Open Geospatial Consortium, Inc.

Date: 2008-09-12

Reference number of this document: OGC 08-079

Version: 0.9.0

Category: Discussion PaperDiscussion Paper

Editor(s): John R. Herring

OWS5: OGC® Web feature service, core and extensions

Warning

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

i. Preface

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

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

ii. Forward

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

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

Contents and Mandatory Requirements

Page

Introduction 8

1 Scope 9

2 Conformance 10

2.1 Introduction 10

2.2 Conformance class parameters 10

2.2.1 Introduction 10
2.2.2 Operation Conformance Classes 10
2.2.3 Interface Protocol parameter 11
2.2.4 Data Representation Language parameter 11
2.2.5 Schema Representation Language parameter 11
2.2.6 Query/Manipulation Language parameter 11

3 Normative references 13

4 Terms and definitions 15

5 Conventions 17

5.1 Abbreviated terms 17

5.2 UML notation 17

6 Data metamodel 18

6.1 Data uses 18

6.2 Data structures and representations 18

6.3 The use of discriminating labels 19

6.4 Type and member naming conventions 19

6.5 General models 19

6.6 General feature model 19

6.6.1 Scoped Names, Local Names 19
6.6.2 Feature 21
6.6.3 Types, properties and property types 22
6.6.4 References 23
6.6.5 Feature Collections 25
6.6.6 Namespace concerns for features and properties 27

6.7 General URI model and HTTP function call variable 28

6.8 Common request parameters 28

7 WFS Core Class 32

7.1 Semantics 32

7.2 Core::Capabilities ... 32
7.2.1 Semantics ... 32
7.2.2 Request ... 32
7.2.3 Response ... 33
7.2.4 Protocols ... 41
7.3 Core::Feature operation ... 42
7.3.1 Introduction ... 42
7.3.2 Request ... 43
7.3.3 Response ... 52
7.3.4 Protocols ... 54
7.3.5 Response ... 57
7.4 Core::Property operation ... 58
7.4.1 Introduction ... 58
7.4.2 Request ... 58
7.4.3 Response ... 58
7.4.4 Protocols ... 58
7.5 Core::DescribeFeatureType operation ... 58
7.5.1 Introduction ... 58
7.5.2 Request ... 59
7.5.3 Response ... 59
7.5.4 Exceptions ... 59
7.6 Core::DescribeFilterModel operation ... 59
7.6.1 Introduction ... 59
7.6.2 Request ... 60
7.6.3 Response ... 61
7.6.4 Exceptions ... 61
8 WFS Data Maintenance extension ... 62
8.1 Introduction ... 62
8.2 LockFeature action ... 62
8.2.1 Request ... 62
8.2.2 <expiry> parameter ... 62
8.2.3 <lockAction> parameter ... 62
8.2.4 Response ... 63
8.2.5 <LockID> return parameter ... 64
8.2.6 Exceptions ... 64
8.3 Request Schema definition ... 64
8.3.1 LockID element ... 64
8.3.2 release Action attribute ... 65
8.4 Insert action ... 65
8.4.1 Encoding ... 65
8.4.2 <inputFormat> parameter ... 66
8.4.3 <srsName> parameter ... 66
8.4.4 <IDgen> parameter ... 66
8.5 Update action ... 67
8.5.1 Request ... 67
8.5.2 <Property> parameter ... 68
8.5.3 <Filter> parameter ... 68
8.5.4 <handle> parameter ... 68

8.5.5 <typeName> parameter 68
8.5.6 <inputFormat> parameter 68
8.5.7 <srsName> parameter 68
8.5.8 <action>, <context> and <position> parameter 68
8.6 Delete action 69
8.6.1 Schema 69
Annex A (Normative) Conformance Test Suite 70
A.1 Conformance Classes 70
A.2 Core or read 70
A.3 Data Maintenance extension or write 70
Annex B (informative) Common Variables Used 71

Figures

Figure 6-1: Name Space Class Diagram 21
Figure 6-2: Feature Class Diagram 22
Figure 6-3: Reference Class Diagram 24
Figure 6-4: Feature and property Collection Class Diagram 26
Figure 6-5: PropertyType and Schema....27
Figure 7-3 : Feature Request 44
Figure 7-4 : Query Actions Class Diagram 46

Tables

Table 2-1: Conformance Class Parameters and their variables 10
Table 6-1: Common parameters for WFS requests 28
Table 6-2: Operations possible within a WFS service....31
Table 7-1: Elements to describe feature types....35
Table 7-2: Query Actions on Features....38
Table 7-3: Feature Operation constraints 39
Table 7-4: Operation parameters 40
Table 7-5: Values for resultType attribute....51
Table 7-6: Example Values for outputFormat attribute 52
Table 7-7: Feature Parameters 55
Table 7-8: Example Values for the outputFormat attribute 59
Table 7-9: Values for the outputFormat attribute 60
Table8-1: IDgen Attribute Values 67
Table B 1: Common Variables used in URIs and HTTP function calls 71
Table B 2: Commonly used file types and extensions for use in REST requests....75

Introduction

A Web Feature Service (WFS) provides operations to create, modify, delete and query feature instances from an underlying data store. WFS operations are invoked using any HTTP-transportable distributed computing platform to send a request message to a service, which then processes the request and generates a response message. This specification defines the logical structure of these request and response messages. The content and format of a request or response messages depends on the operation being invoked however.

This International Standard defines eight general operations:

- Capabilities
  The Capabilities operation provides service-level metadata about the operations that a particular service instances offers and data-level metadata about the features types that the service instance offers.
- DescribeFeatureType
  The DescribeFeatureType operation generates an XML Schema document that defines zero or more of the feature types that the service offers.
- DescribeFilterModel
  The DescribeFilterModel operation generates an XML Schema document that defines zero or more filter models that the service offers. Filter models define feature properties that may be used in the construction of Filter or Query expressions.
- Feature
  The Feature operation allows feature instances to be retrieved based on client specified query constraints. The query predicates, encoded using a Filter expression, can include both spatial and non-spatial constraints.
- FeatureWithLock
  The FeatureWithLock operation behaves exactly like the Feature operation except that all the features instances identified by the operation are locked in anticipation of forthcoming Transaction operations.
- Property
  The Property operations supports requests to retrieve element instances by traversing internal references to them.
- LockFeature
  The LockFeature operation allows feature instances to be locked in anticipation of forthcoming Transaction operations.
- Transaction
  The Transaction operation allows persistent feature instances to be created, modified and deleted.

All Wed Feature Services offer the operations: *Capabilities, DescribeFeatureType, Feature* and *Property*. The remaining operations will be optionally provided in extensions.

1 Scope

This International Standard specifies the behavior of a service that provides transactions on and access to geographic features in a manner independent of the underlying data store. It specifies discovery operations, query operations and transaction operations. Discovery operations allow the service to be interrogated to determine its capabilities and to retrieve the application schema that defines the feature types that the service offers. Retrieval operations allow features to be retrieved from the opaque underlying data store based upon constraints on spatial and non-spatial feature properties defined by the client. Transaction operations allow features to be created, changed and deleted from the opaque underlying data store.

2 Conformance

2.1 Introduction

This specification defines conformance classes based on the operations that a web feature service implements, the structures that it supports and the protocols it uses.

In any statement of a requirement that is tested in a test case in the test suite, each mandatory, testable requirement is given a "Req" number and set off as distinct from the rest of the text in **red and underlined**. Any optional or untested suggestion to the implementation that appears in the text will be similarly set in **red and underlined** but not receive a "Req" number. Each test in the test suite will reference those mandatory requirements which it is to test. Each mandatory requirement will be in at least one test.

The framework, concepts, and methodology for testing, and the criteria to be achieved to claim conformance is specified in ISO 19105: Geographic information — Conformance and Testing.

2.2 Conformance class parameters

2.2.1 Introduction

Each conformance Class will be determined and named with four, possibly related variables, as indicated in Table 2-1:

Table 2-1: Conformance Class Parameters and their variables

Conformance Class Parameter	Variable Name
Operation Conformance Test Class (e.g. read (core), or modify(extension))	`<conformanceClass>`
Interface protocol (e.g.. KVP, SOAP, REST)	`<interfaceProtocol>`
Data representation language (e.g. XML, GML, GeoJson, SQL etc.)	`<dataLanguage>`
Schema representation language (XML Schema, DTD, SQL-DDL, UML/XMI, etc.)	`<schemaLanguage>`
Query/Manipulation language with possible Conformance Test Class	`<queryLanguage>`

References to these parameters in this standard can use either the full parameter name or the "< >" delimited "BNF" style variable name.

Req 1 **Conformance with this specification shall be checked using all the relevant tests specified in the Test Suite based on the parameters Conformance Class, Interface Protocol, Data Representation Language, Schema Representation Language, and Query/Manipulation Language**

2.2.2 Operation Conformance Classes

There are two values for the operation parameter `<ConformanceClass>`.

The first is covered in Clause 7, and is the core class included the capability to read the capabilities of the service, and to read the feature data of the opaque feature store associated to the WFS.

The second is covered in Clause 8, which extends the firs by adding functions to insert, update and delete data from the datastore.

2.2.3 Interface Protocol parameter

The Interface Protocol parameter in this standard is written as `<interfaceProtocol>`.

Req 2 **The Interface Protocol `<interfaceProtocol>` shall be one or more of KVP, SOAP and REST.**

Req 3 **The Interface Protocol `<interfaceProtocol>` shall be given a default value in the capabilities document.**

Req 4 **The Interface Protocol `<interfaceProtocol>` shall be listed in the capabilities document.**

Req 5 **All parameters in all protocol-specific implementations will be uniquely named so that a flattening of the parameters into a list of atomic values in unambiguous.**

This standard, all of its extensions, profiles and implementation will be subject to this last requirement.

2.2.4 Data Representation Language parameter

The Data Representation Language parameter in this standard is written as `<dataLanguage>`.

Req 6 **Any Data Representation Language `<dataLanguage>` shall be capable of representation any information that is consistent with ISO 19109's General Feature Model.**

Req 7 **The Data Representation Language `<dataLanguage>` shall be given a default value in the capabilities document.**

Req 8 **All supported values of the Data Representation Language `<dataLanguage>` shall be listed in the capabilities document.**

Req 9 **The Data Representation Language `<dataLanguage>` shall be capable of supporting anonymous types as required by the conformance class of the WFS service.**

2.2.5 Schema Representation Language parameter

The Schema Representation Language parameter in this standard is written as `<schemaLanguage>`.

Req 10 **The Schema Representation Language `<schemaLanguage>` shall be capable of representing the structure of data elements in the Data Representation Language to a degree sufficient for the Data Representation Language to perform its functions as required by this standard.**

Req 11 **The Schema Representation Language `<schemaLanguage>` shall be given a default value in the capabilities document.**

Req 12 **All supported values of the Schema Representation Language `<schemaLanguage>` shall be listed in the capabilities document.**

2.2.6 Query/Manipulation Language parameter

The Query/Manipulation Language parameter in this standard is written as `<queryLanguage>`.

Req 13 **The Query/Manipulation Language `<queryLanguage>` shall be capable of specifying anonymous types that would allow the service to construct appropriate Data Representation Language representations of the results of any actions supported by the conformance class of the WFS.**

Req 14 **The Query/Manipulation Language `<queryLanguage>` shall be capable of expressing complete representations of its accessible data in the Data Representation Language which are uses as the parameters for or the results of any actions supported by the conformance class of the WFS.**

Req 15 **The Query/Manipulation Language `<queryLanguage>` shall be capable of invoking any actions supported by the conformance class of the WFS.**

Req 16 **The Query/Manipulation Language `<queryLanguage>` shall be given a default value in the capabilities document.**

Req 17 **All supported values of the Query/Manipulation Language `<queryLanguage>` shall be listed in the capabilities document.**

The major functions of the query language are to SELECT, UPDATE, DELETE, or LOCK/UNLOCK feature collections and query feature collections. See Clause 6.6.2

3 Normative references

The following normative documents contain provisions, which, through reference in this text, constitute provisions of this document. For dated references, subsequence amendments to, or revisions of, any of these publications do not apply. For undated references, the latest edition of the normative document referred to applies.

[1] ISO 8601:2004, Data elements and interchange formats – Information interchange – Representation of dates and times

[2] ISO 19013: 2005, Geographic Information – Conceptual schema language

[3] ISO 19105: 2005, Geographic Information – Conformance and Testing

[4] ISO 19109: 2005, Geographic Information – Rules for application schema

[5] ISO 19119: 2005, Geographic Information – Services

[6] ISO 191361, Geographic Information - Geography Markup Language (GML)

[7] ISO 191431, Geographic Information - Filter Encoding

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

[9] IETF RFC 2046 (November 1996), Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types, Freed, N. and Borenstein N., eds., http://www.ietf.org/rfc/rfc2046.txt

[10] IETF RFC 2141 (May 1997), URN Syntax, R. Moats, http://www.ietf.org/rfc/rfc2141.txt

[11] IETF RFC 2396 (August 1998), Uniform Resource Identifiers (URN): Generic Syntax, Berners-Lee, T., Fielding, N., and Masinter, L., eds., http://www.ietf.org/rfc/rfc2396.txt

[12] IETF RFC 2616 (June 1999), Hypertext Transfer Protocol – HTTP/1.1, Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and Berners-Lee, T., eds., http://www.ietf.org/rfc/rfc2616.txt

[13] NAMESPACE, Namespaces in XML, World Wide Web Consortium Recommendation, Bray, T., Hollander, D., Layman, A., eds., available at http://www.w3.org/TR/1999/REC-xml-names-19990114

[14] XML INFOSET, XML Information Set, World Wide Web Consortium Recommendation, Cowan, J., Tobin, R., eds., available at http://www.w3.org/TR/xml-infoset/

[15] XML 1.0, Extensible Markup Language (XML) 1.0 (Third Edition), World Wide Web Consortium Recommendation, Bray, T., Paoli, J., Sperberg-McQueen, C.M., Maler, E., Yergeau, F., eds., available at http://www.w3c.org/TR/REC-xml

[16] XMLSCHEMA1, XML Schema Part 1: Structures, World Wide Web Consortium Recommendation, Thompson, H.S., Beech, D., Maloney, M., Mendelsohn, N., eds., available at http://www.w3.org/TR/2004/REC-xmlschema-1-20041028

---

1) To be published.

[17] XMLSCHEMA2, XML Schema Part 2: Datatypes, World Wide Web Consortium Recommendation, Biron, P.V., Malhotra, A., eds, available at http://www.w3.org/TR/2004/REC-xmlschema-2-20041028

[18] XLINK 1.0, XML Linking Language (XLink) Version 1.0, World Wide Web Consortium Recommendation, DeRose, S., Maler, E., Orchard, D., eds. Available at http://www.w3.org/TR/2001/REC-xlink-20010627/

[19] OGC 05-008, OGC Web Services Common Specification, OGC Document #05-008, Whiteside, A., ed., available at https://portal.opengeospatial.org/files/?artifact_id=8798

[20] OGG 05-010, URNs of definitions in OGC namespace, OGC Document #05-010, Whiteside, A., ed., available at http://portal.opengeospatial.org/files/?artifact_id=8814&version=1

[21] CGI, The Common Gateway Interface, National Centre for Supercomputing Applications, available at http://hoohoo.ncsa.uiuc.edu/cgi/

[22] W3C Recommendation (24 June 2003): SOAP Version 1.2 Part 1: Messaging Framework, http://www.w3.org/TR/SOAP/

4 Terms and definitions

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

4.1
client
software component that can invoke an **operation** of a **service**

4.2
coordinate reference system
set of mathematical rules for specifying how coordinates are to be assigned to points [ISO19111]

4.3
feature
abstraction of real world phenomena [ISO 19101]

4.4
feature identifier
identifier that can be used to access a particular feature

4.5
feature reference
scoped name that identifies a **feature**

4.6
interface
named set of **operations** that characterize the behavior of an entity [ISO 19119]

4.7
local name
name assigned by a **namespace** to an entity that can then be located by that **namespace** by using that name

4.8
lock
limitation or the act of placing that limitation against a data element placed by a client that prevents its modification by any requests except those from that same client

4.9
Multipurpose Internet Mail Extensions (MIME) type
media type and subtype of data in the body of a message and fully specifies the native representation (canonical form) of such data [IETF RFC 2045].

4.10
namespace
authority for assigning **scoped names** and accessing by those names which can be located by another **scoped name** called the namespace id or name

4.11
opaque
not directly accessible to a **client** application

Note Opaque data structures are normally accessed by functions supplied to the client application

4.12
operation
specification of a transformation or query that an object may be called to execute [ISO 19119]

4.13
request

invocation by a **client** of an **operation**

4.14
response
result of an **operation** returned from a **server** to a **client**

4.15
scoped name
combination of **namespace** name and **local name** that identifies an entity to which the **namespace** has assigned that **local name**

4.16
service
distinct part of the functionality that is provided by an entity through interfaces [ISO 19119]

4.17
service metadata
metadata describing the **operations** and **features** offered by a **service**

4.18
server
service instance
particular instance of a **service** [ISO 19119]

4.19
spatial reference system
same as **coordinate reference system**

4.20
traversal
using or following an **XLink link** for any purpose [XLINK 1.0]

4.21
Uniform Resource Identifier (URI)
unique identifier, usually taking the form of a short string or address that is used to identify the location of a resource [IETF RFC 2396]

NOTE The general syntax is `<scheme>::<scheme-specified-part>`. The hierarchical syntax with a namespace is `<scheme>://<authority><path>?<query>&`

4.22
unlock
act of removing the limitations of a previous **lock**

5 Conventions

5.1 Abbreviated terms

CGI	Common Gateway Interface
DAG	Directed Acyclic Graph
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
HTTPS	Secure Hypertext Transfer Protocol
IETF	Internet Engineering Task Force
KVP	Keyword-value pairs
MDA	Model Driven Architecture
MIME	Multipurpose Internet Mail Extensions
OGC	Open Geospatial Consortium
OMG	Object Management Group
OWS	OGC Web Service
OWS	Open Web Services (in OGC)
REST	Representational State Transfer
ROA	Resource Oriented Architecture
SOA	Service Oriented Architecture
SOAP	Simple object access protocol
SSL	Secure Socket Layer
UML	Unified Modeling Language
URI	Uniform Resource Identity
URL	Uniform Resource Locator
URN	Uniform Resource Name
WFS	Web Feature Service
XML	Extensible Markup Language

5.2 UML notation

UML diagrams and class descriptions used in this standard are consistent with UML 2 from the OMG.

6 Data metamodel

6.1 Data uses

Much of the discussion in this standard deals with abstract descriptions of data that will be realized in various representations formats depending on the parameter of conformance first defined in 2.2.4, Data Representation Language parameter. This is similar to the REST model in that a resource can have multiple equivalent representations; where the nature of that equivalence is not described. For the purposes of this standard, data will be transferred between client and server, including but not limited to information on:

1. Capabilities of the server,
2. Metadata about the content of the resources,
3. Metadata about the logical structure of the resources,
4. Query or Filters for creating, selecting, editing and deleting elements of a resource for client purposes,
5. Requests by the client to the server for resources, either existing or created by the server in response to the request, or
6. Responses from the server back to the client in answer to the above requests.

6.2 Data structures and representations

The implementation of data structures in Web Services will be some form of tagged text (like XML), or some text serialization of a programming language (like JSON). The following basic assumptions are made that allow the mapping of these abstract data structures into either UML or XML:

1. Each data element contains either a value (is a property), or a collection of other elements (is a class, or feature type).
2. Each element has either a global identifier (such as a URI or other scoped name) or a local identifier (such as a property name; which is scoped to the context of its container) that distinguishes it from all other elements within its same context.
3. The context of an element with a global identifier is globally universal
4. The context of an element without a global identifier but having a name, is the immediately containing element. Thus each element is the namespace for its subelements and each locally named element in unique in its local context.
5. An element without a name nor a global identifier, is considered a data value (data type in UML). Data types cannot be directly referenced but may be referenced by a namespace and local name, as described above.
6. A data value may be a single instance of a data type or an array of instances of a single data type. The significance of the order in the array of these instances is defined by closest non-data-type container.

This metamodel is consistent with the UML metamodel, and may be represented in UML diagrams.

This metamodel is consistent with XML Schema and may be represented by XML Schema instances as long as the mechanism for mapping to a UML model is defined in a manner consistent with the UML representation of the model.

In general in this standard, the inline text representations of data structures are given in a "boxless" UML structure, where:

1. The first line contains the stereotype of the classifier, the Name, and any Inheritance (specialization - "under" or realization - "realizes")
2. The remainder (if any) is set of in brackets "{" and "}" and contains member UML attributes, association roles and operations. Since members of a classifier are named and not ordered, the ordering of the member declarations in not semantically meaningful.
3. Cardinality is expressed on the "type-side" of any declaration consistent with UML 2, and is in square brackets "[" and "]", and consist of an extended integer (with the star character (*) being infinity), or a pair of extended integers (minimum and maximum) separated by an ellipsis "…" indicating every integer in between these two.

Multiply dimensioned arrays will have multiple "[—]" expressions, one for each dimension. Optional values will have one "[0…1]" dimension

6.3 The use of discriminating labels

In programming, there is a concept call the "discriminated union." Its content is one of several value types whose instances are labeled as to which type they are. The discriminator holds the semantics of the value, while the type holds the value interpreted by the semantics. This is similar to XML "choice blocks" where element names (the things in the "< >" braces) allow the parser to determine the meaning of the included values.

In most cases in this standard, data is assumed to be "discriminated" that is labeled in such a manner as not only its value is understood, but so is its purpose and semantics. Thus a request-block set of parameters holds in its "discriminators" the operation for which the parameters apply. Thus, a request block can be considered as simply yet another representation of the value that would be returned by the operation for which the request was constructed. For example, a feature query collection is first defined by its "selection query" and then realized by executing the query to explicitly list the features that satisfy the query. These two "representations" are logically equivalent and will, for example, be two different representations of the same resource in a RESTful ROA.

6.4 Type and member naming conventions

In the core presentation of the data structures, all "class names" will appear in UpperCamelCase, and all member names in lowerCamelCase. **In using Model Driven Architecture (MDA) and deriving other formats, it is advisable to use the naming conventions specific to the particular representation language that is being targeted**. This could involve using a different case structure, a set of prefixes or suffixes, or classifier types not used in the base UML. It could also involve using namespace prefixes.

Req 18 **When writing a profile of this standard for a particular environment, the mapping of naming conventions shall be well defined and well documented.**

The terms "well defined" and "well documented" are mathematical in nature. They in combination mean that there is a fully described and unambiguous mechanism to accomplish the required function; in this case the mapping of naming conventions from one to another and back again. Such "perfectly mapped" structures are said to be "isomorphic" (meaning "having the same structure") and are considered identical for all logical purposes.

6.5 General models

6.6 General feature model

6.6.1 Scoped Names, Local Names

A great deal of the functionality in this standard will depend on keeping track of information quanta. In the OGC feature model, consistent with the ISO 19109 General Feature Model (GFM), the most usual information atom is a feature or a feature property. In both cases, the standard expects them to be identifiable in a reasonable fashion. While the details of this can vary somewhat, it should be consistent with the resource model used in the World Wide Web, which uses a namespace hierarchy for URIs, URLs and URNs. Since this mechanism is the most common, any other mechanism used should map to it. The core of the abstraction is the "scoped name" which is a combination of the namespace identifier and a local name, which can then be used to navigate to the item referenced by the local name, through the namespace.

Req 19 **Scoped names (namespace plus local name) shall be universally unique identifiers.**

Req 20 **Within a namespace local names shall be unique identifiers.**

Req 21 **The values of members (attributes and operations) within an instance of a type shall be scoped to the containing element or object (the namespace of the member name).**

Req 22 **Local names within a schema definition are the same local names used with a representation or instance based on that schema for the data corresponding to the similarly named local element of the schema specification.**

Note This is not as complex as it sounds. A data definition of an element type in a schema names the subelement definitions using a local name. That same local name is used when identifying the instances of those definitions in instances of the original data definition. These scoped names are shifted from the schema namespace to the instance namespace, while retaining the same local name. This makes the mapping from schema to instance simply a matter of name matching. All reasonable mappings from data definition languages to data languages work in this manner, and a standardization target would be hard-pressed to be in violation of this requirement.

Req 23 **The namespace of the scoped name of a feature type or property type shall be the name of the schema or part of the schema in which the type is defined.**

Req 24 **The namespace of the scoped name of the value of a feature shall be name of the datastore or part of the datastore in which the "definitive" value of the feature is stored.**

Note Following this requirement means that mirrored, value-identical copies of the "definitive" value of a feature should carry the scoped name of the "definitive" value, not of the local value.

Req 25 **The namespace of the scoped name of an instance of a feature property shall be the name of the instance of the feature to which the instance of the property is associated.**

The "scoped names" have a locally unique part call the "local name" or simply "name" and are contained in a "namespace" that acts as a collector of local names into a larger context: Each namespace may be contained in yet a larger namespace. Most naming conventions have a notation for following this hierarchy, but it varies widely. This standard will normally use the "dot" notation of object languages in the classifier and below, and a "::" (double colon) for package name and above. This is consistent with UML 2. Other implementations are far more complex, such as the notation for the HTTP URL, which mixes and matches separators and ordering based on its own logic. Whichever logic is used there are the same common and overriding requirements; that **the hierarchical logic of the namespace model should be preserved**, and that **the parallelism of schema and data instances should be reflected in the common use of local names between parts of the schema and their corresponding parts of the instance data**. These suggested limitations are not mandatory because the control of the namespace structure of existing implementation may vary from this strict parallelism between schema and instance as is the case in XML schema which inserts a global element definition between types and element instances that often uses a shifting and unforced name shifting pattern between type and global element.

The UML diagram for namespaces is in Figure 6-1.

Req 26 **The mechanism for finding an information atom (resource, object, property) from its scoped name shall be well-defined and well documented as part of the implementation of the namespace.**

```
Class ScopedName
    {
    nameSpace  : NameSpace
    localName  : String     {alias(id)}
    }
```

```
Class NameSpace
    {
    id                          : ScopedName
    member (name : String)      : Object
    }
```

A namespace is a named object (usually in a larger namespace, but may be universal such as “http”).

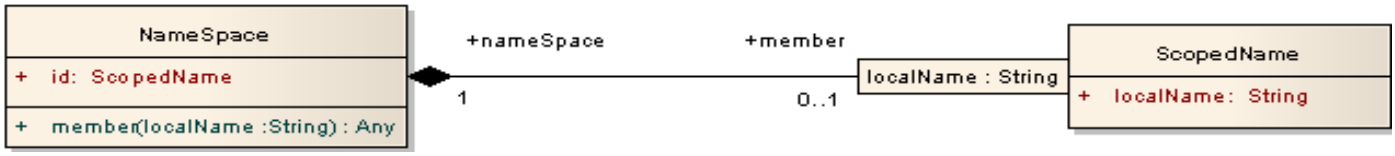

Figure 6-1: Name Space Class Diagram

Note The use of the “localName” qualifier on the member role in Figure 6-1 implies that the cardinality marked if for each value of localName. Thus this means that a local name in a scoped name in a namespace is unique. The member function of the `NameSpace` object means that the “local name” can be mapped to the object with that scoped name. This all implies that given access to the namespace, access to the members is by localName.

The target of the scoped name is not restricted to uniqueness, so any object can have multiple scoped names; but each scoped name is the one and only one instance with that value of `localName` in the `NameSpace`.

Req 27 **The mapping from scoped name to namespace is a child to parent mapping of a hierarchy, and shall form a directed, acyclic graph (DAG).**

Req 28 **There shall be a well defined operation or protocol for finding members within a namespace by their local name.**

For example in http URLs, the full name can be linked by the web to the resource it identifies, hence the “moniker” “universal resource location.”

6.6.2 Feature

Features are data elements with identity (local or global) which contain a collection of properties elements, each named locally with the feature store and feature type both supplying a viable namespace.

```
Class Feature under NameSpace
    {
    name                : FeatureName[0…1]
    type                : FeatureType[0..*]  {set}
    property            : Property[0..*]     {set}
    «readonly» lock     : Lock[0…1]          */ settable by the WFS service
    }
Class FeatureName under ScopedName
```

Note The lock on a feature is only non-NULL when a client request a feature lock.

```
Class Lock
    {
    «readonly» name     : ScopedName
    «readonly» expiry   : DateTime
    «readonly» owner    : ScopedName[0…*]
    }
```

A lock may be referenced by anyone by its name, but modification of lock information is only valid on the service side of the interaction and should be done from the client side by requests (usually through a query).

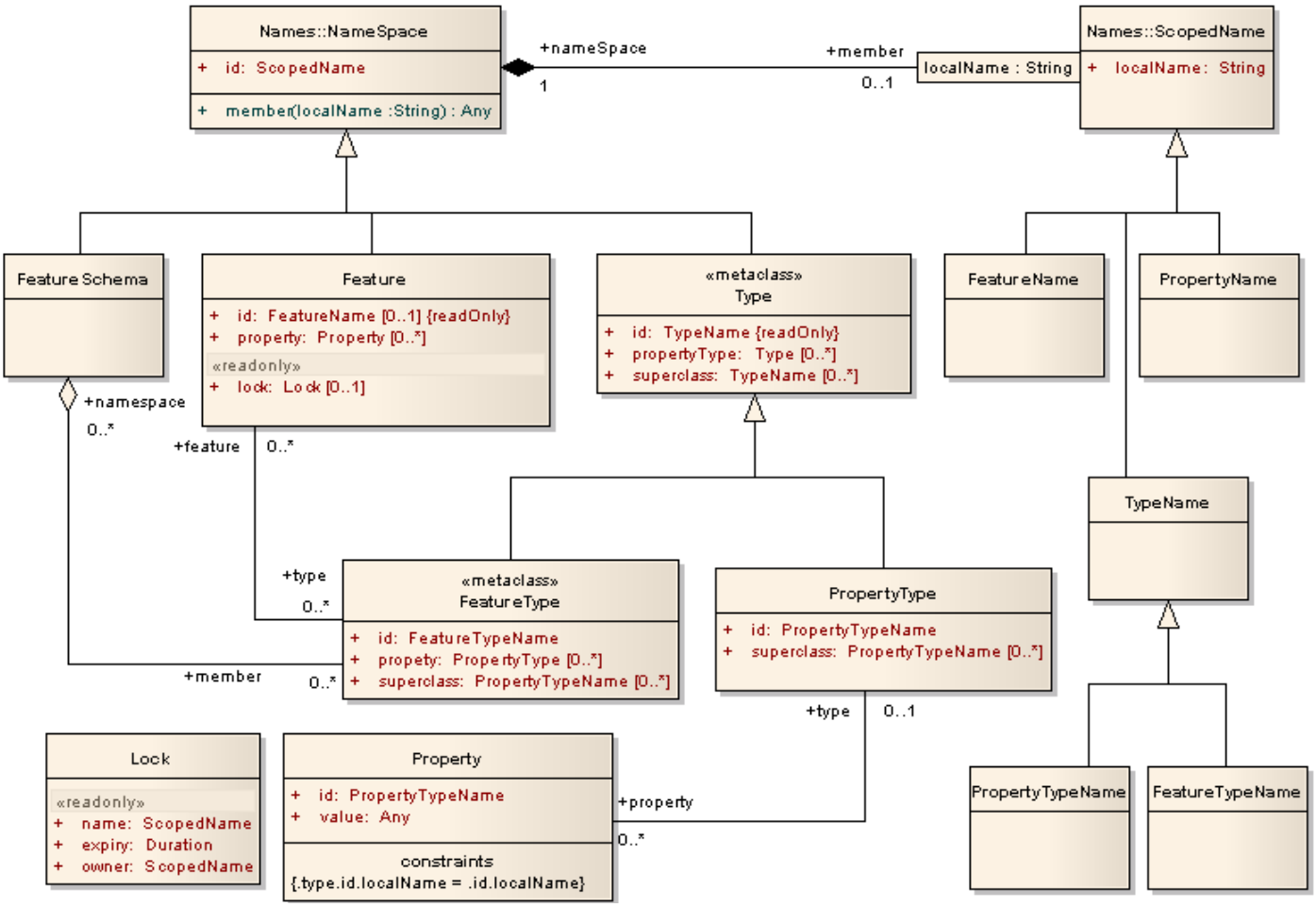

Figure 6-2: Feature Class Diagram

Req 29 **The feature's name `<featureName>` which is a scoped name shall be universally unique.**

Req 30 **A Feature "value" shall be independent of the order in which the properties of that feature are stored. Comparisons for equality shall not differentiate between the order in which a feature stores its properties.**

Req 31 **If a feature instance (value) is associated to a set of types, then the feature shall have all the properties required by each of that set of types.**

Note A property value may be an ordered set (such as an array or list), but the order of the named attributes within a feature has no semantical implications.

A Feature instance may realize multiple feature types. The decision to support multiple realization (each feature instance supporting more than one type other than that required by inheritance is usually a system-wide decision.

Note A Feature type is realized by a feature if it contains all required properties of the type, properly named; and makes the claim that it supports the feature type.

6.6.3 Types, properties and property types

In what is to follow, there are a set of primitive types (integer, character, bit, byte, etc.) that are the basis of the type system, which are defined in the basis of the data definition language used by the service. These primitive act as a "bootstrap" upon which the type system is built. There are also primitive constructors (array, list, set, bag, data structure, enumeration, etc.) that are assumed to also be available.

```
Metaclass Type under NameSpace
    {
```

```
    id              : TypedName
    propertyType    : Type[0…*]
    }
```

Note A metaclass has classes for instances.

```
Class FeatureType under Type
    {
    id              : FeatureTypeName
    propertyType    : PropertyType[0…*]
    }
Class TypeName under ScopedName
Class FeatureTypeName under TypeName
Class PropertyTypeName under TypeName
```

Req 32 **A property type which is unique and used in multiple feature types shall have the namespace of the schema in which it is defined. Such globally unique properties shall have the same semantics regardless of where they are use.**

Req 33 **A property type which is unique only within the feature type shall have the namespace of the feature type in which it is defined and used. Such locally defined properties shall only be considered valid within the context of the "namespace" of the feature type in which they reside.**

A FeatureType most likely resides in a FeatureSchema. A Feature most likely resides in a FeatureCollection. Its "id" will be a scoped name for the representation. Its name is a FeatureName referring to the Features "home datastore", i.e. the one holding the "official copy" of the feature instance. Thus it may be case that a particular representation of a feature may distinct id and name, which would mean that the value is a mirror of the official feature.

```
Class FeatureSchema under NameSpace
    {
    id              : ScopedName
    featureType     : FeatureType [0..*]
    }
```

Req 34 **The feature schema shall be the namespace for all property and feature types defined within it.**

A feature instance (value) may use multiple feature schemas, and if it has naming conflicts between its property values, it may use the schema namespace of those property values to differentiate between them.

6.6.4 References

The fully scoped name of a feature may be used as a reference. The nature of the operation of finding the feature given its scoped name is dependent on the choice of data representation. The UML diagram for these objects is in Figure 6-3

```
Class Reference
    {
    id  : ScopedName  / fully scoped  name of the referenced element
    }
```

A type-safe reference is a scope-name acting as a pointer to an instance of a particular type. The following type inherits the "id" reference from the class `Reference`, and adds a type reference to insure the type of the target of the reference.

```
Class TypedReference under Reference
    {
```

```
    type   : TypeName   */ scoped   name of the type of the reference element
    }
```

Note By inheritance from Reference, a Typed reference is a scoped name plus the name of the type that the reference should point to. If the target of the reference is not of that type, one of its subtypes (or derivations) or one of its realizations, then the typed reference is invalid.

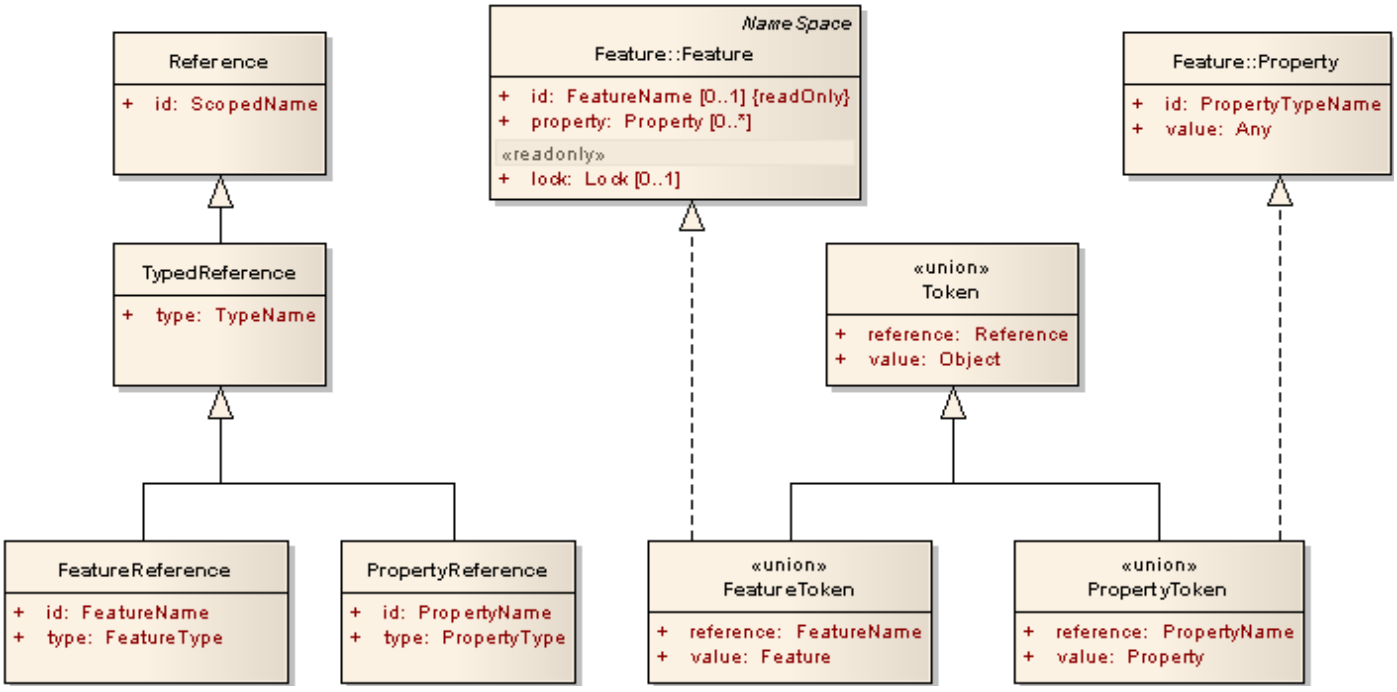

Figure 6-3: Reference Class Diagram

```
Class FeatureReference under TypedReference realize Feature
    {
    Reference  : FeatureName   */ the name of the feature
    type       : FeatureType   */ scoped   name of the feature's type
    }
```

Note A feature reference is simple a typed reference that has to refer to some specified type of feature (must be a subtype or realization of feature). In all cases where type is used in a name, the "type" is a metaclass of the correspondingly named instances.

```
Class PropertyReference under TypedReference realize Property
    {
    Reference  : PropertyName   */ scoped name of the property
    type       : PropertyType   */ scoped   name of the property's type
    }
```

A reference to an element with a scoped name may be used in place of the referenced element regardless of other circumstances. Service response data instances should carry local copies of any item that is referenced in their data element. The use of references without local copies would require additional service request to retrieve this data. The use of local reference within the response data has no such issue.

```
DiscriminatedUnion FeatureToken under Token realize Feature
    {
    id     : FeatureReference
    value  : Feature           alias feature
    }
DiscriminatedUnion PropertyToken under Token realize Property
    {
    id     : PropertyReference
    value  : Property           alias property
    }
```

6.6.5 Feature Collections

A feature collection is a by-value, by reference or by-selection logical container for feature instances. If the feature collection is by value or reference, it is said to be "explicit." It is specified by a query selection, it is implicit. A implicit feature collection is "realized" by making its contents explicit. Any modification of the features in the feature store that are referenced by the query makes the query-defined feature collection "stale" in that any explicit representation of the collection might be no longer valid.

```
Class FeatureCollection under Feature
    {
    member[0…*]   : FeatureToken
    }
```

Note The lock on a feature collection is inherited from feature. If its value is non-null the each feature in the collection shares that same lock. The granularity of locking is the feature.

```
Class FeaturTuple under FeatureCollection
    {
    member[0…*]   : FeatureToken  {ordered}
    }
```

Feature Tuples are ordered sets of features that are usually the result of a query filter involving relations involving sets of related feature.

```
Class QueryFeatureCollection under FeatureCollection
    {
    filter                    : Filter
    «readonly» lastComputed   : DateTime  */ settable only by the WFS
    }
```

If multiple queries are defined within a Query Feature Collection, then each query creates a selection set, which is one element of the Query Feature Collection. If only a single query is contained, then the collection is the result of that query. If no query is given, then the collection is the universal collection, i.e. the collection of all features (logically only limited by the datastore or WFS service to which it is associated).

```
DiscriminatedUnion FeatureCollectionToken under FeatureToken
```

Note Since a Feature Token is a typed reference, and Feature Collections are a type of feature, there is no distinction between the two types of token. A single feature can be considered a feature collection with one element.

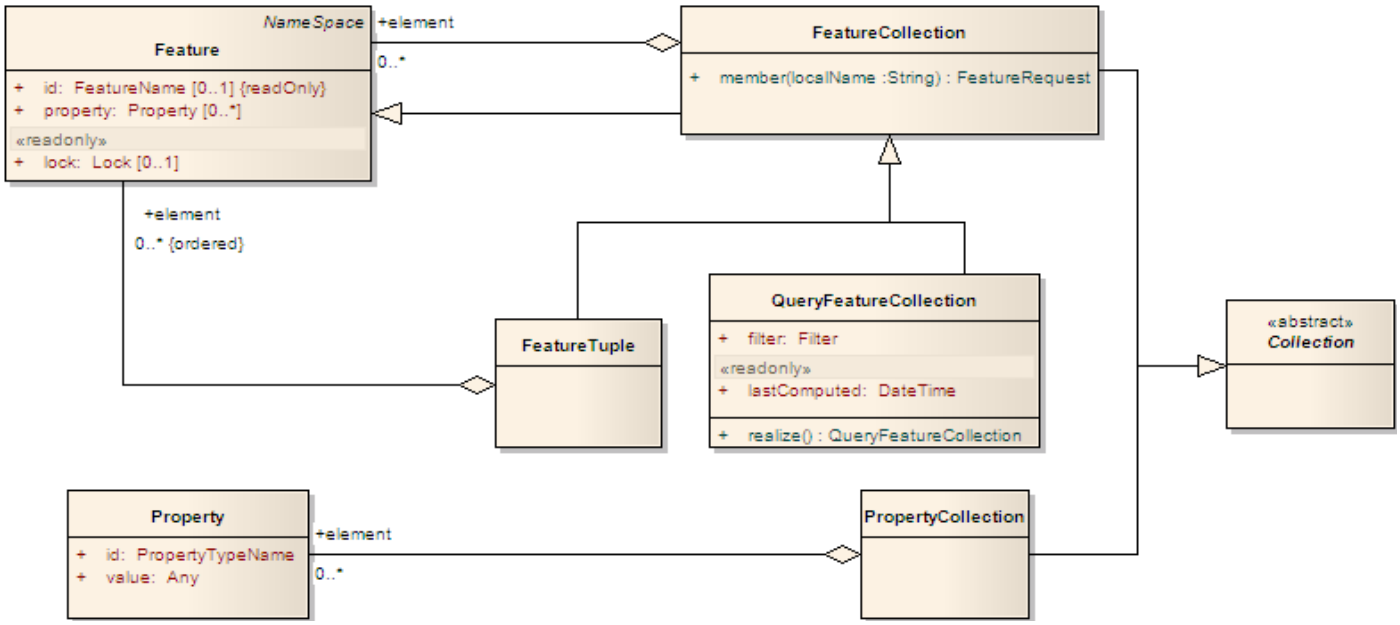

Figure 6-4: Feature and property Collection Class Diagram

A Query Feature Collection may lazily execute (calculate its explicit value from its query representation) so that the element array set of features may be unpopulated until it is accessed.

Req 35 **Access of the features of a query feature collection shall invoke its realization (to make it explicit) if it has not been evaluated since the last time a modification to tha datastore may have modified its implicit value.**

The auto-refresh of an query feature collection may be skipped by explicit client request. Features and Feature Collections are often used in passing parameters in and out of WFS service actions. In general, a reference is a good as the instance it reference for input or output parameters. In general, outputs or responses will normally be labeled copies of data store features, complete with scoped names.

Query Feature Collections give an alternate representation, and can be used in or as input parameters, can be used directly in HTTP "GET" calls to retrieve "concrete" representations of feature instances that satisfy a given query.

Properties are locally named holders of value. The type of that value will be defined in the feature schema, but there are no a priori restrictions on these property values.

```
Class Property
    {
    id      : ScopedName
    value   : Any
    Type()  : PropertyType[0…*]
    }

Class PropertyType
    {
    id         : PropertyTypeName  / scoped name of this property definition
    valueType  : TypeName          / scoped name of its contents
    }

Class PropertySchema under NameSpace
    {
    id           : ScopedName
    propetyType  : PropertyType[1..*]
    }
```

Req 36 **Any property shall be associated to a property type and a value type. The value of the opeations (or attribute to support that operation) of the property shall be the property type name. The value type shall be the type of the properties value.**

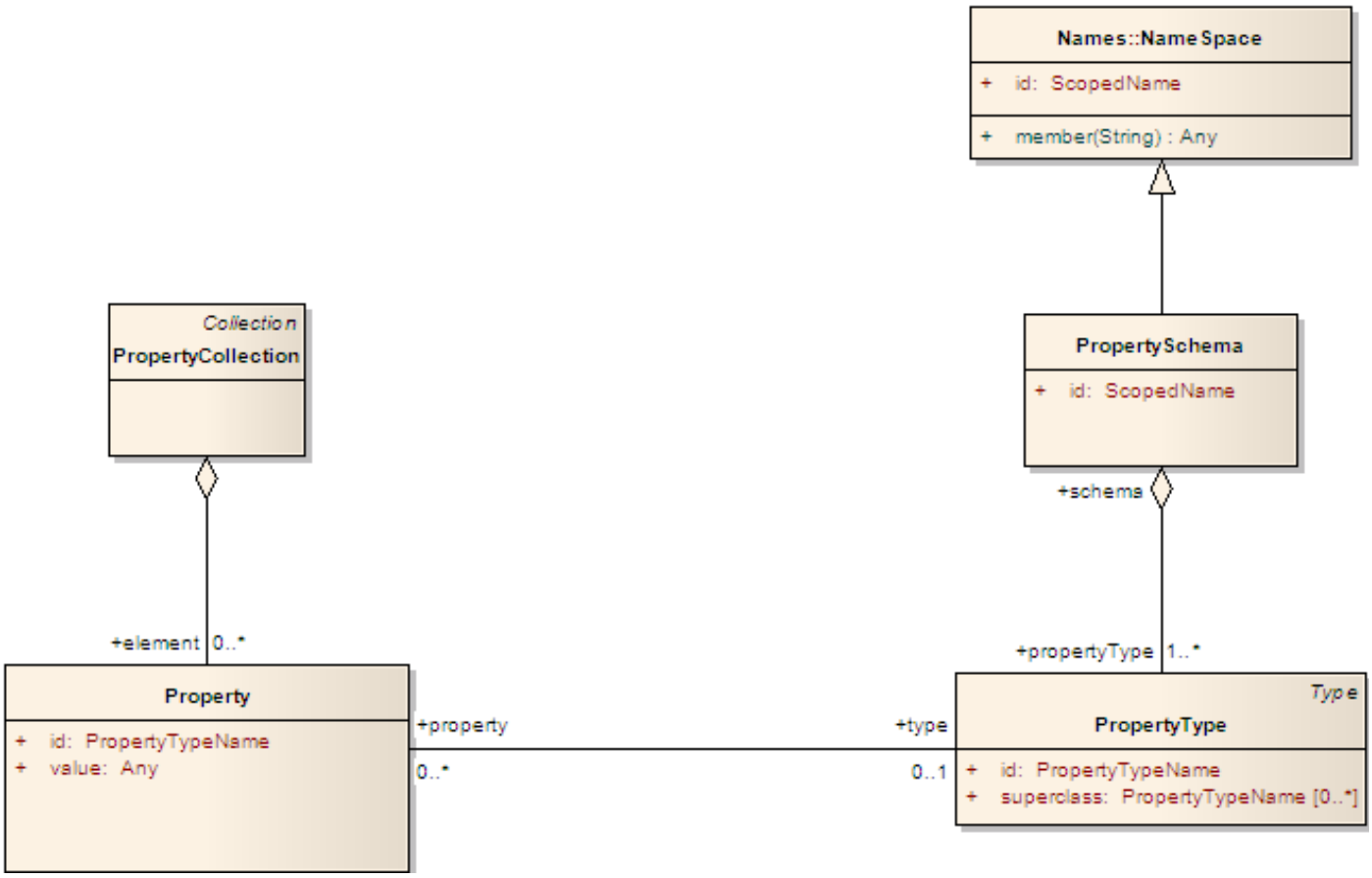

Figure 6-5: PropertyType and Schema

The mechanism to associate the various types to instances of those types is function of the `<dataLanguage>` and `<schemaLanguage>`.

6.6.6 Namespace concerns for features and properties

The namespace of a property instance (value) is always derived from the containing feature. The namespace of a property type (if it is from a feature type) is derived from the feature type or the schema (see requirements Req 32 and Req 33 in Clause 6.6.2) .

The property type of a feature may be limited by its schema to particular value types. This is not strictly a requirement, and is left up to the datastore of the feature to ensure any such limitations.

Each property value has at least one scoped name based on it containing property. So if the Feature instance has a id of `<featureID>`, then its contained property instances have a namespace of `<featureID>` and a local name equal to the property name. Using a dot notation:

- `<featureID>.<propertyName> = <propertyID>`
- `<propertyID>.value = the property's value`

Use of a reference within a property may break this pattern. Each feature is a "first class" entity, carrying the namespace of the datastore within which it primary value is kept.

Req 37 **A feature property having, as its value, the ID (scoped name) of another feature or of the property of another feature shall be interpreted as the current value is by definition the current value of the referenced data element.**

A single WFS can serve data out of multiple datastores, so it may serve feature, even of the same type, within a single response which have different namespaces associated to their scoped names.

6.7 General URI model and HTTP function call variable

If any of these values are lists, then the usual representation of lists of strings should be used as appropriate for the context and representation language. In a URI, a list is comma separated list with no additions whitespace. If whitespace is needed, the usually character codes may be used such as "%20" for a space, etc.

Req 38 **In an XML or similar document, a list of values shall be whitespace delimited.**

Req 39 **In a list giving the value of a KVP value shall be comma delimited, as is usual in URI representations.**

Req 40 **If reserved characters or whitespace is needed, it shall be properly escaped or represented as character codes of the appropriate type.**

6.8 Common request parameters

Table 6-1 describes parameters common to all WFS requests. Subsequent tables can redefine some of the facets of one or more of the parameters in this table.

Table 6-1: Common parameters for WFS requests

Parameter	Cardinality	DEFAULT	Description
`<host>` =http:/host/path/script	[1]		URL prefix of web service
version=`<version>`	[1]	2.0	Service version requested
service=`<service>`	[1]	WFS	Service type requested.
request=`<request>`	[1]		Type of WFS request.
namespace=`<namespace>`	[0..*]		Any namespaces needed each with a prefix.

NOTES: VERSION is mandatory for all operations except the Capabilities operation.

The request type holds information on the service being accessed and the request being made. Thus the URI use to address the service need not embed any of this information unless it is for convenience.

Req 41 **If the URI includes information that is also specified in the request, then the service shall use the information encoded in the request as opposed to that encoded in the URI. If this is impossible, the service shall raise an exception.**

Req 42 **If the URI includes information that could be but is not specified in the request, then the service shall use that included information as the default value for the request parameters left NULL.**

Thus the usual URI for a service request (except capabilities) may use the URI address:

`<host>/<service>`

Parameters are passed according to the `<interfaceProtocol>` as defined in Clause 2.2.2.

Additional parameters may be added to any request by subtyping. This puts the new values at the end of the list. Since in most data representations, values are associated to keys or tags, the order of the list of parameters should not cause in issue for a well-implemented service. This brings up the generalized "robustness principle" (also known as Postel's Law):

Be conservative in what you do; be liberal in what you accept from others.

Req 43 **A service operation receiving a request with an unambiguously labeled set of parameters in an order other than what the service expected, shall respond to that request in a manner identical with a request in which the order was what was expected.**

Req 44 **A service operation receiving an extended request structure shall ignore any parameters in that request that are not advertised in their capabilities document.**

Req 45 **If a service supports multiple interface protocols, the results of logically equivalent request from different protocols shall be functionally identical.**

The "version" parameter specifies the protocol version number and allows for negotiation. For the purposes of this standard, "2.0" is the default value of version. **In most cases, a service should support as many previous versions as possible.**

The "service" parameter specifies which of the available service types at a particular service instance is being invoked. The value WFS is used to indicate that the Web Feature Service should be invoked. For the purposes of this standard, WFS is the default required value of service.

The parameter REQUEST parameter indicates which of the web feature service operations to invoke. The possible values of the REQUEST parameters are given in Table 6-2.

The optional parameter "namespace" is included to support qualified feature and property names such as *myns:InWaterA_1M* where the prefix *myns* is bound to a particular namespace. The format of the parameter values for XML data encoding is as follows:

```
xmlns(prefix=EscapedNamespaceName)
```

where EscapedNamespaceName is the escaped URL of the namespace and is defined in CGI (see [21]). The *prefix* may be omitted to denote a default namespace. The "namespace" parameter may contain a list of namespace declaration values in order to bind all the namespaces being referenced in an URI-only (such as an HTTP GET) request.

The eventual intent is to make a WFS a full "datastore" interface to an on-line geographic feature datastore (or database), exposing all the functionality of a fully capable supporting database (if that is the implementation chosen. This would eventually involve:

- The retrieval and manipulation of data (CRUD — create, retrieve, update, delete) with as much data integrity protection as possible
- The retrieval and manipulation of the feature schema.
- Query and filtering capabilities to fully support the "feature collections defined by query filters" defined above as "query feature collections" (Clause 0)
- Equal support of all web service models, including at least KVP, SOAP and REST both SOA and ROA, in a logically consistent manner derived from the abstract models of feature data defined in OGC's abstract specification.

Req 46 **A WFS shall be prepared to encounter parameters that are not part of the specification (`<lagniappe>`) that may be part of a later extension or vendor-specific parameters.**

Req 47 **A WFS shall produce valid results even if additional parameters are missing or malformed.**

Req 48 **A WFS shall declare new parameters that it wishes to support within its Capabilities Document as delivered to a client.**

Clients may read the capabilities schema and formulate requests using any vendor-specific parameters advertised therein.

Table 6-2: Operations possible within a WFS service

Default Request Name	Allowable Alias	Access	Purpose
FeatureType	DescribeFeatureType	«read»	Get the schema information for one or more feature types. Currently the default is that the schema is readonly.
PropertyType	DescribePropertyType	«read»	Get the schema information for one or more property types.
Schema (future)	FeatureType, PropertyType	«read» «write»	Reserved for future use to support schema change functions.
FilterModel	DescribeFilterModel	«read»	Information on the "selection" Booleans qualifications possible when specifying a "query."
Feature	GetFeature	«read» «write»	Any operation on features, such as CRUD (create, retrieve, update or delete), lock and unlock. Alternatively from SQL syntax: INSERT, SELECT, UPDATE, DELETE, LOCK, RELEASE. The most common action is to get (retrieve or select). GetFeature will normally only be used for SELECT and LOCK. Locking granularity is currently the feature.
Property	GetProperty	«read» «write»	An alias for FEATURE, usually used when retrieving partial features, i.e. only some properties as opposed to complete feature objects. Supports the full "CRUD + Lock" semantics above, but may requires fully scoped names for properties such as `<featureName>.<propertyName>` to fully identify which properties are being targeted. GetProperty will normally only be used for SELECT. Locking by property is currently not supported.
Capabilities	GetCapabilities.	«read»	Access to service specific metadata about various functionalities and protocols. The capabilities document can be thought of as a static metadata document, but it is usually not since it can change with modifications of the feature schema. Capabilities may also be user specific; e.g. the capabilities of a server may depend on who the client is and what services and data he may have rights to access. This is a matter for Access Control or Digital Rights Management and will not be fully addressed in this standard.

7 WFS Core Class

7.1 Semantics

The Core class of the WFS standard defines a core set of requirements that all WFS service implementation must satisfy. Because CORE does not require specific data encodings or representations, but relies instead on abstract definitions of requests, responses and data structures, it is not directly testable until the parameters described in Clause 2 are specified.

7.2 Core::Capabilities

7.2.1 Semantics

The Capabilities or GetCapabilities operation is usually a static read-only data member that includes the information as required by OWS common in addition to specific WFS information on feature types and operations associated to the service. The decision to allow Capabilities in addition to GetCapabilities is to make the model more consistent with a Resource Oriented Architecture, without damaging the Service Oriented Architecture view of the WFS service. In any service operation used solely in a GET HTTP function may use a "Get" prefix without any change in interpretation.

7.2.2 Request

The Capabilities request is used to request a capabilities document from a web feature service.

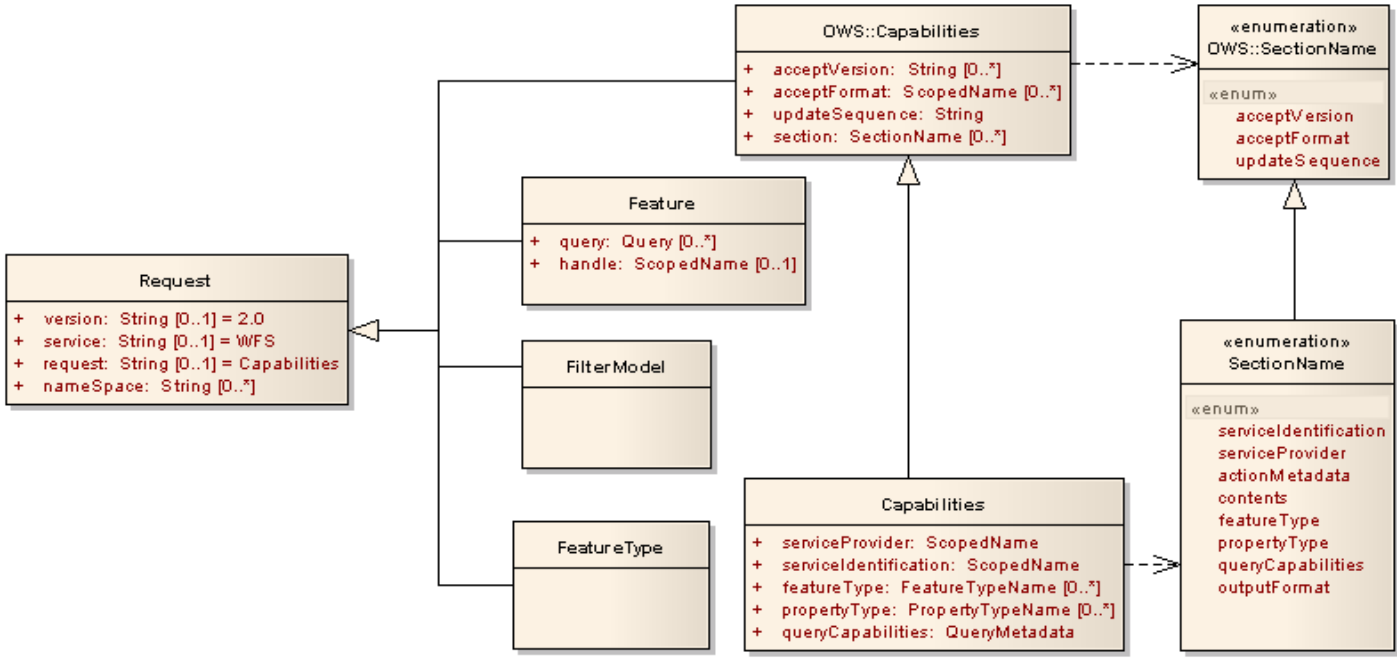

Figure 7-1: Capabilities Request Class Diagram

```
Class Request
    (
    version     : String[0…1]   = "2.0"
    service     : String[0…1]   = "WFS"
    request     : String[0…1]   = "Capabilities"
    namespace   : String[0…*]
    }
Class OWS::Capabilities under Request
    (
    acceptVersion    : String[0…*]
    acceptFormat     : ScopedName[0…*]
    updateSequence   : String[0…1]
```

```
    Section          : SectionName[0…1]


Class WFS::Capabilities under OWS::Capabilities
    {
    serviceProvider         : ScopedName[0…1]
    serviceIdentification   : ScopedName[0…1]
    featureType             : FeatureTypeName[0…*]
    propertyType            : PropertyTypeName[0…*]
    queryCapabilities       : QueryMetadata[0…1]
    }
```

Any WFS request data block may include information on the service, version being requested; any number of namespace declarations that may be needed for the request, and the request body itself.

7.2.3 Response

7.2.3.1 Data Definitions

The root of the response to a Capabilities request is the Capabilities element which is defined by the following:

```
Class OWS:CapabilitiesResponse under Response
    {
    acceptVersion    : String[0..*]
    acceptFormat     : String[0..*]
    updateSequence   : String
    section          : SectionName[0..*]
    }
Class WFS:CapabilitiesResponse under OWS:CapabilitiesResponse
    {
    serviceProvider         : ScopedName[0…1]
    serviceIdentification   : ScopedName[0…1]
    featureType             : FeatureTypeName[0…*]
    propertyType            : PropertyTypeName[0…*]
    queryCapabilities       : QueryMetadata[0…1]
    }
Enumeration OWS::SectionName
    {
    acceptVersion
    acceptFormat
    updateSequence
    }
Enumeration WFS::SectionName under OWS::SectionName
    {
    serviceIdentification
    serviceProvider
    actionMetadata
    contents
    all
    featureType
    propertyType
    outputFormat
    queryCapabilities
    }
Class Capabilities under GetCapabiliites
    {
    featureType          : ScopedNames[0..*]
    propertyType         : ScopedNames[0..*]
    filterCapabilities   : String[0..*]
    }
```

The base type, `OWS:CapabilitiesResponse`, is defined in the OWS Common Implementation Specification [OGC 05-008].

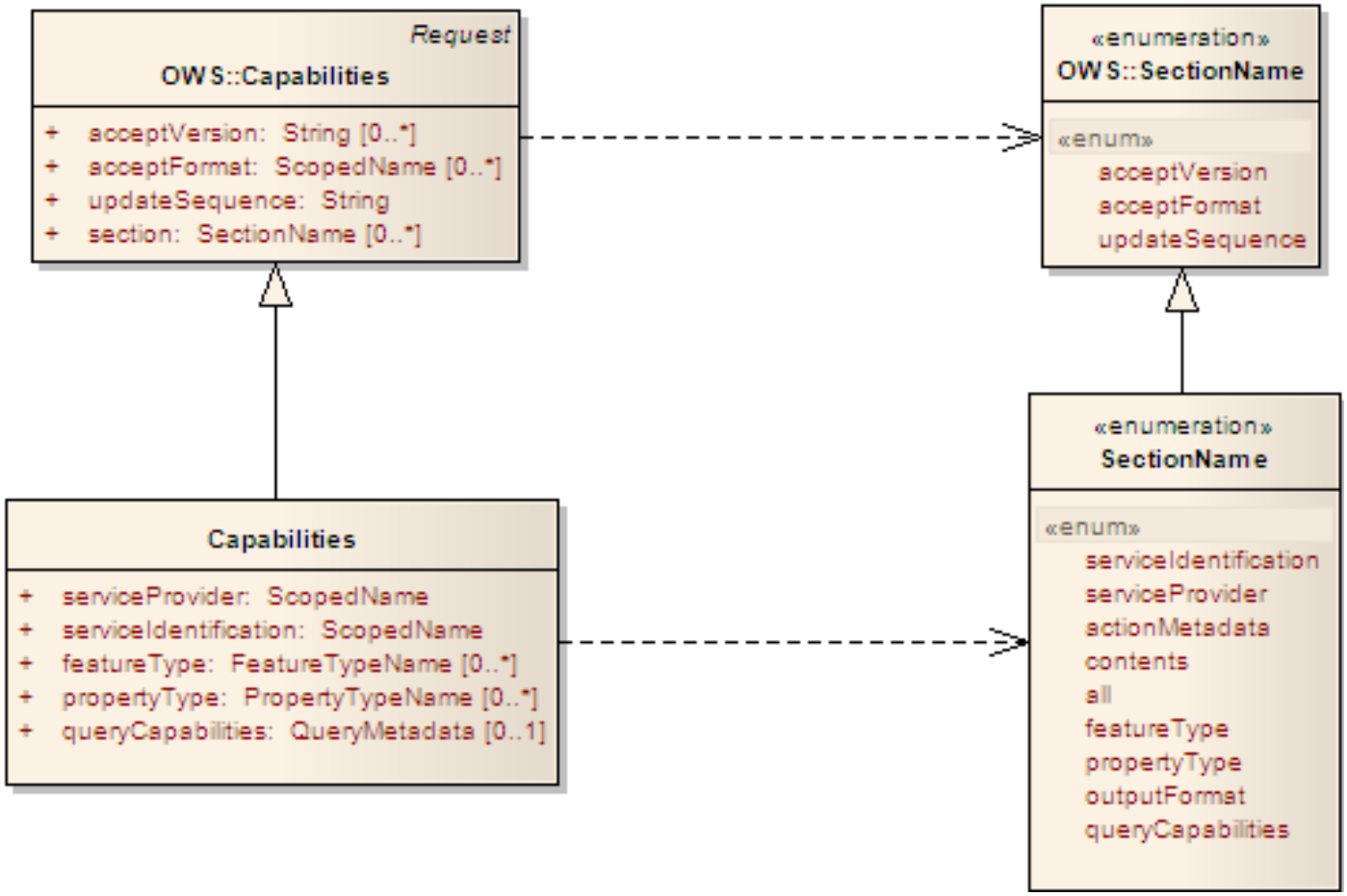

Figure 7-2: Capabilities

7.2.3.2 Service Provider section

Req 49 **The service provider section shall provide metadata about the organization offering the web feature service as defined in the OWS Common Implementation Specification.**

7.2.3.3 Service Identification section

Req 50 **The service identification section shall provide information about the web feature service itself, as defined in the OWS Common Implementation Specification.**

7.2.3.4 Feature Types section

Req 51 **The feature types section shall define the list of feature types (and operations on each feature type) that are offered by a web feature service. Any feature type used in a WFS response shall be listed in the feature types section of its Capabilities document.**

The purpose of the `<FeatureTypes>` element is to contain a list of feature types, that a WFS offers and defines the transaction and query operations that are supported on each feature type.

The following schema defines the Feature Type List element:

```
Class FeatureTypeList
    {
    action       : QueryActionName [0…*]
    featureType  : FeatureTypeDescription[1…*]
```

```
    }
Class Capabilities::FeatureTypeDescription
    {
    name                  : FeatureTypeName
    title                 : Title[0…*]
    abstract              : Abstract[0…*]
    keywords              : Keywords[0…*]
    SRS                   : SRSChoice
    outputFormats         : ScopedName[1…*]
    boundingBox           : BoundingBox[1…*]
    metadata              : MetadataReference
    extendedDescription   : PropertyTypeName[0…*]
    }
DiscriminatedUnion SRSchoice

    SRS    : SRSDescription
    noSRS  : NULL
    }
Class SRSDescription
    {
    defaultSRS  : SRS
    otherSRS    : SRS[0…*]
    }
Class Abstract under String

Class MetadataReference
    {
    document  : ScopedName[1..*]
    about     : ScopedName[1..*]
    }
Class QueryReference
    {
    document  : ScopedName[1..*]
    about     : ScopedName[1..*]
    }
Class BoundingBox
    {
    minimums  : Number[1..*]          */ minimum value for each coordinate
    maximums  : Number[1..*]          */ maximum value for each coordinate
    SRS       : SRSDescription[0…1]
    }
```

Table 7-1 lists the elements that may be used to describe each feature type in a <FeatureType> element:

Table 7-1: Elements to describe feature types

Element	Description
Name	The namespace-scoped name of the feature type. This element is mandatory.
Title	An unordered list of zero or more human-readable titles that briefly identify this feature type in menus.
Abstract	An unordered list of zero or more <Abstract> text elements. Each <Abstract> element is a descriptive narrative for more information about the feature type.
Keyword	The <Keyword> element contains a list of short words to aid catalog searching.

Element	Description
SRS.DefaultSRS	The <SRS> element indicates which spatial reference system shall be used by a WFS to express the state of a spatial feature if not otherwise explicitly identified within a query or transaction request. For example, if a Feature request specifies noSRS value for the <Query>.<srsName> attribute, any spatial properties of feature data satisfying the request shall be expressed using the <DefaultSRS> value. The SRS may be indicated using either the European Petroleum Survey Group form 'EPSG:<POSC Code>' or the URL format defined in URNs of definitions in OGC namespace (URN) [OGC 05-010]. The <DefaultSRS> shall not necessarily be the internal storage SRS used for the feature data, and therefore should not be interpreted as such. If the <DefaultSRS> is different from the internal storage SRS, then the WFS shall support a transformation between the <DefaultSRS> and the internal storage SRS. The effects of such a transformation shall be considered when determining and declaring the guaranteed data accuracy.
SRS.OtherSRS	The <OtherSRS> element is used to indicate other supported SRSs within query transaction and query requests. A 'supported SRS' means that the WFS supports the transformation of spatial properties between the <OtherSRS> and the internal storage SRS. The effects of such a transformation shall be considered when determining and declaring the guaranteed data accuracy.
SRS.NoSRS	The <NoSRS> element is used for feature types that have no spatial properties, and therefore no SRS whatsoever. It is not a requirement for features and feature collections to have spatial properties. The <NoSRS> element shall never imply, and therefore cannot be used for, semantics of "Unknown SRS". This element is used as an identifying label only, and therefore has no element or attribute content.
OutputFormat	The <OutputFormat> element is a list of MIME types indicating the output formats that may be generated for a feature type. If this optional element is not specified, then all the result formats listed for the Feature operation are assumed to be supported.
BoundingBox	The <BoundingBox> element is used to indicate the edges of an enclosing rectangle in decimal degrees of latitude and longitude in WGS84. Its purpose is to facilitate geographic searches by indicating where instances of the particular feature type exist. Since multiple latitude-longitude bounding boxes may be specified, a WFS may indicate where various clusters of data exist. This knowledge aids client applications by letting them know where they should query in order to have a high probability of finding feature data.
Metadata	A WFS may use zero or more <MetadataLink> elements to offer detailed, standardized metadata about the data in a particular feature type. The optional <about> attribute may be used to reference the aspect of the element which includes this <MetadataLink> element that this metadata provides more information about.

Element	Description
ExtendedDescription	A WFS may optionally add elements to the description of a feature type, without have to redefine the capabilities schema, using the <ExtendedDescription> element. The <ExtendedDescription> element contains one or more property values. In all cases, clients may safely ignore all or some of the extended descriptive properties.

7.2.3.5 Supports Property Types section

Req 52 **The supports property types section shall define all property types that a WFS is capable of serving. Any property used in a WFS response shall be listed in the property section of its Capabilities document.**

The following schema defines the <SupportsPropertyTypes> element:

```
Class Capabilities::SupportsProperty under Capabilities::PropertyCatalog

Class Capabilities::PropertyCatalog
    {
    Properties     : PropertyDescription[0…*]
    Abstract       : MetadataReference[0…*]
    Keywords       : Stings[0…*]
    outputFormat   : ScopedName[0..*]
    }
Class Capabilities::PropertyDescription
    {
    property   : ProptertyTypeName
    title      : String[0…*]
    abstract   : Abstract[0…*]
    keywords   : Stings[0…*]
    SRS        : SRSDescription[0…1]
    metadata   : MetadataReference
    }
Class Capabilities::ParameterDescription under Capabilities::PropertyDescription

Class Capabilities::OperationDescription under Capabilities::PropertyDescription
    {
    id          : LocalName              =”return”
    parameter   : ParameterDescription
```

The information values used to define each offer are identical to those used to describe feature types and are described in detail in.

7.2.3.6 Query Action Metadata section

Req 53 **The action metadata section shall provide metadata about the actions defined in this specification and implemented by the target web feature service being described. All data operations that a WFS service supports using this standard shall be listed in the operation metadata section of its Capabilities document.**

Req 54 **All services capable of creating a lock shall be capable of deleting that lock.**

Req 55 **All update, delete and select (retrieve) actions shall target existing features.**

Req 56 **All WFS services that are capable of query shall also be capable of creating query collections that shall be substitutable for any collection of the appropriate type in any action request.**

```
Enumeration QueryActionName
    {
    select  \ to be able to select feature or properties from the datastore
    insert  \ to be able to insert features or properties into the datastore
    update  \ to be able to change feature or property values in the datastore
    delete  \ to be able to delete features or properties from the datastore
    lock    \ to be able to lock features in the datastore for future actions
    }

Class Lock under QueryAction
    {
    lockID      : ScopedName[0…*]  \ name by which the lock will be identified
                                   \ usually defined by the service
    lockRange   : AllSome[0…*]     = "All"
    lockAction  : LockAction[0…1]  = "Lock
    expiry      : Duration[0…*]    = " 5 minutes"
    }

Enumeration AllSome
    {
    All   \ all feature locked or none locked
    Some  \ lock what can be locked
    }

Enumeration LockAction
    {
    Lock     alias Set
    Release  alias Unlock
    Renew
    }

Discriminated Union FeaturePropertySelection
   {
   feature         : FeatureTypeName
   property        : PropertyTypeName
   featureTuple    : FeatureTypeName[1…*]   \ tuple result of a join
   propertyTuple   : PropertyTypeName[1…*]  \ non-homogeneous tuple of properties
   mixedTuple      : TypeName[1…*]
   }
```

The contents of the operation metadata section are defined in the OWS Common Implementation Specification. This metadata includes the DCP, parameters and constraints for each operation.

The possible actions that may be specified in this section are described in Table 7-2.

Table 7-2: Query Actions on Features

Action	Alias	Description
<insert>	<create>	The <insert> <create> action is used to create new instances of a feature type.
<update>		The <update> action is used to change the existing state of a feature.
<delete>		The <delete> action is used to remove instances of a feature type from the data store.
<select>	<retrieve>	The <select> or <retrieve> action is used to filter only some features based on a "filter" or set of Boolean (true-false) criteria.
<lock>		The <lock> action is used to lock (limit the modification by others for a certain time) or unlock (release a lock) feature values. Rest of request is a "feature select" to determine those to be locked.

Transaction and query operations may be specified globally for all feature types or locally for each specific feature type contained in the <FeatureType> element. Globally specified transaction and query operations are inherited by every feature type contained in the <FeatureType> element and may be augmented by

specifying local transaction and query operations using the <Operation> element contained by each <FeatureType> element.. For example, the Query operation may be specified globally for all feature types contained in the <FeatureType>, but the Update operation may only be specified locally for a small number of feature types. If no transaction or query actions are defined either globally for all feature types of locally for specified features type, then the default operation, Query (indicating the query operation) shall be implied for all feature types contained in the <FeatureType> element.

Table 7-3: Feature Operation constraints

Constraint Name	Possible Values and/or Value Types	Description
<PostEncoding>	One or more of: "SOAP" "XML" “GeoJson” “GML” “KML” Etc.	Specifies the formats that can be used with HTTP POST transfer of operation requests. The value “SOAP” shall indicate that SOAP encoded operation requests can be handled, The value “XML” shall indicate that (bare) XML encoded operation requests can be handled. If the connect point URL is the same for all SOAP-encoded and bare-XML operation requests, this Constraint element shall be included in the ActionMetadata element. If the connect point URL is different for SOAP-encoded and bare-XML operation requests, this Constraint element shall be included in each Post element.
<DefaultMaxFeatures>	Integer value greater than zero.	Specifies the default value for the <maxFeatures> attribute of the <Feature> element. If the constraint is not specified then there is no limit on the number of features that a Feature request may return.
<DefaultLockExpiry>	ISO 8601 duration.	Define the default lock expiry duration (e.g. PT5S for 5 seconds). If the constraint is not specified then locks will be held indefinitely unless clear using some administrative function.
<PreservesSiblingOrder>	Boolean value; either "TRUE" or "FALSE"	Specified whether the WFS support sibling element ordering for actions. See clause 12.2.5 for an explanation of the significance of sibling element ordering.
<SupportsSubtype>	Boolean value; either "TRUE" or "FALSE"	Specifies that the service is capable to process the subtypesOf() function in query
<SortLevelLimit>	Integer value greater than zero.	Specifies that the service supports a set maximum of <SortProperty> elements to be processed in the Query portion of a Feature request

Constraint Name	Possible Values and/or Value Types	Description
<SupportsSpatialJoin>	Boolean value; either "TRUE" or "FALSE"	Indicates that the server has the capability to process joins with a spatial join predicate.
<SupportsJoins>	Boolean value; either "TRUE" or "FALSE"	Indicates that the server has the capability to process joins with non-spatial predicates.
<SupportsTransactions>	Boolean value; either "TRUE" or "FALSE"	Indicates whether transactions are atomically committed.

7.2.3.7 Filter capabilities section

The schema of the Filter Capabilities Section is defined in ISO document 19143.

Req 57 **The WFS shall support the operations advertised in the Filter Capabilities Section of its Capabilities Document.**

Req 58 **All filter operations required by this document and the Filter conformance class of the service shall be listed in the Filter Capabilities Section of its Capabilities Document.**

Req 59 **No conformant client shall use hidden filter capabilities of a service not listed in the Filter Capabilities Section of its Capabilities Document.**

7.2.3.8 Serves Parameter Types section

Req 60 **The serves parameter type section shall define the parameter types that are available from a web feature service that supports the Property operation. These types may be defined in a base schema, or in an application schema. Any parameter used in a WFS request shall be listed in the serves parameter section of its Capabilities document.**

The <Parameter> and <Constraint> elements are defined in the OWS Common Implementation Specification [OGC 05-008] and allow valid domain values and constraints to be defined globally for all operations or locally for specific operations that a web feature service offers.

The following table defines the parameter domains which may be defined in the capabilities document of a web feature service.

Table 7-4: Operation parameters

Operation Name	Parameter Name	Value Type	Values
All	<version>	String	List version number of the service supported.
Capabilities	<acceptFormat>	ScopedName	Includes types the server is capable of generating.
DescribeFeatureType	<outputFormat>	ScopedName	Includes any data format that the server supports.

Operation Name	Parameter Name	Value Type	Values
DescribeFilterModel	<outputFormat>	ScopedName	Includes any data format that the server supports.
Feature	<srsName>	ScopedName	List of SRS's that the WFS is capable of handling.
Feature	<IDgen>	String	Includes one or more of the values: "GenerateNew", "UseExisting" or "ReplaceExisting".
Feature	<inputFormat>	ScopedName	Shall include the value "text/xml; subtype=gml/3.2.0". May include any other string or MIME type that the server supports including previous version of GML.
Feature	<lockAction>	String[]	Shall include the values "ALL" and "SOME".
Feature	<outputFormat>	ScopedName	Includes any data format that the server supports.
Feature	<releaseAction>	String	Shall include the values "ALL" and "SOME".
Feature	<resultType>	String	Shall include the values "results" and "hits".
Feature	<resultType>	String	Shall include the values "results" and "hits".
Feature	<vendorID>	String	Any string that is used as a vendor identifier.
Feature Property	<outputFormat>	ScopedName	Includes any data format that the server supports.

In general the domain of a parameter is specific to a web feature service implementation. For example the allowed values for the <srsName> parameter are dictated by the particular transformations that a WFS supports. In some cases, however, a parameter domain is defined in this specification (e.g. <IDgen>). In such cases a web feature service may only restrict the domain.

The following table defines constraints, which may be specified by a web feature service in its capabilities document.

7.2.4 Protocols

7.2.4.1 KVP protocol

The standard way to retrieve a capabilities document is defined in OWS Common. Since the actually document is the same in all cases, the KVP user can use the REST protocols, which are equivalent. The most common protocols is as follows:

- GET <host>/WFS/Capabilities HTTP/1.1
- GET <host>/WFS/Capabilities/<sectionName> HTTP/1.1
- GET <host>/WFS/Capabilities?SECTION=<sectionName>& HTTP/1.1
- GET <host>/WFS/Capabilities/FeatureTypes/<featureTypeName> HTTP/1.1

- GET <host>/WFS/Capabilities?SECTION=FeatureTypes&
  FEATURE=<featureTypeName>& HTTP/1.1
- GET <host>/WFS/Capabilities/FeatureTypes?FEATURE=<featureTypeName>& HTTP/1.1

Note: The HTTP GET function does not support body content.

7.2.4.2 SOAP protocol

A SOAP request for capabilities can use the <host> URI as the recipient of the SOAP request. The SOAP body shall contain the following information in a format that the Capabilities document of the service specifies it will accept.

- <service>
- <version> which will default to unspecified
- <sectionName>, <featureTypeName>, <propertyTypeName>.
- <outputFormat> which will default to XML if not specified

7.2.4.3 REST protocol

A REST request for the capabilities document assumes that the capabilities request is a subelement of the service (no version is required), so the two following URIs will answer to a HTTP GET call with the capabilities document

- <host>/<service>/<capabilities>
- <host>/<service>/<version>/<capabilities>

This gives HTTP get request of the following format:

- GET <host>/WFS/Capabilities HTTP/1.1
- GET <host>/WFS/Capabilities/<sectionName> HTTP/1.1
- GET <host>/ WFS/Capabilities/FeatureTypes HTTP/1.1
- GET <host>/ WFS/Capabilities/FeatureTypes/<featureTypeName> HTTP/1.1
- GET <host>/ WFS/Capabilities/PropertyTypes HTTP/1.1
- GET <host>/ WFS/Capabilities/PropertyTypes/<propertyTypeName> HTTP/1.1
- GET <host>/ WFS/Capabilities/FeatureTypes/roads HTTP/1.1
  */to get data on the road feature type

7.3 Core::Feature operation

7.3.1 Introduction

Req 61 **The Feature operation shall return a selection of features from a data store in a format derived by negotiation.**

A WFS processes a Feature request and returns a response to the client that contains zero or more feature instances that satisfy the query constraints specified in the request.

Req 62 **The Feature operation shall be implemented by all web feature services**

Req 63 **The content of a <feature> element shall be a set of <property> elements, each of which partially describes the feature.**

The name of the property element indicates the property semantics, conventionally given in lowerCamelCase, such as *boundedBy* or *collarLocation*.

The value of a property is given either by the content of the property element, or by-reference as the value of a resource identified in the property element. The value of a property may be any element type, but is usually restricted by the schema of the feature-type. In some cases the value of a property of feature may be another feature, for example a School feature may have a property *frontsOn*, whose value is a Road, which may itself have further properties, etc. However, note that the properties of the second feature (the Road) are not properties of the first feature (the `School`) and it is an error to refer to them as such.

7.3.2 Request

7.3.2.1 Data Structures

The encoding of a Feature request is defined by the following abstract schema:

```
Class FeatureRequest under Request
    {
    query         : Query[0..*]
    maxFeatures   : Integer[0..1]     constraint {>0}
    resultType    : ResultType[0..1]  default = "results"
    outputFormat  : Format[0..1]      */ default format given in Capabilities
    }

Enumeration Feature::ResultType
    {
    results
    count
    }

Class Feature::FeatureResult under Result
    {
    results                       : Collection
    timeStamp                     : DateTime
    «readonly» numberOfFeatures   : Integer[0..*]     */ number of non-NULL elements
    «readonly» lockID             : ScopedName[0..*]
    }

Class Feature::Query
    {
    action      : QueryAction      = "select features"
    filter      : Filter[0…1]
    lockID      : ScopedName[0…1]
    lockRange   : AllSome[0…*]     = "Some"
    lockAction  : LockAction[0…*]  = "Release"
    }

DiscriminatedUnion Collection
    {
    feature   : FeatureCollection[0..*]
    property  : PropertyCollection[0..*]
    }
```

Req 64 **A Feature request shall contain one or more Query that describe a query action operation to be performed on a specific feature type.**

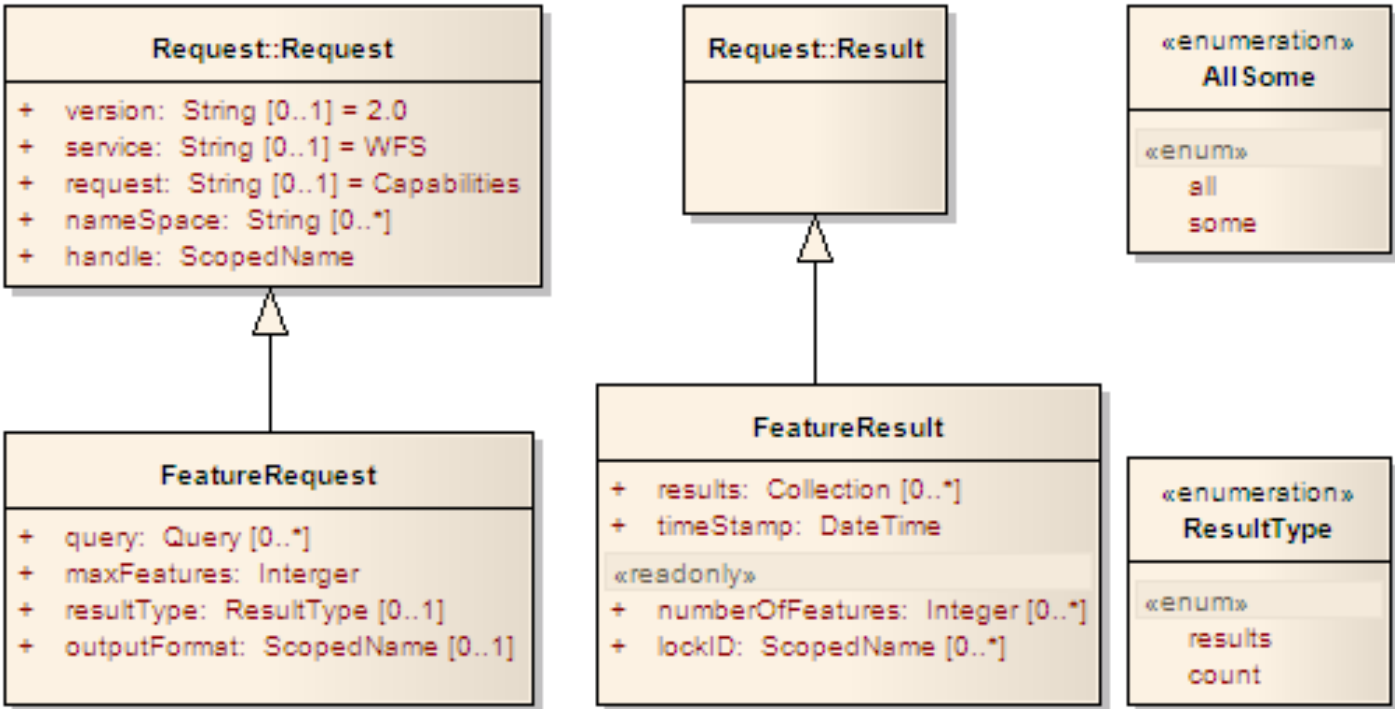

Figure 7-3 : Feature Request

```
«abstract» Class QueryAction

Class Select alias Retrieve, Projection Clause under QueryAction
    {
    ResultType    : FeaturePropertySelection  */ query response structure
    sourceAlias   : String[0…*]
    }
Class Insert alias Create under QueryAction
   {
   features       : FeatureCollection[1…*]
   localTypeName  : ScopedName[1…*]
   overWrite      : Boolean[0…1] = "False"
   }
Class Update alias UpdateProperty under Select
   {
   selection        : FeaturePropertyCollection[1…*]
   alias            : String[1…*]
   }
Class Delete under QueryAction
```

A Query action is defined be the following schema fragment:

```
Class Query
    {
    Feature         : Feature[0…*]  */ feature list to be acted upon
    featureType     : String[0…*]   */ feature type names
    featureAlias    : String[0…*]   */ featuer type aliases used in filter
    propertyType    : String[0…*]   */ property type names
    propertyAlias   : String[0…*]   */ property type aliases used in filter
    filter          : Filter[0…1]   */ coditions on features or propeties to search
    Action          : QueryAction   */ Action to take
    sortBy          : SortBy[0…1]   */ sorting instructions for response
    handle          : String[0…1]   */ identity reference for later use
    srsName         : ScopedName    */ SRS of the response
    }
DiscriminatedUnion DataElement
    {
    featureTypeName  : FeatureTypeName[0…*]  */ complete features to be retireved
```

```
    propertyName     : String                */ propeties to be retrieved
    queryFunction    : QueryFunction         */ function of the features
    constant         : ValueExpression
    }
Enumeration LogicalFunction
    {            */ Boolean valued function to perform on simple types
    and
    or
    not
    equals
    notEquals
    lessThan
    greaterThan
    }
«abstract» DiscriminatedUnion QueryTypeName
    {      */ Boolean valued functions to perform on element types
    type   : TypeName
    alias  : String
    }
DiscriminatedUnion LocalFeatureName Under QueryTypeName
    {     */ Boolean valued functions to perform on element types
    type  : FeatureTypeName
    }
DiscriminatedUnion LocalpropertyName Under QueryTypeName
    {         */ Boolean valued functions to perform on element types
    type      : PropertyTypeName
    Function  : QueryFunction
    }
Class QueryFunction
    {               */ Boolean valued functions to perform on element types
    functionName    : String
    operands[0..*]  : Value
    }
union QueryFunction
    {
    functionValue  : QueryFunction
    elementValue   : QueryElement
    }
Class LockFeatureRequest under GetFeatureRequest
    {
    lock        : Lock
    lockAction  : LockAction
    }
Enumeration LockAction
    {
    lock
    release
    }
Class Lock
    {
    handle  : ScopedName      */ defaults to system generated id
    expiry  : Duration[0..1]  */ default advertized in capabilities
    }
Enumeration LockAction
    {
    Some
    All
    }
```

Req 65 **Actions in a Feature request shall be processed in order in which they appear in the request.**

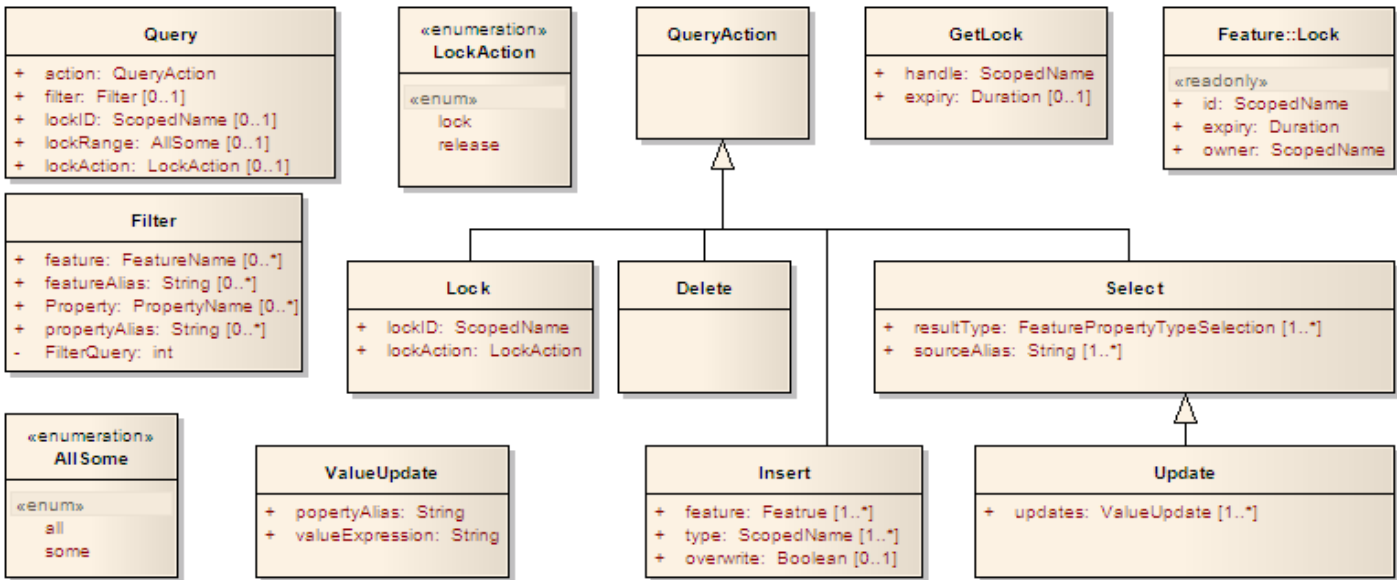

Figure 7-4 : Query Actions Class Diagram

7.3.2.2 Query elements

7.3.2.2.1 General structure

Each individual query packaged in a Feature request is defined using the `<query>` value. The `<query>` value defines which feature type(s) to query, what properties to retrieve and what constraints (spatial and non-spatial) to apply to the feature properties in order to select the valid feature set. The `<query>` value includes:

a filter (selection) clause, that describes the rule used to specify which instances of listed feature types will be reported using the syntax of the `<queryLanguage>`

a projections clause, that specifies the model for the response, in terms of the requested feature types and the properties of the features to be presented (given by the `<propertyName>`, or `<function>` values)

Req 66 **In the event that no selection clause is specified in a `<query>` value, a WFS shall respond by generating a response document that contains all instances of the explicitly requested feature types (subject to the value of the maxFeatures attribute.**

Req 67 **In the event that no projection clause is specified in a `<query>` value, then a WFS shall include complete instances of all of the explicitly requested feature types in its response.**

Req 68 **The schema of a feature-based query response shall be equivalent to the feature schema except that all properties are "optional" and are listed if only if requested in the `<query>`.**

Req 69 **The schema of a property-based query response shall be equivalent to the property schema from the `DescribeFeatureType`.**

7.3.2.2.2 Projection clause

7.3.2.2.2.1 Introduction

The purpose of the projection clause is to define the structure of the returned response as an anonymous type. This will contain some set of features or properties organized so that the results of the various filters and joins can be interpreted in the structure.

Req 70 **The response to a Feature or Property request shall contain a document with a root element of some subtype of Feature, Feature Collection or Property Collection which has the semantics and structure of an structured list of features or properties as defined in the projection clause.**

7.3.2.2.3 FeatureName elements and Aliases

Req 71 **The value of each `<featureTypeName>` element shall be a scoped name matching the name of one of the features or one of the properties of one of the features referenced in the typeName attribute of the parent Query element in the representation of the relevant feature.**

7.3.2.2.4 PropertyName elements and Aliases

Req 72 **The value of each `<propertyName>` element shall be a scoped name matching the name of one of the features or one of the properties of one of the features referenced in the typeName attribute of the parent Query element in the representation of the relevant feature.**

Req 73 **The names of the property elements shall be discoverable from the schema description of the feature types, being the names of the immediate child elements of the feature elements.**

Req 74 **The schema description shall be discoverable as the result of a DescribeFeatureType operation.**

There is typically some flexibility in the structure of the description of a feature type of interest, especially concerning the optional or mandatory nature of each property. For the purposed of the query response, the WFS service shall return only those properties requested, essentially treating the query specification as containing an anonymous type defined by its projection clause.

Req 75 **The `<propertyName>` elements shall be used to enumerate which of the properties shall be included in the response to a Feature request.**

Depending on the purpose of the request, the client may select complete features as defined in the schema or may use the projection clause to define anonymous types that do not match the defined features. In the latter case, it is the responsibility of the client to use the information with care since it will not be capable of being parsed by schema aware processes using the datastore schema.

Req 76 **A WFS client shall thus be prepared to deal with a situation where it receives a response that does not agree with its application schema, but does agree with the anonymous types defined in its query specification.**

In the case that a client wishes to assure itself that a particular feature is application schema compliant, it should request an unprojected version of that feature, or ask specifically for all mandatory feature properties. This it the preferred manner in which features are acquired with the intent to later update them. The local copies are precise mirrors of the datastore definitive versions, and therefore given proper locking procedures, an update using the modified local copies would have the precise semantics that would logically be expected by the client. **If the intent is to copy for later update, the client should lock the features in the datastore by including a lock request in this query.**

7.3.2.2.5 Function element

The <function> value may be used to apply a function, to the value of a feature property. The function shall be one of the functions that the WFS declares it supports in its filter capabilities [clause 20, ISO 19143]. Application of the function must result in a feature instance that is valid according to the application schema that the WFS advertises using the DescribeFeatureType operation.

Req 77 **If the application of a function would result in an invalid feature instance then the service shall raise an exceptions.**

7.3.2.2.6 Filter or Selection limitations clause

The <filter> value may be used to define constraints on a query. Both spatial and/or non-spatial constraints may be specified as described in ISO document 19143.

Req 78 **If no filter is contained within the query, then the query is unconstrained and the maximum number of feature instances shall be returned in the response.**

The value of the <filter> expression is based on values of elements within a representation of a feature instance. The mechanism for storing these values is dependent on the logical structure of the feature model and not its physical realization.

In the default case (i.e. no Filter Model) the model for the <filter> expression is identical to the model for the features reported in the response or that generated using the DescribeFeatureType request [clause 8].

However, if the <FeatureType> section of the Capabilities response [subclause 14.3.4] includes any <FilterModel> elements, then any <filter> element within the Feature request MUST indicate one of these filter models when the value of the typeName attribute on the <query> value is this feature type. If a filter model is specified, then the filter expression must include conditions for all properties according to the cardinality and value-space constraints indicated in the filter model. For example, a condition must be provided for all properties with minOccurs greater then zero; or if a set of properties are specified in a <sequence>, then conditions for all members of the sequence must be provided. Conditions may not be set for any property that does not occur in the filter model.

7.3.2.2.7 Join operation

A WFS may optionally support the join operation.

Req 79 **If a web feature service does support joins, it shall advertise this fact in its capabilities document.**

A join operation finds tuples (i.e. pairs, triples, etc.) of features, among a list of feature types, that satisfy some join condition specified using a filter expression [ISO 19143]. If the join condition is satisfied then that tuple of features is considered to be in the result set of the query. A join operation is encoded by:

listing the feature types to join using the typeName attribute and

specifying a join predicate using a filter expression [ISO 19143]

Web feature services that implement the join operation shall implement an inner join meaning that only feature tuples that match the join condition shall be returned in the result set. This means that all slots in any tuple returned will be populated, either by a feature or a feature reference depending on

the capabilities of the <outputFormat> and

the contends of the <query> specification

The following sequence of REST calls performs such a query using the contains operator.

- POST <host>/<service>/<version>/FEATURE HTTP 1.1

```
<query>
        <filter>
                <contains>Park.geometry Lake.geometry</contain>
        </filter>
        <alias>Park Lake</alias>
        <typeName>myns:Park myns:Lake</typeName>
        <propertyName>myns:Park myns:Lake</propertyName>
</query>
```

This operation returns a URI to a request posted to the appropriate FEATURE storage on the servers, say <request###>

- GET <host>/<service>/<version>/FEATURE/<request###> HTTP 1.1

The list of feature types to join is specified using the <typeName> attribute (e.g. typeName="myns:Park myns:Lake") on the <query> value. The join predicate is specified using the <filter> element and finds all pairs of myns:Park and myns:Lake features whose geometries satisfy the spatial operator used in the filter. The response to a request with a join is described in subclause 10.3.1.

A projection clause, selecting specific properties from each feature type being queried, may be specified subject to the discussion above.

Representation overrides for REST protocols may be done in the GET call by apply an appropriate "file suffix" to the resource name. The following examples will return the indicated types (as associated to the suffix in the services capabilities document).

- GET <host>/<service>/<version>/FEATURE/<request###>.gml HTTP 1.1
- GET <host>/<service>/<version>/FEATURE/<request###>.json HTTP 1.1
- GET <host>/<service>/<version>/FEATURE/<request###>.bxfs HTTP 1.1

7.3.2.2.8 Sorting

The <sortBy> value [ISO 19143] may be used to define content ordering in a query response. If no <sortBy> value is contained within the <query> value, then the query response is unordered and the order in which the feature instances are returned shall not be interpreted by the client as having any significance.

7.3.2.2.9 Parameter discussions

7.3.2.2.9.1 typeName parameter

The mandatory typeName attribute shall list the names of all feature types to be used in the query queried. Its value is a list of scoped names including namespace and local names (e.g. myns:School) or subtypesOf() functions returning a scoped name. Each typeName value shall match one of the feature types advertised in the service's capabilities document.

If the list contains only one typeName then the subtypesOf() function may be used to indicate that not only the specified feature type shall be queried but also all features of feature types whose object element are in the substitutable for the feature type mentioned and which meet the criteria expressed in the filter expression. "Substitutability" is defined by the schema language. In XML this is the substitution group, in object structures this is the inheritance structure. For example,

Example: typeName="subtypesOf(ns1:Vehicle)"

might along query the feature types ns1:Cars, ns1:Boats, etc …

If the list contains more that one scoped name this indicates that a join operation is being performed between the feature of the listed feature types. The join predicate is specified is specified using a filter as described in ISO 19143.

Example: typeName="ns1:FeatureTypeOne ns2:FeatureTypeTwo"

The `subtypesOf()` function shall not be used if a join operation is being performed.

Req 80 **In the event that a subtypesOf() function is present within a <query> value that encodes a join operation, the WFS shall raise an exception.**

7.3.2.2.9.2 <aliases> parameter

The optional <aliases> parameter may be used to specify a list of alternate names for the feature type and feature property names specified as the value of the <typeName> attribute. A feature type alias may be used anywhere the feature type name may be used within the context of a <query> action within a Feature request.

Req 81 **The server shall recognize an properly apply any alias defined by a client within its proper scope in a query.**

Req 82 **The number of list elements in the value of the <aliases> attribute shall match the number of feature type names in the value of the <typeName> attribute and shall be correlated 1:1.**

Req 83 **Each alias specified in the value of aliases attribute shall be unique.**

Req 84 **If the <aliases> attribute is used, an alias shall be specified for each feature type name listed in the value of <typeName> attribute.**

Aliases are typically used in queries, that perform a join operation, to support self-joins. That is a join of one feature type back to itself.

Example: `typeName="myns:Feat1 myns:Feat1" aliases="a b"`

In this example, the first feature type, `myns:Feat1`, is aliased to the name "a" and the second feature type, `myns:Feat1`, is aliased to the name "b". Thus, "`/myns:Feat1/boundedBy`" and "`/b/boundedBy`" is considered equivalent by the WFS. In this sort of query the alias is an independent role for the feature type, and the values taken on during query for both aliases will range over all instances of the feature type `myns:Feat1`.

7.3.2.2.9.3 <srsName> parameter

Req 85 **The optional <srsName> attribute of the <query> value shall be used to specify a specific WFS-supported SRS to be used for returned feature geometries. Its value may be the <DefaultSRS> or any of the <OtherSRS> values listed for the feature type in WFS capabilities document. If no <srsName> value is supplied, then the features shall be returned using the advertised <DefaultSRS> value.**

This attribute may have no meaning for responses that contain no spatial properties.

Any valid URI value may be assigned to the <srsName> attribute. However, in order to enhance interoperability, a WFS shall be able to process <srsName> attribute values with the following format models:

Examples: `EPSG:<EPSG code>`

```
http://www.opengis.net/gml/srs/epsg.xml#<EPSG code>
urn:ogc:def:crs:EPSG:version:<EPSG code> [OGC 05-010]
```

In these format models, the values `<EPSG code>` are placeholders for actual EPSG code values.

Example: `srsName="urn:ogc:def:crs:EPSG:6.3:26986"`

7.3.2.3 `<maxFeatures>` parameter

Req 86 **The `<maxFeatures>` shall be used to limit the number of explicitly requested features that a Feature request presents in the response document.**

The maxFeatures value applies to the entire result set (i.e. the result set generated by processing one or more Query actions contained is a Feature request) and the constraint shall be applied to the features in the order in which they are presented. In addition, feature members contained in an explicitly requested feature collection do not count – the requested feature collection counts as one feature. Furthermore, if more than one feature type name is specified as the value of the Feature.Query.typeName attribute then each resulting join tuple shall be counted as one feature. Once the maxFeatures limit is reached, request processing can terminate and the response document, containing at most maxFeatures features, may be presented to the client as a valid response. There is no default value defined for the maxFeatures attribute and the absence of the value means that all feature type instances in the result set are to be presented to the client if possible, given other limitations

7.3.2.4 `<maxSize>` parameter

Req 87 **The `<maxSize>` shall be used to limit the data volume of any response document. The limit shall be given in bytes (b), kilobytes (kb) or megabytes (mb) with the units explicitly given.**

7.3.2.5 `<resultType>` parameter

A WFS can respond successfully to a Feature request in one of two ways.

1. It may either generate a complete response document that contains the explicitly requested features (i.e. features specified via the `Feature.Query.typeName` attribute) that satisfy the query constraints in the request, or
2. it may simply return a count of the total number of features that satisfy the request.

Which of these two responses a WFS generates is determined by the value of the optional `<resultType>` attribute. The possible values for the attribute are summarized in the following Table 7-5:

Req 88 **The `<resultType>` shall determine the type of a result to a Feature request.**

Req 89 **If the value of `<resultType>` is "`results`" the return shall be a complete response document with explictly represented features that satisfy that request subject to the restrictions of `<maxFeatures>`.**

Req 90 **If the value of `<resultType>` is "`hits`" the return shall be a count of the number of features that satisfy the request.**

Table 7-5: Values for resultType attribute

resultType Value	Description

resultType Value	Description
<results>	The default value results indicates that a WFS shall generate a complete response that contains all explicitly requested features that satisfy the query constraints in the request.
<hits>	The value hits indicates that a WFS shall process the Feature request and rather than return the complete result set, it shall simply indicate the total number of explicitly requested features that would be contained in the response document. The total feature count is simply be the sum of the number of explicitly requested feature that each Query action in a Feature request would return. Only instances of the types specified as values of the *Feature.Query.typeName* attribute shall be included in total count. If more than one feature type is specified as the value of the *Feature.Query.typeName* attribute, then the resulting join tuple shall count as one feature instance..

7.3.2.6 <outputFormat> parameter

The optional <outputFormat> specifies the format of the response to a Feature request. Table 7-6 summarizes the example values for the <outputFormat> for generating some common types of response.

Req 91 **The value of <outputFormat> in a service request shall be the type of the data return part of the response**

Table 7-6: Example Values for outputFormat attribute

outputFormat Value	Description
text/xml; subtype=gml/3.2.0	This value indicates that an XML instance document shall be generated that validates against a GML 3.2.0 application schema.
text/xml; subtype=gml/3.1.1	This value is kept for backward compatibility and indicates that an XML instance document shall be generated that validates against a GML 3.1.1 application schema.
text/xml; subtype=gml/2.1.2	This value is kept for backward compatibility and indicates that an XML instance document shall be generated that validates against a GML 2.1.2 application schema.
GML2	Same as text/xml; subtype=gml/2.1.2
text; subtype=geojson	A GeoJson text file containing the requested information.
text/xml; subtype=bxfs/1.0.0	This value indicates that an XML instance document shall be generated that validates against the BXFS 1.0.0 XML Schema.

Req 92 **Any output format shall be usable by a client if and only if the appropriate value for <outputFormat> is advertised in the Capabilities document.**

A descriptive narrative should be included in the capabilities document for each output format listed there.

7.3.3 Response

Req 93 **The response to a Feature request with a <resultType> = <results> shall be a set of Feature Collections, each of which is one instance of a selected feature tuple; and a time stamp indicating the time at which this collection was constructed.**

Req 94 **If the number of features exceed the <maxFeatures> or <maxSize> value of the request or the default value of <maxFeatures> or <maxSize> then the last feature collection in the response shall be a reference to the next set of features in the aggregated result. The number of feature collection results shall contain at least one non-NULL instance, and no more than <maxFeatures> instances.**

The actual number of features or size of the response in the "partial results" case may be chosen by the server by any criteria for the optimization of the delivery of the response. The decision mechanism may be included in the capabilities document as part of the service's metadata, but this description is not required. The details of the implementation strategy for this mechanism is strictly an implementation decision.

Req 95 **The response to a Feature request with a <resultType> = <hits> shall be the count of the cardinality of a set of Feature Collections, each of which is one instance of a selected feature tuple.**

```
Class FeatureResponse under Response
    {
    features           : FeatureCollection[0..*]
    numberOfFeatures   : Integer[0…1]
    timeStamp          : DateTime[0…1]
    }
```

The content of the <FeatureCollection> value is controlled by the value of the <resultType> attribute on the <Feature> request.

Req 96 **If the specified value of the <resultType> attribute is <results> (the default value) then a WFS shall generate a complete response with the feature instances in the result set as the content of the <FeatureCollection>.**

Req 97 **If the value of the <resultType> attribute is specified as <hits>, a WFS shall generate a result value with no content (i.e. empty) but shall populate the values of the <timeStamp> attribute and the <numberOfFeatures> attribute.**

In the case where <resultType> = <hits> then the time stamp and number of feature values need not include a feature collection. In this way a client may obtain a count of the number of features that a Feature operation would return without having to incur the cost of transmitting the entire result set.

Req 98 **In the case of a Feature request with multiple <query> values a web feature service shall return a <FeatureCollection> corresponding to each <query> value in the Feature request. The outer <FeatureCollection> shall contain the component feature collections (one for each <query> value in the Feature request) and shall have the other <FeatureCollection> values as subelements.**

Alternatively, the WFS may return handles expressed as a URI for any or all of these feature collections. This URI can be navigated to and though as can any URI with simple HTTP calls.

Note that a feature member that satisfies more than one <query> value in the Feature request will be duplicated in the response. **Depending on the <outputFormat> value chosen, these repeating feature values may be given as local references to single full value.**

7.3.3.1 Join operation response

Req 99 **In the case of a Query action performing a join operation a WFS shall use the `<featureTuple>` (a subtype of feature collection) element in the response to report each set of features that satisfy the join predicate. The order of the features in the feature tuple will correspond to the order specified by feature type specified in the selection clause of the query filter.**

```
Class FeatureTuple under FeatureCollection
```

7.3.3.2 `<timeStamp>` attribute

Req 100 **The optional `<timeStamp>` attribute in a returned `<featureCollection>` shall be used by a WFS to indicate the time and date when a response was generated. The optional `<timeStamp>` attribute in a response may be used if there is no non-NULL feature collection to be returned.**

7.3.3.3 `<numberOfFeatures>` attributes

Req 101 **The optional `<numberOfFeatures>` attribute in a returned `<featureCollection>` shall be used to indicate the number of features that are in the response document. The optional `<numberOfFeatures>` attribute in a response may be used if there is no non-NULL feature collection to be returned.**

The same counting rules described in subclause 10.2.2.3 shall be used to set the value of the numberOfFeatures attribute.

7.3.3.4 Exceptions

Req 102 **In the event that a web feature service encounters an error servicing a Feature or another operation derived from it, in a situation not specifically covered in detail by this standard, the operation shall raise an appropriate exception as described in this standard.**

7.3.4 Protocols

7.3.4.1 KVP protocol

In KVP encodings, the request structure is flattened so that all parameters of a request use only their local names.

Req 103 **In using a KVP propocol to implement a service as defined in this standard, the base URL shall be that of the service, and the parameters shall follow in key-value pairs as defined in The Common Gateway Interface (CGI).**

The simplest Feature protocol is as follows

```
GET <host>/?SERVICE=<service>&REQUEST=Feature&VERSION=2.0
        &TYPENAME=<featureType>& HTTP/1.1
```

The following gets a property collection:

```
GET <host>/?SERVICE=<service>&REQUEST=Feature&VERSION=2.0
        &TYPENAME=<featureType>&PROPERTYNAME=<propertyName>& HTTP/1.1
```

The complete list of possible parameters are listed in Table 7-7.

Table 7-7: Feature Parameters

Feature Parameters	Cardinality	DEFAULT	Containing Type	Description
REQUEST=[Feature]	[1]		Feature Request	The name of the WFS request.
VERSION	[1]		Feature Request	
RESULTTYPE	[0,1]	results	Feature Request	The resulttype parameter is used to indicate whether a WFS should generate a complete response document or whether it should generate an empty response document indicating only the number of features that the query would return. A value of **results** indicates that a full response should be generated. A value of **hits** indicates that only a count of the number of features should be returned.
OUTPUTFORMAT	[0,1]	text/xml; subtype=gml/3.2.0	Feature Request	The output format to use for the response. **text/xml; subtype=gml/3.2.0** shall be supported. Other output formats are possible as well as long as their MIME type is advertised in the capabilities document.
MAXFEATURES=N	[0,1]		Feature Request	A positive integer indicating the maximum number of features that the WFS should return in response to a query. If no value is specified then all result instances should be presented.
FEATUREID (Mutually exclusive with FILTER and BBOX)	[0,1]		Feature Request	An enumerated list of feature instances to fetch identified by their feature identifiers.
BBOX (Prerequisite: TYPENAME) (Mutually exclusive with FEATUREID and FILTER)	[0,1]		Feature Request	In lieu of a FEATUREID or FILTER, a client may specify a bounding box as described in subclause 14.3.3.
PROPERTYNAME	[0,1]		Query	A list of properties may be specified for each feature type that is being queried. Refer to subclause **Error! Reference source not found.** on how to form lists of parameters. A "*" character may be used to indicate that all properties should be retrieved. There is a 1:1 mapping between each element in a FEATUREID or TYPENAME list and the PROPERTYNAME list. The absence of a PROPERTYNAME value indicates that all properties shall be fetched.
TYPENAME (Optional if FEATUREID is specified.)	[0,1]		Query	A list of feature type names to query. Multiple feature type name values shall be interpreted to mean multiple distinct queries analogous to multiple `<query>` values within a `<Feature>` value. Multiple feature type name values shall NOT be interpreted as a single query performing a join operation on the multiple feature types. Joins shall not be supported for KVP encoded WFS requests.

Feature Parameters	Cardinality	DEFAULT	Containing Type	Description
SRSNAME	[0,1]		Query	This parameter is used to specify a WFS-supported SRS that should be used for returned feature geometries. The value may be the Default SRS or any of the Other SRS values that a WFS declares it supports in the capabilities document. The SRS may be indicated using EPSG codes or the URL form defined in [ISO 19136]. If the parameter is not specified then the value of the Default SRS for the feature type being queried shall be used.
FILTER (Prerequisite: TYPENAME) (Mutually exclusive with FEATUREID and BBOX)	[0,1]		Query	A filter specification describes a set of features to operate upon. The filter is defined as specified in ISO 19143. If the FILTER parameter is used, one filter shall be specified for each feature type listed in the TYPENAME parameter. Individual filters encoded in the FILTER parameter are enclosed in parentheses "(" and ")".
SORTBY	[0,1]		Query	The SORTBY parameter is used to specify a list of property names whose values should be used to order (upon presentation) the set of feature instances that satisfy the query. The value of the SORTBY parameter shall have the form "*PropertyName [ASC|DESC][,PropertyName [AASC|DESC],...]"* where the letters ASC are used to indicate an ascending sort and the letters DESC are used to indicate a descending sort. If neither ASC nor DESC are specified, the default sort order shall be ascending. An example value might be: *"SORTBY=Field1 DESC,Field2 DESC,Field3"*. In this case the results are sorted by Field 1 descending, Field2 descending and Field3 ascending.

7.3.4.2 SOAP protocol

A SOAP request for capabilities can use the `<host>` URI as the recipient of the SOAP request. The SOAP body shall contain the following information in a format that the Capabilities document of the service specifies it will accept.

- `<service>`
- `<version>` which will default to unspecified
- `<outputFormat>` which will default to XML if not specified
- Feature request parameters
  - `<query>[0..*]`
  - `<queryString>[0..*]`
  - `<filter>[0..1]`
  - `<sortBy>[0..1]`
  - `<handle>[0..1]`
  - `<alias>[0..*]`
  - `<typeName>`
  - `<propertyName>[0..1]`
  - `<srsName>[0..1]`
  - `<maxFeatures>[0..1]`
  - `<resultType>[0..1]`
  - `<outputFormat>[0..1]`
  - `<featureID>[0..1]`
  - `<bBox>[0..1]`

7.3.4.3 REST protocol

A REST request for features can be done in two separate manners based upon the assumption of the structuring of the resources. assumes that the capabilities request is a subelement of the service (no version is required), so the two following URIs will answer to a HTTP GET call with the capabilities document

`<host>/<service>/<version>/<operation>` HTTP 1.1

This gives HTTP post request of the following format:

POST `<host>`/WFS/`<version>`/Feature HTTP/1.1
`< requestValue >`

Which will return a URI value something like

`<host>`/WFS/`<version>`/Feature/`<request###>`

Which will lead to getting the result using the following GET:

GET `<host>`/WFS/`<version>`/Feature/`<request###>`/results HTTP/1.1

GET `<host>`/WFS/`<version>`/Feature/`<request###>`/hits HTTP/1.1

In addition to the standard Feature requests using a query filter, there shall also be a direct feature and property access using the `<featureID>` of a particular feature. For example

GET `<host>`/WFS/`<version>`/Feature/`<featureID>` HTTP/1.1

Which returns a complete single feature i.e. "`<feature>` to `</feature>`".

GET `<host>`/WFS/`<version>`/Feature/`<featureID>`/`<propertyName>` HTTP/1.1

Which returns a complete feature property i.e. "`<property>` to `</property>`".

The first of which will return a complete feature with the `<featureID>`, and the second will return the value of a property of that feature of the given name; assuming one exist. If any URI reaches a "feature value" then augmentation of a `<propertyName>` value will produce another such GET URI. So, given the following has a schema that works this way ("road segments" are "linked" to "intersections" by the "startIntersection role" and "intersections" have a "location property"), the Get will return the location of the start intersection of "roadSegment 45":

GET `<host>`/WFS/`<version>`/Feature/roadSegment45/startIntersection/location HTTP/1.1

As discussed above, a REST user can modify the output format in the GET call by applying an acceptable file extension as defined in the services capabilities document. For examples, the following may work given the proper Capabilities document:

GET `<host>`/WFS/`<version>`/Feature/roadSegment45/startIntersection/location.gml HTTP/1.1

GET `<host>`/WFS/`<version>`/Feature/roadSegment45/startIntersection/location.json HTTP/1.1

GET `<host>`/WFS/`<version>`/Feature/roadSegment45/startIntersection/location.bxfs HTTP/1.1

7.3.5 Response

The response to any of these calls is a complete `<feature>` or `<featureCollection>` as asked for in the request contained in a `<FeatureResponse>`.

7.4 Core::Property operation

7.4.1 Introduction

The Property operation is a variant of the Feature operation where property collections are returned instead of feature collections. It uses Feature request parameters that limit the return type to a single property value or list of property values. The Property operation allows retrieval of features and elements by ID from a WFS. A Property request is processed by a WFS, and an document fragment, containing the result set, is returned to the client. The Property request provides the interface through which a WFS may be asked to traverse and resolve references to the features and elements it serves. This should be of no consequence to the user and the response should not be affected by the internal data representation of the WFS database.

7.4.2 Request

The Property request is defined by either of the following parameter sets:

- `<propertyName>` and `<featureID>` or
- a query that returns a set of properties through its projection clause.

The `<Property>` element is used to request the return an element with a ID value by an PropertyID. A Property element contains exactly one `<PropertyID>`. The value of the ID attribute on the PropertyID element is used as a unique key to retrieve the complex element with a ID attribute with the same value. The requested element could be any identified element of the GML data being served by a WFS, such as a feature, a geometry, a topology, or a complex attribute.

The outputFormat attribute defines the format to use to generate the result set. All possible formats supported by the service (including GML versions, non-XML and binary formats) shall be declared in the capabilities document.

7.4.3 Response

The format of the response to a **Property** request is controlled by the **outputFormat** attribute. The default value for the **outputFormat** attribute shall be **text/xml; subtype=gml/3.2.0.** This will indicate that a WFS shall generate a GML document fragment of the result set that conforms to ISO 19136.

The response to a **Property** request is the referenced GML element returned as an XML document fragment. This differs from the response to a Feature request, which returns a complete document containing a FeatureCollection.

7.4.4 Protocols

The protocols of a Property request are identical to that of GetFeature with the exception that the value of `<operation>` is now “Property.” All Property invocation are still valid if the `<operation>` value is replaced by GetFeature.

7.5 Core::DescribeFeatureType operation

7.5.1 Introduction

The `<DescribeFeatureType>` operation returns a schema description of feature types offered by a WFS instance. The schema descriptions define how a WFS expects feature instances to be encoded on input (via Insert and Update requests), how feature instances will be generated on output (in response to Feature and Property requests). The `DescribeFeatureType` operation shall be implemented by all web feature services.

7.5.2 Request

A <DescribeFeatureType> request element contains zero or more <TypeName> elements that encode the names of feature types that shall be described. If the content of the <DescribeFeatureType> element is empty, then all the feature types offered by a WFS shall be described in the response. The following XML Schema fragment defines the XML encoding of a DescribeFeatureType request:

```
Class DescribeFeatureType under Request
    {
    featureType    : ScopedNames[0…*]  = [ALL]
    outputFormat   : ScopedNames[0…1]  =
                                       [<schemaLangauge>]
    }
```

The default value for <featureType> is a complete list of all types of features in the schema associated to the WFS to which the particular client has access. The <outputFormat> attribute is used to indicate the schema description language that shall be used to describe feature types, the default value being the default or first <schemaLanguage> in the capabilities document.

Table 7-8 provides well known values for the <outputFormat> attribute that also ensures backward compatibility with previous versions of the OGC Web Feature Service specifications :

Table 7-8: Example Values for the outputFormat attribute

outputFormat Value	Description
text/xml; subtype=gml/3.2.0	This values indicates that the feature types offered by a WFS shall be described using a GML 3.2.0 application schema.
text/xml; subtype=gml/3.1.1	If a WFS implementation supports multiple versions of GML, this value shall be used to indicate that the feature types offered by the WFS should be described using a GML 3.1.1 application schema.
text/xml; subtype=gml/2.1.2	If a WFS implementation supports multiple versions of GML, this value shall be used to indicate that the feature types offered by the WFS shall be described using a GML 2.1.2 application schema.

7.5.3 Response

In response to a <DescribeFeatureType> request, a WFS service shall return a valid application schema that defines the feature types listed in the request, in the <schemaLanguage>. The document(s) returned by the <DescribeFeatureType> request may be used to validate feature instances generated by the WFS in the form of feature collections on output or feature instances specified as input for transaction operations.

7.5.4 Exceptions

In the event that a web feature service encounters an error servicing a <DescribeFeatureType> request, it shall raise an exception.

7.6 Core::DescribeFilterModel operation

7.6.1 Introduction

The <Query> of a Feature or Property request allow WFS clients to construct a request containing a filter expression using an arbitrary combination of tests on any or all of the properties of the requested feature type or types.

The Filter Model will often be a modification of the associated feature type, perhaps with certain properties suppressed, with additional properties added, or with a specified value-space for a property. However, this specification provides no specific constraints on the design of a filter model: it is the responsibility of the provider of the particular WFS instance to design filter models that enforce the desired business rules or

ensure the required performance protection, and if the filter model deviates from the response model, to devise a suitable mapping from the filter model to the response model.

The function of the Describe Filter Model operation is to generate a description of filter models for feature types offered by a web feature service implementation. The descriptions generated in response to this operation define how feature instances may be constrained in the filter expression of GetFeature requests. The only mandatory output in response to a Describe Filter Model request is a GML [ISO19136] application schema.

For the purposes of experimentation, vendor extension, or even extensions that serve a specific community of interest, other acceptable output format values may be advertised by a web feature service in its capabilities document. The meaning of such values in not defined by this International Specification. The only proviso in such cases is that WFS clients may safely ignore `<outputFormat>` values that they do not recognize.

7.6.2 Request

The XML encoding of a `<DescribeFilterModel>` request is defined by the following:

```
Class DescribeFeatureMode under Request
      {
      modelName     : ScopedName[0…*]
      outputFormat  : ScopedName[0…*]
      }
```

A Describe Filter Model element contains zero or more Model Name elements that encode the names of filter models that are to be described. If the content of the Describe Filter Model element is empty, then that shall be interpreted as requesting a description of all filter models that a WFS offers.

The output format attribute is used to indicate the schema description language that should be used to generate the description of filter models. Other schema languages may also be used to describe filter models as long as the MIME type value for the output format attribute is advertised in the capabilities document [clause 14]. Table 6 defines other values for the output format attribute that may be specified to request application schemas that conform to other versions of GML:

Table 7-9: Values for the outputFormat attribute

outputFormat Value	Description
text/xml; subtype=gml/3.2.0	This value indicates that a GML 3.2.0 application schema, using XML Schema, should be generated in response to a DescribeFilterModel request. This is the default values for the outputFormat attribute in the event that it is not specified.
text/xml; subtype=gml/3.1.1	This value indicates that a GML 3.1.1 application schema, using XML Schema, should be generated in response to a DescribeFilterModel request.
text/xml; subtype=gml/2.1.2	This value indicates that a GML 2.1.2 application schema, using XML Schema, should be generated in response to a DescribeFilterModel request.

For the purposes of experimentation, vendor extension, or even extensions that serve a specific community of interest, other acceptable output format values may be advertised by a WFS in the capabilities document [clause 14]. The meaning of such values in not defined in the WFS specification.

As specified by GML [ISO19136], the filter model schema definition is entirely at the discretion of the particular WFS implementation. The only caveats are:

Geometry must be expressed using the GML [ISO 19136] geometry description.

Spatial Reference Systems must be expressed as defined in GML [ISO19136].

The schema must be consistent with the GML feature model. This means that the filter model schema defines properties of a feature. The GML interpretation of this statement is that the elements nested immediately below the root element of a feature type define properties of that feature.

7.6.3 Response

In response to a `<DescribeFilterModel>` request, where the value of the `<outputFormat>` attribute has been set to text/xml; subtype=gml/3.2.0, a WFS shall generate a valid GML [ISO 19136] application schema that defines the schema of the filter model(s) listed in the request. The document presented by the Describe Filter Model request may be used to construct Filter clauses within a GetFeature request for the associated feature type(s).

7.6.4 Exceptions

In the event that a web feature service encounters an error servicing a Describe Filter Model request, it shall raise an exception as described subclause 7.4.

8 WFS Data Maintenance extension

8.1 Introduction

The **Data Maintenance extensions** are data transformation operations to be applied to feature instances under the control of a web feature service. This adds actions to the **Feature** operation via extensions to the query language actions. Using the **Feature** operation and these action clients can create, modify and delete features in the web feature services' opaque data store. The Data Representation Language, denoted as the parameter `<dataLanguage>` of the conformance class and as described in Clause 2.2.4, is used to pass feature and property information both into and out of the datastore.

Req 104 **All communication of feature and property information between client and server shall use the `<dataLanguage>`.**

The WFS implementation is responsible for any representational transformation required to do this server-side process.

Req 105 **When the transaction specified has been completed, a web feature service shall generate an response document, indicating the completion status of the operation.**

Req 106 **Web feature services that support the Data Maintenance extension actions shall advertise this fact in their capabilities document.**

8.2 LockFeature action

8.2.1 Request

The following parameters defines the contents of a LockFeature action:

1. A feature or feature collection subject to the lock (A Feature Request returning a feature or list of features, usually of the same type)
2. `<lock>` including a `<handle>` and a `<expiry>`

8.2.2 `<expiry>` parameter

The `<expiry>` value is used to set a limit on how long a WFS should hold a lock on feature instances in the event that a action is never issued that would release the lock. This limit is specified using an ISO 8601:2004 duration. The lock timer may be started once the entire lock request has been processed and the lock response has been completely transmitted to the client.

Req 107 **Once the specified `<expiry>` time period and elapsed (the lock expires), a WFS shall release the lock if it still exists and there is no ongoing or scheduled actions using that lock. Once all actions received befor the expiration of the lock have been processed, any remaining locks from this original request will be released.**

This means that the time limit is on the client to issue its request, not on the WFS to complete its operations.

Req 108 **Once a lock has been release, for any reason, then any further actions issued against that lock will not be honored, and the service shall return an exception message indicating that the lock is no longer valid.**

This default and maximum time for the `<expiry>` value should be specified by the service.

8.2.3 `<lockAction>` parameter

The `<lockAction>` value is used to control how feature locks are acquired.

Req 109 **A lock action of <All> indicates that the WFS shall acquire a lock on all requested feature instances. If all feature instances cannot be locked, then the operation shall fail, the service shall return an exception and no feature instances shall remain locked.**

Req 110 **If the lock action is set to Some, then the WFS shall lock as many of the requested feature instances as it can.**

Req 111 **The default lock action shall be All.**

Req 112 **A value of <Some> indicates that the WFS shall lock as many feature in the result set as possible. The response document shall only contain those features that were successfully locked**

Req 113 **If any features are successfully locked, the WFS shall return to the client a lock identifier (using the <lockID> attribute on the <FeatureCollection> element).**

Req 114 **The client shall use this value for subsequent actions on the locked features and to release the locks, until that time that the lock is released or has reached expiration.**

Req 115 **The result Type attribute shall be ignored in the case of a FeatureWithLock request. A WFS shall always generate a complete response (i.e. resultType="results") in response to a FeatureWithLock request.**

8.2.4 Response

A LockFeature Request creates a feature collection consistent with the GetFeature Request except that in general , there are two collections in the response, those successfully locked and those not locked. The following defines the contents of a **LockFeature** request:

```
Class LockFeatureResponse
    {
    lockedFeature  : FeatureCollectionWithLock[1..*]
    otherFeaure    : FeatureCollection[0..1]
    }
```

In response to a LockFeature request, a web feature service shall generate an XML document. This document will contain a lock identifier that a client application may use in subsequent WFS operations to operate upon the set of locked feature instances. The response may also contain the optional elements feature collections <lockedFeature> and <otherFeature> depending on the value of the <lockAction> attribute.

If the lock action is specified as <ALL> and all identified feature instances may be locked, a WFS shall respond with a <LockFeatureResponse> element that contains the <LockedFeature> element and no <OtherFeature> element (since either all identified feature instances may be locked or none at all). If some or all feature instances cannot be locked, a WFS shall response with an exception indicating that the lock request failed because some or all feature instances are locked by other clients.

If the lock action is specified as <SOME>, then the < LockFeatureResponse> element shall contain the <lockedFeature> and <otherFeature> elements.. The <LockedFeature> element shall list the feature identifiers of all the feature instances that were locked by the LockFeature request. The <OtherFeature> element shall contain a list of feature identifiers for the feature instances that could not be locked by the web feature 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.

If a lock request results in no features being locked, then a WFS shall respond with <LockFeatureResponse> document that contains a value for the lockID attribute but that contains neither a <LockedFeature> element nor a <otherFeature> element. In other words, an empty response. After the request is completed, the lockID shall be immediately released since no resources were locked. If that same lockID is used in a subsequent transaction an exception shall be raised, as described in subclause 7.4, since that lockID value for longer exists.

8.2.5 <LockID> return parameter

For a FeatureWithLock request, a WFS shall generate a response document that includes the lock identifier used to lock features in the result set. The lock identifier is encoded using the <lockID> attribute that is returned with the <FeatureCollection> value.

8.2.6 Exceptions

If a WFS does not implement the **LockFeature** operation then it shall generate an exception, indicating that the operation is not supported, if such a request is encountered.

In the event that a web feature service does support the **LockFeature** operation and encounters an error servicing the request, it shall raise an exception as described in subclause 7.4.

8.3 Request Schema definition

The encoding of a query <request> is defined by the following:

```
Class WfsRequest under Request
      {
      lockID            : LockID[0…1]
      operationRequest  : DataActionRequest[0…*]
      releaseAction     : LockAction[0…1]
      }

DiscriminatedUnion DataAction under Request
      {
      insert  : InsertRequest
      update  : UpdateRequest
      delete  : DeleteRequest
      lock    : LockRequest
      unlock  : UnlockRequest
      }
```

A <request> element may contain zero or more <Insert>, <Update>, or <Delete> elements that describe operations to create, modify or destroy feature instances. A WFS shall process <Insert>, <Update> and <Delete> elements in the order in which they are presented in the request. Subsequent update and delete actions, in a <request>, may operate on feature instances created by previous insert actions in the same transaction <request>. An empty <request> is valid but not very useful. Any lock releases occur at the end of the entire <request>.

8.3.1 LockID element

In the event that a transaction request operates on locked features, then the content of the <LockID> element shall be a lock identifier obtained from a previous FeatureWithLock or LockFeature operation. If a WFS does not support feature locking but is presented with a transaction containing the <LockID> element, the service shall ignored the <LockID> element.

If a WFS is presented with a transaction request that does not modify any locked features but contains a <LockID> element then the service shall ignore the <LockID> element.

If a WFS is presented with a transaction request that modifies locked feature and the **Transaction** request does not contain a <LockID> element or does not contain the matching lock identifier, the transaction operation shall fail and the service shall raise as exception as described in subclause 7.4.

8.3.2 release Action attribute

In a LockFeature and/or FeatureWithLock operations, the value of the releaseAction attribute controls how locked features are treated when a transaction request is completed.

A value of <ALL> indicates that the locks on all feature instances locked using the specified <LockID> should be released when the transaction completes, regardless of whether or not a particular feature instance in the locked set was actually operated upon. This is the default action if no value is specified for the releaseAction attribute.

A value of <SOME> indicates that only the locks on feature instances modified by the transaction shall be released. The others, unmodified but locked feature instances, shall remain locked using the same lock identifier so that subsequent transactions may operate on those feature instances. In the event that the releaseAction attribute is set to the value <SOME>, and an expiry period was specified on the <LockFeature> or <FeatureWithLock> elements using the expiry attribute, the expiry counter shall be reset to its initial value after each transaction unless all feature instances in the locked set have been operated upon.

For example, if a client application locks 20 feature instances and then submits a transaction request that only operates on 10 of those locked feature instances, a <releaseAction> of <SOME> would mean that the 10 remaining unaltered feature instances shall remain locked when the request completes. Subsequent transaction operations may then be submitted by the client application, using the same lock identifier to modify the remaining 10 feature instances.

8.4 Insert action

8.4.1 Encoding

The following declarations describe the <Insert> element:

```
Class Insert under Request
      {
      feature      : Feature[1…*]
      idgen        : IdentifierGenerationOption[0…*]  = "generateNew"
      handle       : ScopedName[0…1]
      inputFormat  : ScopedName[0…1]
      srsName      : ScopedName[0…1]
      }
Enumeration IdentifierGenerationOption
      {
      useExisting
      replaceDuplicate
      generateNew
      }
```

The <insert> element is used to create new feature instances in a web feature service's opaque data store. By default, the initial state of a feature to be created is expressed using GML [ISO 19136] and shall validate relative to a GML application schema generated by the DescribeFeatureType operation. Multiple <Insert> elements may be enclosed in a single Transaction request and multiple feature instances may be created using a single <insert> element.

8.4.2 <inputFormat> parameter

The <inputFormat> attribute may be used to support older versions of GML by indicating which version of GML is being used to encode the new feature to be created.

Req 116 **Any output format supported by a WFS server shall also be an allowable input format.**

Req 117 **Any input format supported by a WFS server shall also be an allowable output format.**

This will both be advertised in the Capabilities document as output formats. .

8.4.3 <srsName> parameter

The optional <srsName> attribute is used to assert the SRS of the incoming feature data, which can be useful if the incoming feature data does not have an SRS declared for each geometry. If the <srsName> attribute exists on an <Insert> element, its value shall be equivalent to the value of <DefaultSRS> or any of the <OtherSRS> of the relevant feature types.

Req 118 **If the SRS passed requested in any request is not supported, the service shall raise an exception.**

Req 119 **If the srsName is not specified in the request element, the service shall interpret this to mean that the feature data is given in the <DefaultSRS> unless an SRS is specified on the feature geometry.**

In this case, if the SRS for such a geometry is one of the <DefaultSRS> or <OtherSRS> values for the respective feature types, it will be transformed as required before insertion.

Req 120 **If SRS is not supported for the respective feature type, the entire transaction shall fail and the WFS shall raise an exception**

If atomic transactions are not supported by the underlying opaque data store, the WFS shall skip any feature with an unsupported SRS and continue. No exception shall be raised in this case.

The <srsName> attribute on the <Insert> element cannot be specified if the features types being inserted have no spatial properties, which is indicated via the <NoSRS> element in the capabilities document.

In response to an <Insert> operation, a WFS shall generate a list of identifiers assigned to the new feature instances as described in clause 13.3.

8.4.4 <IDgen> parameter

Feature identifiers may be generated by a WFS or specified by the client upon input using **ID** attribute values on inserted features and elements. A specific WFS implementation shall support one of these methods of assigning feature identifiers to new feature instances and may support all methods. The capability of the WFS shall be advertised in the capabilities document as described in clause 14.

The <IDgen> attribute may be used to indicate the method of assigning feature identifiers to new feature instances being inserted into a WFS's opaque data store. Table8-1 defines the possible values for the attribute and their meaning:

Table8-1: IDgen Attribute Values

IDgen Value	Action
<GenerateNew> **(default)**	The web feature service shall generate unique identifiers for all newly inserted feature instances.
<UseExisting>	The web feature service shall not generate new unique identifiers for the new feature instances being inserted into the WFS's data store but shall, instead, use the existing **ID** values specified with the new feature instances. If a new feature instance does not have a value specified for the **ID** attribute then the WFS shall raise an exception. If the **ID** value specified on a new feature instance already exists in the WFS's data store then the WFS shall raise an exception.
<ReplaceDuplicate>	The web feature service shall not generate new unique identifiers for the new feature instances being inserted into the WFS's data store but shall, instead, use the existing **ID** values specified with the new feature instances. If a new feature instance does not have a value specified for the **ID** attribute then the WFS shall raise an exception. If the **ID** value specified on a new feature instance already exists in the WFS's data store then the WFS shall replace the corresponding feature instance in the data store with the new feature instance with the corresponding **ID** value contained in the **Insert** action.

Req 121 **In the response to a <Insert> request, the feature identifiers for successfully inserted features shall be presented in the order in which the <Insert> operations were encountered in the request.**

These feature identifiers will have either originated from the input data or have been generated by the WFS according to the identifier generation policy described in Table 10.

8.5 Update action

8.5.1 Request

The following declares the <Update> element:

```
Class Update
    {
    property      : Property[0…*]
    typeName      : PropertyTypeName
    inputFormat   : ScopedName[0…1]
    handle
    srsName
    action        : UpdateAction
    context       : String
    position      : PropertyPosition
    }

Enumeration  UpdateAction
    {
    modifyProperty
    insertProperty
    deleteProperty
    }

Enumeration  PropertyPosition
    {
    beforeProperty
    afterProperty
    anywhere
    }
```

The <Update> element contains one or more <Property> elements that describe a value change to be made to one of the properties of the explicitly named feature type. Multiple <Update> elements can be contained in a single <Transaction> request so that multiple changes can be made to features of the same or different types.

8.5.2 <Property> parameter

Req 122 The <property> parameter shall contain a list of feature property instances contains the new value of each property to be updated. The property instance shall be consistent with the property type and feature type bing modified according to the feature schema for the datastore If this is not the case, the WFS implementation shall return an error message. .

8.5.3 <Filter> parameter

Req 123 The scope of the <Update> element shall be the defined by <filter> element defined in the <queryLanguage>. The <filter> element shall limit the scope of an update operation to an enumerated set of features or a set of features defined using query constraints. If the <filter> element does not identify any feature instances to work on, the update action shall have no effect and no error message shall be generated

8.5.4 <handle> parameter

Req 124 The <handle> attribute if set by the a client shall be used by the server as an identifier for any response to this request.

8.5.5 <typeName> parameter

Req 125 The value of <typeName> shall be one of the feature types listed in the feature type list found in the web feature service's capability document and shall be used to specify the feature type to be modified by this request.

8.5.6 <inputFormat> parameter

Req 126 The <inputFormat> parameter if set shall describes the format in the request for the data to be used in the update action . If not set the format shall be identifiable by its cotent such as headers or tags or associated metadata information.

8.5.7 <srsName> parameter

Req 127 The <srsName> parameter if set shall describes the SRS of the geometry elements in the data to be used fin the update action. If not supplied, the SRS should be the default or specifed in the data.

8.5.8 <action>, <context> and <position> parameter

The <Update> element contains one or more <Property> elements that specify the name and the update action to be applied to the value of the named feature property. The update action is specified by the <action> attribute on the <Value> element. The supported update actions are:

Req 128 If the action parameter is set to <modifyProperty> then the contained value shall be used in place of the current value of the selected property

Req 129 If the action parameter is set to <insertProperty>- then the contained value shall be inserted into a new the property.

Since the feature model does not assign any meaning to the order of properties in a feature intance, the position of the insert in the feature may be controlled by the server in accordance with its own internal requirements.

Req 130 If the action parameter is set to <removeProperty>- meaning to remove the specified child object of the named property.

Req 131 **If no <action> parameter is specified, the content of the <Value> element shall replace the current value of the property specified using the <Name> element.**

Req 132 **When all actions have been accomplished the feature instances modified shall be in valid representations of instances of their type; if not the server shall undo any modifications and report an error in the response to the client.**

8.6 Delete action

8.6.1 Schema

The following XML Schema fragment declares the <Delete> element:

```
Class Delete under QueryAction
      {
      typeName     : FeatureTypeName
      queryFilter  : Filter
      Handle       : String
      Name         : ScopedName
      }
```

Req 133 **The <Delete> action shall to remove some number of feature instances from a web feature service's opaque datastore. The <typeName> attribute shall explicitly name the type for feature instances to be deleted. The scope of a delete operation shall be constrained by the <filter>. In the event that the <filter> does not identify any feature instances to delete, the delete action will simply have no effect. This is not an exception condition.**

Annex A
(Normative) Conformance Test Suite

A.1 Conformance Classes

There are two conformance classes in the specification for each set of parameters described in Clause 2. These classes are:

1. Core or read
2. Data maintenance or write.

Specific conformance tests for Web Feature Services have not yet been determined for this version of the specification and will be added in a future revision of this specification.

A.2 Core or read

The first conformance class will test Req 1 up to but not including Req 104.

A.3 Data Maintenance extension or write

The second conformance class will test Req 104 and onward to the end of the document, Req 133.

Annex B
(informative)
Common Variables Used

In defining URI structures and parameter/message contents below and to support the Web Resource hierarchical model, the "BNF" style variables in Table B 1 are used throughout this standard. In the text, a variation of this name without the "<...>" and with spaces where the "camel case" changes can be used.

Table B 1: Common Variables used in URIs and HTTP function calls

Symbol	Format	Type	Definition
<version>	Text	URI part, localName, or parameter	The version of the service, which can be carried as part of the URI or as part of the passed parameters of the operations being called. Only operations or requests for "Capabilities" can omit the <version> variable.
<host>	Text	URI part, localName	The root of all of the URI's, including the HTTP prefixes, the actual host name and portal number and any other layers required locally to get the the Service container
<service>	Text	URI part	The name of the service, usually a 3-letter abbreviation, and here "WFS" always.
<<operation>Result>	Data representation language	return parameter	Value returned from an <operation>.
<result>	Data representation language	return parameter	Value returned from an unspecified operation.
<feature>	Data representation language	return parameter	A feature record in the format requested, including both it name and value (properties). Possibly defined anonymously by the query that produced it.
<featureCollection>	Data representation language	return parameter	A ordered list of features. This may be used in place of a feature
<featureTuple>	Data representation language	return parameter	A ordered list of features of a fixed length (number of entries not length in bytes). This will be an instance of a tuple type, where each offset is a specific feature type

Symbol	Format	Type	Definition
<featureSchema>	Schema representation language	return parameter	A collection of feature types.
<featureType>	Text	return parameter	A ordered list of property types included in a particular feature type.
<propertyCollection>	Data representation language	return parameter	A ordered list of properties.
<propertyTuple>	Data representation language	return parameter	A ordered list of a fixed number of property instances. Usually defined anonymously by the query that produced it.
<propertyType>	Schema representation language	return parameter	A ordered list of typed subelements included in a particular property type.
<lockID>	Text	return parameter	Optional in a returned result <featureCollection> that tells gives the user an key to the features locked in his behalf.
<numberOfFeatures>	Text	return parameter	The number of features contained in a <featureCollection>
<property>	Data representation language	return parameter	A feature-property record in the format of the response, including both its name and value
<timestamp>	Text	return parameter	The time that the associated information was created from its primary source.
<expiry>	Text	parameter or return parameter	Time a feature lock is asked to scheduled to expire
<featureName>	Text	parameter or return parameter	The scoped name of a feature type.

Symbol	Format	Type	Definition
<propertyName>	Text	parameter or return parameter	The name of a property type. This will use a "dot" notation if an object view is being used, or the appropriate path notation if some other feature representation is being used as the default view. For example, a service based on use of GML or some other XML based representation might use XPATH. If a <featureName> in included in a list of <propertyName>s, it means that the entire feature value of that type will be treated as a single property value.
<request###>	Text	parameter or return parameter	A particular request block for an operation given a "identity extension" by the server so that it can be used in a separate call (such as in REST, where the "put of the request" and the "get of the answers" are consecutive but separate HTTP function calls.
<<operation>Request>	Data representation language	parameter	A set of values needed as the parameter of an <operation>.
<request>	Data representation language	parameter	A set of values needed as the parameter to a service, which will include an indication if needed of the operation to perform.
<request-block>	Data representation language	parameter	A hierarchical structure consisting of one or more requests to the same service, possibly with different <handles>.
<bBox>	Data representation language	parameter	Bounding Box used in lieu of a spatial query in a feature request,
<featureID>	Text	parameter	ID of a feature used to retrieve property information. This may or may not be the features name.
<propertyID>	Text	parameter	Scoped name of a property. The name space of a property is always its containing feature.
<function>	Text	parameter	The name of a function supported by the service as specified in the capabilities document for that service.

Symbol	Format	Type	Definition
<lagniappe>	Any	parameter	Any extra parameter or set of parameters required for reasons not directly associated to the semantics of the service itself. This can include for example license or user information or timing constraints required for referenced data that may not be under the direct control of the services host system. All parameters will be tagged, and so the order of the parameters in a message does not affect behavior.
<lockAction>	Text	parameter	An Enumeration of possible locking strategies (ALL or SOME) requested in a transaction sequence.
<maxFeatures>	Text	parameter	Maximum number of features to return in a request.
<maxSize>	Text	parameter	Maximum volume of data to return in a request; in either kilobytes (kb) or megabytes (mb). The units of the specification will always be specified.
<namespace>	Text	parameter	Set of namespaces used in a message or file.
<outputFormat>	Text	parameter	The requested format for the response. For tracking the types of files, each format should be associated to a file extension for use in REST when specifying which representation of a resource is being requested.
<query>	Data representation language	parameter	The query or filter that restricts the return of the particular features.
<requestValue>	Data representation language	parameter	The contents of a request for the operation being invoked.
<resultType>	Text	parameter	The switch between “hits” and “results” which determines the contents of a return value.
<section>	Text	parameter	A name for a section of a document, usually the Capabilities document for the service.

Symbol	Format	Type	Definition
<subtypesOf>	Text	parameter	A description the feature types associated to some particular feature type by substitutability. In XML this is defined by the substitution group of the particular feature. In an object environment, this is defined by inheritance of the feature type. In an SQL environment, this is probably implemented by a substitution group represented by a view defined by a union query.
<srsName>	Text	parameter	The name of the spatial reference system being used. The definition of the SRS should be somewhere available, and linked to in the capabilities document. The ordering of the SRS coordinates should be maintained, and if a service wishes to use a different coordinate ordering for a more classic SRS, then that should be also specified in the capabilities or in a "proxy" SRS acting as a reordering "façade" of the more-classic root SRS.
<SRS>	Data representation language	parameter	A spatial reference system.
<acceptVersion>	Data representation language	parameter	A list of acceptable version of the service.
<acceptFormat>	Data representation language	parameter	A list of acceptable output formats from the service. For tracking the types of files, each format should be associated to a file extension as in the output format.

Table B 2: Commonly used file types and extensions for use in REST requests

Format	File or Resource Extension
Undifferentiated XML	.xml
GML	.gml
GeoJson, or Json	.json
HTML	.html or .htm
XML schema files	.xsd

Format	File or Resource Extension
SQL text file	.sql