos:UpdateElementType"/>
<xsd:element name="Delete" type="wos:DeleteElementType"/>
<xsd:complexType name="InsertElementType">
   <xsd:annotation>
      <xsd:documentation>
         The Insert element may contain any XML encoded object
         or proxy for an object.  Optionally, an Insert
   element
         may contain an object reference for objects that are
   not
         easily mapped to XML elements.
      </xsd:documentation>
   </xsd:annotation>
   <xsd:sequence>
      <xsd:choice>
         <xsd:any namespace="##targetNamespace"/>
         <xsd:element ref="wos:ObjectRef" />
      </xsd:choice>
   </xsd:sequence>
```

```
   <xsd:attribute name="handle" type="xsd:string"
   use="optional"/>
</xsd:complexType>
<xsd:complexType name="UpdateElementType">
   <xsd:annotation>
      <xsd:documentation>
         An Update element may contain a reference to an
   entire
         object instance that is to replace an existing object
   instance.

         Optionally, an Update element may contain a list of
   object
         properties and values that are to be used to update
   the
         corrsponding object properties.  The new value of a
   property
         may be encoded directly in the operation or a
   reference to
         the property value may be used.

         The object instances affected by the update operation
   are
         identified using the QueryConstaint element.
      </xsd:documentation>
   </xsd:annotation>
   <xsd:sequence>
      <xsd:element ref="wos:Property"
                   maxOccurs="unbounded"/>
      <xsd:element name="QueryConstraint"
                   type="QueryConstraintType"
                   minOccurs="0" maxOccurs="1"/>
   </xsd:sequence>
   <xsd:attribute name="handle" type="xsd:string"
   use="optional"/>
   <xsd:attribute name="objectName" type="xsd:QName"
   use="required" />
</xsd:complexType>
<xsd:complexType name="DeleteElementType">
   <xsd:annotation>
      <xsd:documentation>
         A delete element constains a QueryConstraint that
   identifies
         the feature instances of the specified object type
   that are
         to be removed from the repository.
      </xsd:documentation>
   </xsd:annotation>
   <xsd:sequence>
      <xsd:element name="QueryConstraint"
                   type="QueryConstraintType"
                   minOccurs="0" maxOccurs="1" />
   </xsd:sequence>
   <xsd:attribute name="handle" type="xsd:string"
   use="optional"/>
   <xsd:attribute name="objectName" type="xsd:QName"
   use="required" />
</xsd:complexType>
```

```
<!-- define structure to specify a property value -->
<xsd:element name="Property" type="wos:PropertyType">
   <xsd:annotation>
      <xsd:documentation>
         The Property element is used to specify the new value
   of
         a property in an update operation.  The element
   constains
         the name of the property and its new value OR a
   reference
         to its new value.
      </xsd:documentation>
   </xsd:annotation>
</xsd:element>
<xsd:complexType name="PropertyType">
   <xsd:sequence>
      <xsd:element name="Name" type="xsd:string">
         <xsd:annotation>
            <xsd:documentation>
               The Name element contains the name of a feature
   property
               to be updated.
            </xsd:documentation>
         </xsd:annotation>
      </xsd:element>
      <xsd:choice minOccurs="1" maxOccurs="1">
         <xsd:element name="Value">
            <xsd:annotation>
               <xsd:documentation>
                  The Value element contains the replacement
   value for the
                  named property.
               </xsd:documentation>
            </xsd:annotation>
         </xsd:element>
         <xsd:element name="ValueRef" type="wos:ValueRefType">
            <xsd:annotation>
               <xsd:documentation>
                  The ValueRef element contains a reference to
   the
                  replacement value for the named property.
               </xsd:documentation>
            </xsd:annotation>
         </xsd:element>
      </xsd:choice>
   </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="ValueRefType">
   <xsd:annotation>
      <xsd:documentation>
         The reference to a value is compose of a link to the
   location
         of the data and a mime type used to indicate what
   type of data
         is being pointed to.
      </xsd:documentation>
   </xsd:annotation>
   <xsd:attribute name="resolveURL"
```

```
                type="xsd:boolean" use="optional"
   default="true" />
    <xsd:attribute name="mimeType"
                type="xsd:string" use="optional"
   default="text/xml" />
    <xsd:attributeGroup ref="wos:objRef"/>
</xsd:complexType>

<xsd:element name="ObjectRef" type="wos:ObjectRefType">
    <xsd:annotation>
       <xsd:documentation>
          An object reference is composed of a link to the
   resource
          and a mime type indicating the type of data being
   pointed
          to.
       </xsd:documentation>
    </xsd:annotation>
</xsd:element>
<xsd:complexType name="ObjectRefType">
    <xsd:attribute name="resolveURL"
                type="xsd:boolean" use="optional"
   default="true" />
    <xsd:attribute name="mimeType"
                type="xsd:string" use="optional"
   default="text/xml" />
    <xsd:attributeGroup ref="wos:objRef"/>
</xsd:complexType>

<xsd:attributeGroup name="objRef">
    <xsd:annotation>
       <xsd:documentation>
       </xsd:documentation>
    </xsd:annotation>
    <xsd:attribute name="resolveURL"
                type="xsd:boolean"
                use="optional"
                default="true" />
    <xsd:attribute name="mimeType"
                type="xsd:string"
                use="optional"
                default="text/xml"/>
    <xsd:attribute name="href"
                type="xsd:anyURI"
                use="required" />
</xsd:attributeGroup>

<!--
   ============================================================
   ===
      RESPONSE TYPES

   ============================================================
   === -->
<xsd:complexType name="LockObjectResponseType">
  <xsd:sequence>
    <xsd:element ref="wos:LockId"/>
    <xsd:element name="ObjectsLocked"
```

```
                type="wos:ObjectsLockedType" minOccurs="0"/>
    <xsd:element name="ObjectsNotLocked"
                type="wos:ObjectsNotLockedType"
   minOccurs="0"/>
  </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="ObjectsLockedType">
  <xsd:sequence maxOccurs="unbounded">
    <xsd:element ref="ogc:ObjectId"/>
  </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="ObjectsNotLockedType">
  <xsd:sequence maxOccurs="unbounded">
    <xsd:element ref="ogc:ObjectId"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="TransactionResponseType">
   <xsd:sequence>
     <xsd:element name="InsertResult"
                 type="wos:InsertResultType"
                 minOccurs="0" maxOccurs="unbounded"/>
     <xsd:element name="TransactionResult"
                 type="wos:TransactionResultType"/>
   </xsd:sequence>
   <xsd:attribute name="version" type="xsd:string"
  use="required" />
</xsd:complexType>
<xsd:complexType name="TransactionResultType">
   <xsd:sequence>
     <xsd:element name="Status" type="wos:StatusType"/>
     <xsd:element name="Locator" type="xsd:string"
  minOccurs="0"/>
     <xsd:element name="Message" type="xsd:string"
  minOccurs="0"/>
   </xsd:sequence>
   <xsd:attribute name="handle"  type="xsd:string"
  use="optional" />
</xsd:complexType>
<xsd:complexType name="InsertResultType">
   <xsd:sequence>
     <xsd:element ref="ogc:ObjectId" maxOccurs="unbounded"/>
   </xsd:sequence>
   <xsd:attribute name="handle"  type="xsd:string"
  use="optional" />
</xsd:complexType>
<xsd:complexType name="StatusType">
   <xsd:choice>
      <xsd:element ref="wos:SUCCESS"/>
      <xsd:element ref="wos:FAILED"/>
      <xsd:element ref="wos:PARTIAL"/>
   </xsd:choice>
</xsd:complexType>
<xsd:element name="SUCCESS" type="wos:EmptyType"/>
<xsd:element name="FAILED" type="wos:EmptyType"/>
<xsd:element name="PARTIAL" type="wos:EmptyType"/>
```

```
   <!--
      ============================================================
      ===
         NATIVE REQUEST

      ============================================================
      === -->
   <xsd:element name="Native" type="wos:NativeType"/>
   <xsd:complexType name="NativeType">
      <xsd:attribute name="vendorId" type="xsd:string"
      use="required"/>
      <xsd:attribute name="safeToIgnore" type="xsd:boolean"
      use="required"/>
   </xsd:complexType>

   <!-- MISC TYPES -->
   <xsd:complexType name="EmptyType"/>
   <xsd:simpleType name="AllSomeType">
      <xsd:restriction base="xsd:string">
         <xsd:enumeration value="ALL"/>
        <xsd:enumeration value="SOME"/>
      </xsd:restriction>
   </xsd:simpleType>
</xsd:schema>
```

WOS-query.xsd

The *WOS-query.xsd* file defines the basic operations for a Web Object Service.

```
<?xml version="1.0"?>
<xsd:schema
    targetNamespace="http://www.opengis.net/wos"
    xmlns:wos="http://www.opengis.net/wos"
    xmlns:ogc="http://www.opengis.net/ogc"
    xmlns:gml="http://www.opengis.net/gml"
    xmlns:xlink="http://www.w3.org/1999/xlink"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    elementFormDefault="qualified">

   <!--
      ============================================================
      ===
         Includes and Imports

      ============================================================
      === -->
   <xsd:import namespace="http://www.opengis.net/gml"
               schemaLocation="../../gml2.1/feature.xsd"/>
   <xsd:import namespace="http://www.opengis.net/wos"
               schemaLocation="../wos.xsd"/>

   <!--
      ============================================================
      ===
         REQEUST MESSAGES
```

```
   ============================================================
   === -->
<xsd:element name="GetCapabilities"
             type="wos:GetCapabilitiesType"/>
<xsd:element name="DescribeObjectType"
             type="wos:DescribeObjectType"/>
<xsd:element name="GetObject">
   <xsd:complexType>
      <xsd:complexContent>
         <xsd:extension base="wos:GetObjectType">
            <xsd:attribute name="outputFormat"
                           type="xsd:string"
                           use="optional" default="XML" />
            <xsd:attribute name="archiver"
                           type="ows:knownArchivers"
                           use="optional"/>
            <xsd:attribute name="compressor"
                           type="ows:knownCompressors"
                           use="optional"/>
         </xsd:extension>
      </xsd:complexContent>
   </xsd:complexType>
</xsd:element>

<!--
   ============================================================
   ===
      RESPONSE MESSAGES

   ============================================================
   === -->
<xsd:element name="ObjectCollection"
             type="wos:ObjectInstanceType" />

<!--
   ============================================================
   ===
      RESPONSE TYPES

   ============================================================
   === -->
<xsd:complexType name="ObjectCollectionType">
   <xsd:sequence>
      <xsd:element name="ObjectInstance"
                   type="wos:ObjectInstanceType"
                   maxOccurs="unbounded" />
   </xsd:sequence>
   <xsd:attribute name="lockId" type="xsd:string"
  use="optional" />
</xsd:complexType>
<xsd:complexType name="ObjectInstanceType">
   <xsd:sequence>
      <xsd:choice>
         <xsd:any namespace="##targetNamespace"/>
         <xsd:element ref="wos:ObjectRef" />
      </xsd:choice>
   </xsd:sequence>
```

```
      <xsd:attribute name="oid" type="xsd:anyURI" use="required"
      />
   </xsd:complexType>

   <xsd:simpleType name="knownArchivers">
      <xsd:restriction base="xsd:string">
         <xsd:enumeration value="TAR"/>
         <xsd:enumeration value="ZIP"/>
      </xsd:restriction>
   </xsd:simpleType>

   <xsd:simpleType name="knownCompressors">
      <xsd:restriction base="xsd:string">
         <xsd:enumeration value="COMPESS"/>
         <xsd:enumeration value="GZIP"/>
      </xsd:restriction>
   </xsd:simpleType>

</xsd:schema>
```

WOS-transaction.xsd

This *WOS-transaction.xsd* file defines additional, optional, operations to support transaction and object locking.

```
<?xml version="1.0"?>
<xsd:schema
    targetNamespace="http://www.opengis.net/wos"
    xmlns:wos="http://www.opengis.net/wos"
    xmlns:ogc="http://www.opengis.net/ogc"
    xmlns:gml="http://www.opengis.net/gml"
    xmlns:xlink="http://www.w3.org/1999/xlink"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    elementFormDefault="qualified">

   <!--
      ============================================================
      ===
         Includes and Imports

      ============================================================
      === -->
   <xsd:import namespace="http://www.opengis.net/gml"
               schemaLocation="../../gml2.1/feature.xsd"/>
   <xsd:import namespace="http://www.opengis.net/wos"
               schemaLocation="../wos.xsd"/>

   <!--
      ============================================================
      ===
         REQEUST MESSAGES

      ============================================================
      === -->
   <xsd:element name="GetObjectWithLock">
      <xsd:complexType>
         <xsd:complexContent>
```

```
            <xsd:extension base="wos:GetObjectWithLockType">
               <xsd:attribute name="outputFormat"
                              type="xsd:string"
                              use="optional" default="XML" />
            </xsd:extension>
         </xsd:complexContent>
      </xsd:complexType>
   </xsd:element>
   <xsd:element name="LockObject"
                type="wos:LockObjectType"/>
   <xsd:element name="Transaction"
                type="wos:TransactionType"/>

   <!--
      ================================================================
      ===
         RESPONSE MESSAGES

      ================================================================
      === -->
   <xsd:element name="LockObjectResponse"
                type="wos:LockObjectResponseType"/>
   <xsd:element name="TransactionResponse"
                type="wos:TransactionResponseType"/>
</xsd:schema>
```

OGC-exception.xsd

The *OGC-exception.xsd* file defines a standard exception message schema for all OGC services.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema
   targetNamespace="http://www.opengis.net/ogc"
   xmlns:ogc="http://www.opengis.net/ogc"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   elementFormDefault="qualified">

   <xsd:element name="ServiceExceptionReport">
      <xsd:complexType>
         <xsd:sequence>
            <xsd:element name="ServiceException"
                         type="ogc:ServiceExceptionType"
                         minOccurs="0" maxOccurs="unbounded"/>
         </xsd:sequence>
         <xsd:attribute name="version" type="xsd:string"
      fixed="1.2.0"/>
      </xsd:complexType>
   </xsd:element>

   <xsd:complexType name="ServiceExceptionType">
      <xsd:simpleContent>
         <xsd:restriction base="xsd:string">
            <xsd:attribute name="code" type="xsd:string"/>
            <xsd:attribute name="locator" type="xsd:string"/>
         </xsd:restriction>
      </xsd:simpleContent>
```

```
   </xsd:complexType>
</xsd:schema>
```

ANNEX B – Web Feature Service profile (Informative)

Introduction

The schema files in this annex are meant to illustrate how to derive the WFS interface using the base types defined in *wos.xsd* [Annex A].

WFS-query.xsd

```
<?xml version="1.0"?>
<xsd:schema
    targetNamespace="http://www.opengis.net/wfs"
    xmlns:wfs="http://www.opengis.net/wfs"
    xmlns:wos="http://www.opengis.net/wos"
    xmlns:ogc="http://www.opengis.net/ogc"
    xmlns:gml="http://www.opengis.net/gml"
    xmlns:xlink="http://www.w3.org/1999/xlink"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    elementFormDefault="qualified">

   <!-- ============================================================
        Includes and Imports
        ============================================================ -->
   <xsd:import namespace="http://www.opengis.net/gml"
               schemaLocation="../../gml2.1/feature.xsd"/>
   <xsd:import namespace="http://www.opengis.net/wos"
               schemaLocation="../wos.xsd"/>

   <!-- ============================================================
        REQEUST MESSAGES
        ============================================================ -->
   <xsd:element name="GetCapabilities"
                type="wos:GetCapabilitiesType"/>
   <xsd:element name="DescribeFeatureType"
                type="wos:DescribeObjectTypeType"/>
   <xsd:element name="GetFeature">
      <xsd:complexType>
         <xsd:complexContent>
            <xsd:extension base="wos:GetObjectType">
               <xsd:attribute name="maxFeatures"
                              type="xsd:positiveInteger"
                              use="optional" />
            </xsd:extension>
         </xsd:complexContent>
      </xsd:complexType>
   </xsd:element>

   <!-- ============================================================
        RESPONSE MESSAGES
        ============================================================ -->
   <xsd:element name="FeatureCollection"
                type="wfs:FeatureCollectionType"
                substitutionGroup="gml:_FeatureCollection"/>

   <!-- ============================================================
        REPONSE TYPES
        ============================================================ -->
   <xsd:complexType name="FeatureCollectionType">
     <xsd:complexContent>
       <xsd:extension base="gml:AbstractFeatureCollectionType">
         <xsd:attribute name="lockId" type="xsd:string" use="optional"/>
       </xsd:extension>
     </xsd:complexContent>
   </xsd:complexType>
</xsd:schema>
```

WFS-transaction.xsd

```
<?xml version="1.0"?>
<xsd:schema
    targetNamespace="http://www.opengis.net/wfs"
    xmlns:wfs="http://www.opengis.net/wfs"
    xmlns:wos="http://www.opengis.net/wos"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:xlink="http://www.w3.org/1999/xlink"
    xmlns:gml="http://www.opengis.net/gml"
    xmlns:ogc="http://www.opengis.net/ogc"
    elementFormDefault="qualified">

   <!-- ============================================================
        Includes and Imports
        ============================================================ -->
   <xsd:include schemaLocation="WFS-basic.xsd"/>
   <xsd:import namespace="http://www.opengis.net/gml"
               schemaLocation="../gml21/feature.xsd"/>
   <xsd:import namespace="http://www.opengis.net/wos"
               schemaLocation="../wos.xsd"/>

   <!-- ============================================================
        REQUEST MESSAGES
        ============================================================ -->
   <xsd:element name="GetFeatureWithLock">
      <xsd:complexType>
         <xsd:complexContent>
            <xsd:extension base="wos:GetObjectWithLockType">
               <xsd:attribute name="maxFeatures"
                              type="xsd:positiveInteger"
                              use="optional"/>
            </xsd:extension>
         </xsd:complexContent>
      </xsd:complexType>
   </xsd:element>
   <xsd:element name="LockFeature"
                type="wos:LockFeatureType"/>
   <xsd:element name="Transaction"
                type="wos:TransactionType"/>

   <!-- ============================================================
        RESPONSE MESSAGES
        ============================================================ -->
   <xsd:element name="WFS_LockFeatureResponse"
                type="wos:LockObjectResponseType"/>
   <xsd:element name="WFS_TransactionResponse"
                type="wos:TransactionResponseType"/>
</xsd:schema>
```

ANNEX C – Web Registry Service profile (Informative)

Introduction

The schema in this annex are meant to illustrate how the WRS interface may be derived using the base types defined in *wos.xsd* [Annex A].

WRS-basic.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema
   targetNamespace="http://www.opengis.net/wrs"
   xmlns:wrs="http://www.opengis.net/wrs"
   xmlns:wos="http://www.opengis.net/wos"
   xmlns:ogc="http://www.opengis.net/ogc"
   xmlns:xlink="http://www.w3.org/1999/xlink"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   elementFormDefault="qualified">

   <!-- ============================================================
        Includes and Imports
        ============================================================ -->
   <xsd:import namespace="http://www.opengis.net/ogc"
               schemaLocation="filter/filter.xsd"/>
   <xsd:import namespace="http://www.opengis.net/wos"
               schemaLocation="../wos.xsd"/>

   <!-- ============================================================
        REQEUST MESSAGES
        ============================================================ -->
   <xsd:element name="GetCapabilities">
      <xsd:complexType>
         <xsd:complexContent>
            <xsd:extension base="wos:GetCapabilitiesType">
               <xsd:attribute name="section" type="xsd:string" use="optional"/>
            </xsd:extension>
         </xsd:complexContent>
      </xsd:complexType>
   </xsd:element>
   <xsd:element name="DescribeRecordType" type="wos:DescribeObjectTypeType"/>
   <xsd:element name="GetRecord">
      <xsd:complexType>
         <xsd:complexContent>
            <xsd:extension base="wos:GetObjectType">
               <xsd:attribute name="outputRecType"
                              type="xsd:string" use="optional"/>
               <xsd:attribute name="startPosition"
                              type="xsd:positiveInteger" use="optional"/>
               <xsd:attribute name="maxRecords"
                              type="xsd:positiveInteger" use="optional"/>
            </xsd:extension>
         </xsd:complexContent>
      </xsd:complexType>
   </xsd:element>

   <!-- ============================================================
        RESPONSE MESSAGES
        ============================================================ -->
   <xsd:element name="searchResponse" type="wrs:searchResponseType">
      <xsd:annotation>
         <xsd:documentation>
            The top level holder for a response from a catalog.
         </xsd:documentation>
      </xsd:annotation>
   </xsd:element>
```

```
   <!-- ============================================================
        RESPONSE TYPES
        ============================================================ -->
   <!-- GETRECORD RESPONSE TYPE -->
   <xsd:complexType name="searchResponseType">
      <xsd:sequence>
         <xsd:element name="searchParameter"
                      type="wrs:searchParameterType"
                      minOccurs="0" maxOccurs="unbounded"/>
         <xsd:element name="searchStatus"
                      type="wrs:searchStatusType"/>
         <xsd:element name="searchResults"
                      type="wrs:searchResultsType"/>
      </xsd:sequence>
      <xsd:attribute name="version" type="xsd:string" fixed="1.1.0"/>
   </xsd:complexType>
   <xsd:complexType name="searchParameterType">
      <xsd:simpleContent>
         <xsd:extension base="xsd:string">
            <xsd:attribute name="name" type="xsd:string" use="required"/>
         </xsd:extension>
      </xsd:simpleContent>
   </xsd:complexType>
   <xsd:complexType name="searchStatusType">
      <xsd:annotation>
         <xsd:documentation>
            This element contains a number of attributes that detail
            the status of the search.

            elementSetName  - The element set that has been returned
                              (e.g. brief, summary, full)
            numberOfRecords - The number of hits returned in this
                              result.
            schema          - The type of response returned (e.g.
                              OGCService, FGDC)
            success         - Was the search successfull (true,
                              false)
            timestamp       - The date and time at the completion
                              of the search.
         </xsd:documentation>
      </xsd:annotation>
      <xsd:attribute name="elementSetName"  type="xsd:string" fixed="brief"/>
      <xsd:attribute name="success"         type="xsd:string" default="true"/>
      <xsd:attribute name="numberOfRecords" type="xsd:string"/>
      <xsd:attribute name="schema"          type="xsd:string"/>
      <xsd:attribute name="timestamp"       type="xsd:string"/>
   </xsd:complexType>
   <xsd:complexType name="searchResultsType">
      <xsd:annotation>
         <xsd:documentation>
            The holder for the results from the catalog.
         </xsd:documentation>
      </xsd:annotation>
      <xsd:sequence>
         <xsd:any namespace="##targetNamespace"/>
      </xsd:sequence>
   </xsd:complexType>
</xsd:schema>
```

WRS-transaction.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema
   targetNamespace="http://www.opengis.net/wrs"
   xmlns:wrs="http://www.opengis.net/wrs"
   xmlns:wos="http://www.opengis.net/wos"
   xmlns:ogc="http://www.opengis.net/ogc"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   elementFormDefault="qualified">

   <!-- ============================================================
        Includes and Imports
        ============================================================ -->
```

```
   <xsd:include schemaLocation="WRS-basic.xsd"/>
   <xsd:import namespace="http://www.opengis.net/wos"
               schemaLocation="../wos.xsd"/>

   <!-- ================================================================
        REQUEST MESSAGES
        ================================================================ -->
   <!-- request messages -->
   <xsd:element name="GetRecordWithLock">
      <xsd:complexType>
         <xsd:complexContent>
            <xsd:extension base="wos:GetObjectWithLockType">
               <xsd:attribute name="outputRecType"
                              type="xsd:string" use="optional"/>
               <xsd:attribute name="startPosition"
                              type="xsd:positiveInteger" use="optional"/>
               <xsd:attribute name="maxRecords"
                              type="xsd:positiveInteger" use="optional"/>
            </xsd:extension>
         </xsd:complexContent>
      </xsd:complexType>
   </xsd:element>
   <xsd:element name="RegisterData"
                type="wrs:RegisterDataType"/>
   <xsd:element name="RegisterService"
                type="wrs:RegisterServiceType"/>
   <xsd:element name="LockRecord"
                type="wos:LockObjectType"/>
   <xsd:element name="Transaction"
                type="wos:TransactionType"/>

   <!-- ================================================================
        RESPONSE MESSAGES
        ================================================================ -->
   <xsd:element name="WRS_LockRecordResponse"
                type="wos:LockObjectResponseType"/>
   <xsd:element name="WRS_TransactionResponse"
                type="wos:TransactionResponseType"/>

   <!-- ================================================================
        TYPES
        ================================================================ -->
   <!-- REGISTER SERVICE -->
   <xsd:complexType name="RegisterServiceType">
      <xsd:sequence>
         <xsd:element name="Name" type="xsd:string" minOccurs="0" />
         <xsd:element name="Title" type="xsd:string" minOccurs="0" />
         <xsd:element name="ServiceAddr" type="wrs:ServiceAddrType" />
         <xsd:element name="ServiceOwnerContactInfo" type="xsd:string" />
         <xsd:element name="HarvestFrequency" type="xsd:positiveInteger" />
      </xsd:sequence>
   </xsd:complexType>
   <xsd:complexType name="ServiceAddrType">
      <xsd:attribute name="url" type="xsd:anyURI" use="required"/>
   </xsd:complexType>
   <!-- REGISTER DATA -->
   <xsd:complexType name="RegisterDataType">
      <xsd:sequence>
         <xsd:element name="Name" type="xsd:string" minOccurs="0" />
         <xsd:element name="Title" type="xsd:string" minOccurs="0" />
         <xsd:element name="DataMetadataAddr" type="wrs:DataMetadataAddrType" />
         <xsd:element name="ServiceOwnerContactInfo" type="xsd:string" />
         <xsd:element name="HarvestFrequency" type="xsd:positiveInteger" />
      </xsd:sequence>
   </xsd:complexType>
   <xsd:complexType name="DataMetadataAddrType">
      <xsd:attribute name="url" type="xsd:anyURI" use="required"/>
   </xsd:complexType>
</xsd:schema>
```

ANNEX D - Conformance tests (Normative)

Specific conformance tests for Web Object Services have not yet been determine and will be added in a future revision of this specification.

At the moment, a WOS implementation must satisfy the following system characteristics to be minimally conformant with this specification:

1. The GetCapabilities, DescribeObjectType and GetObject operations must be supported.

2. The Extensible Markup Language (XML) document returned in response to a GetCapabilities request must be valid against the XML Schema definition in Annex A.4. Such validation may be performed using commonly available XML validating tools.

3. In response to a GetObject operation, the WOS must be able to generate GML [2] as one of its output formats.

4. The Extensible Markup Language (XML) document returned in response to a GetObject request must validate against the schema generated in response to a DescribeObjectType request. Such validation may be performed using commonly available XML validating tools.

5. All clauses in the normative clauses of this specification that use the keywords "must", "must not", "required", "shall", and "shall not" have been satisfied.

Bibliography

[16] de La Beaujardière, Jeff (ed.), "OpenGIS Implementation Specification #01-047r2: Web Map Service Implementation Specification", June 2001

[17] Vretanos, Panagiotis, "OpenGIS Discussion Paper: Transaction Encoding Specification Version 0.0.5", March 2000

[18] Arctur, D., Pilkington, P., Cuthbert, A., "Spatial Object Transfer Format (SOTF): Initial High-Level Design, Version 1.2", Laser-Scan Inc., November 1999

[19] Rumbach, James, et al., "Unified Modeling Language Reference Manual", 1999

[20] Murata, St. Laurent, Kohn, "XML Media Types, January 2001, http://www.ietf.org/rfc/rfc3023.txt

[21] Grady Booch. Object-Oriented Analysis And Design With Applications, 2nd Ed. Benjamin Cummings. ISBN 0-8053-5340-2

[22] Lieberman, Joshua ed., Service Information Model DIRP, OpenGIS Document 02-055, 2002

[23] Fellah, Stephane, Keens, Steven, "DIRP: OGC Messaging Framework", 24-DEC-2002