Open Geospatial Consortium

Submission Date: 2014-03-11

Approval Date: 2016-02-29

Publication Date: 2016-06-10

External identifier of this OGC® document: http://www.opengis.net/doc/IS/cat/3.0

URL for this OGC® document: http://docs.opengeospatial.org/is/12-168r6/12-168r6.html

Internal reference number of this OGC® document: OGC 12-168r6

Version: 3.0

Category: OGC® Implementation Standard

Editors: Douglas Nebert, Uwe Voges, Lorenzo Bigagli

OGC® Catalogue Services 3.0 - General Model

Interface Standard

The OGC Catalog Service 3.0 SWG would like to dedicate this work to the memory of Douglas D. Nebert. Doug was the convener and chair of the group who coordinated the editing of this specification. He was a long time advocate for developing a catalogue standard within OGC and indeed was a strong, vocal and passionate supporter of the OGC and its ideals. He will be remembered and missed. You can read more about Doug on our website.

Document type:	OGC® Standard
Document subtype:	Part of Implementation Standard
Document stage:	Approved for public release
Document language:	English

Contents

Figures

Tables

i. Abstract

OGC® Catalogue Services support the ability to publish and search collections of descriptive information (metadata records) for geospatial data, services, and related information. Metadata in catalogues represent resource characteristics that can be queried and presented for evaluation and further processing by both humans and software. Catalogue services are required to support the discovery and binding to registered information resources within an information community.

This part of the Catalogue Services standard describes the common architecture for OGC Catalogue Services. This document abstractly specifies the interfaces between clients and catalogue services, through the presentation of abstract models. This common architecture is Distributed Computing Platform neutral and uses UML notation. Separate (Part) documents specify the protocol bindings for these Catalogue services, which build upon this document, for the HTTP (or CSW) and OpenSearch protocol bindings.

An Abstract Conformance Test Suite is not included in this document. Such Suites shall be developed by protocol bindings and Application Profiles (see 8.5, ISO/IEC TR 10000-2:1998) that realize the conformance classes listed herein. An **application profile** consists of a set of metadata elements, policies, and guidelines defined for a particular application[1].

OGC document number 14-014r3 – HTTP Protocol Binding – Abstract Test Suite is available to address conformance with the provisions of OGC document number 12-176r7 – HTTP Protocol Binding. All annexes to this document are informative.

ii. Keywords

The following are keywords to be used by search engines and document catalogues.

OGC Catalogue Services, metadata, geospatial data, geospatial services, search, discovery, abstract model, general model, HTTP, CSW, OpenSearch, Abstract Conformance Test Suite, ogcdoc, OGC document, asynchronous, catalogue, CQL, client, csw:Record, distributed, Dublin Core, federated, filter, GetCapabilities, GetDomain, GetRecords, GetRecordById, Harvest, http, https, KVP, metadata, record, request, resource, response, server, schema, spatial, temporal, Transaction, UnHarvest, XML, XML-Schema.

iii. Preface

This document is one part of the OGC® Catalogue Services version 3.0 Implementation Standard. Unlike previous versions, Catalogue 3.0 is now divided in multiple parts, with this part specifying the abstract model and another to describe the HTTP protocol binding known as Catalogue Service for the Web (CSW).

[1] http://dublincore.org/documents/2001/04/12/usageguide/glossary.shtml#A

This version of the Catalogue Standard has been significantly improved, largely based on change requests submitted by both Open Geospatial Consortium (OGC) members and the public. The changes made in this version relative to version 2.0.2 (OGC document 07-006r1) are summarized in Annex B.

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 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.

iv. Submitting organizations

The following organizations submitted this document to the OGC.

- con terra GmbH
- National Research Council of Italy (CNR)
- Cubewerx Inc.
- Intergraph Corporation
- Joint Research Centre (JRC), European Commission
- U.S. Geological Survey

Earlier versions of this Standard were submitted to the OGC by the following organizations:

- BAE SYSTEMS Mission Solutions (formerly Marconi Integrated Systems, Inc.)
- Blue Angel Technologies, Inc.
- Environmental Systems Research Institute (ESRI)
- Geomatics Canada (Canada Centre for Remote Sensing (CCRS)
- Intergraph Corporation
- MITRE
- Oracle Corporation
- U.S. Federal Geographic Data Committee (FGDC)
- U.S. National Aeronautics and Space Administration (NASA)
- U.S. National Imagery and Mapping Agency (NIMA)

v. Submitters

All questions regarding this submission should be directed to the editors or the contributors:

Name	Organization
Doug Nebert	U.S. Geological Survey
Uwe Voges	con terra GmbH
Lorenzo Bigagli	National Research Council of Italy (CNR)
Panagiotis (Peter) Vretanos	CubeWerx, Inc.
Bruce Westcott	Intergraph Corporation

OGC® Catalogue Services Specification - General Model

1. Scope

This document abstractly specifies the interfaces and a framework for defining bindings and application profiles required to publish and access digital catalogues of metadata for geospatial data, services, and related resource information. These Catalogue Services support the use of one of several identified query languages to find and return results using well-known content models (metadata schemas) and encodings.

This Standard is applicable to the implementation of interfaces on catalogues of a variety of information resources. The target audience for this standard is the community of software developers who are implementers of OGC compliant Catalogue servers and clients.

2. Conformance

Conformance to the mandatory catalogue service abstract interfaces is described in section 8. It is the requirement of protocol-specific bindings and application profiles to provide concrete tests and validation in conformance with these abstract conformance classes. Test data and queries may be included in Application Profiles associated with this abstract model and with specific protocol bindings.

3. References

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

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

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

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

IANA, Internet Assigned Numbers Authority, *MIME Media Types*, available at http://www.iana.org/assignments/media-types/

ISO/IEC 8825:1990, Information technology – Open Systems Interconnection – Specification of Basic Encoding Rules for Abstract Syntax Notation One (ASN.1)

ISO/IEC TR 10000-1:1998. *Information Technology – Framework and taxonomy of International Standardised Profiles – Part 1: General principles and documentation framework*. Technical Report, JTC 1. Fourth edition, Available [online]: <http://standards.iso.org/ittf/PubliclyAvailableStandards/c030726_ISO_IEC_TR_10000-1_1998(E).zip>.

ISO/IEC 10746-2:1996. *Information Technology – Open Distributed Processing – Reference Model: Foundations*. Common text with ITU-T Recommendation X.902, Available [online]: <http://standards.iso.org/ittf/PubliclyAvailableStandards/s018836_ISO_IEC_10746-2_1996(E).zip >.

ISO 8601:2000(E), *Data elements and interchange formats - Information interchange - Representation of dates and times*

ISO 19101:2002, Geographic information – Reference model

ISO 19103 (DTS), Geographic information – Conceptual schema language, (Draft Technical Specification)

ISO 19106:2003, *Geographic Information – Profiles*

ISO 19108:2002, Geographic information – Temporal schema

ISO 19109:2002 (DIS), Geographic information – Rules for application schema

ISO 19110:2001 (DIS), Geographic information – Methodology for feature cataloguing

ISO 19113:2002, Geographic information – Quality principles

ISO 19114:2001, (DIS) Geographic information – Quality evaluation procedures

ISO 19118:2002, (DIS) Geographic information – Encoding

ISO/IEC 14977:1996, *Information technology – Syntactic metalanguage – BNF*

ISO 19115:2003, *Geographic Information – Metadata*

ISO 19119:2005, *Geographic Information – Services*

ISO/TS 19139:2007, *Geographic Information – Metadata -Implementation Specification*

OASIS/*ebXML Registry Services Specification* v2.5

OGC 99-113, *OGC Abstract Specification Topic 13: Catalogue Services*

OGC 02-112, *OGC Abstract Specification Topic 12: OpenGIS Service Architecture*

OGC 09-026r1, OGC *Filter Encoding 2.0 Encoding Standard*,

OGC 06-121r9, *OGC Web Service Common Implementation Specification, Version 2.0.0*

OMG UML, *Unified Modeling Language, Version 1.3*, The Object Management Group (OMG): http://www.omg.org/cgi-bin/doc?formal/00-03-01

OGC 12-176r2, OGC® Catalogue Services specification – HTTP protocol binding (v3.0.0)

4. Terms and Definitions

This document uses the terms defined in Sub-clause 5.3 of [OGC 06-121r8], which is based on the ISO/IEC Directives, Part 2, Rules for the structure and drafting of International Standards. In particular, the word “shall” (not “must”) is the verb form used to indicate a requirement to be strictly followed to conform to this standard.

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

4.1
client
software component that can invoke an **operation** from a **server**

4.2
data clearinghouse
collection of institutions providing digital data, which can be searched through a single interface using a common metadata standard [ISO 19115]

4.3
data level
stratum within a set of layered levels in which data is recorded that conforms to definitions of types found at the application model level [ISO 19101]

4.4
dataset series
collection of datasets sharing the same product specification [ISO 19113, ISO 19114, ISO 19115]

4.5
feature catalogue
catalogue containing definitions and descriptions of the feature types, feature attributes, and feature relationships occurring in one or more sets of geographic data, together with any feature operations that may be applied [ISO 19101, ISO 19110]

4.6
geographic dataset
dataset with a spatial aspect [ISO 19115]

4.7
geographic information
information concerning phenomena implicitly or explicitly associated with a location relative to the Earth [ISO 19128 draft]

4.8
identifier
a character string that may be composed of numbers and characters that is exchanged between the client and the server with respect to a specific identity of a resource

4.9
interface
named set of operations that characterize the behaviour of an entity [ISO 19119]

4.10
metadata dataset
metadata describing a specific dataset [ISO 19101]

4.11
metadata entity
group of metadata elements and other metadata entities describing the same aspect of data

NOTE 1 A metadata entity may contain one or more metadata entities.

NOTE 2 A metadata entity is equivalent to a class in UML terminology [ISO 19115].

4.12
metadata schema
conceptual schema describing metadata

NOTE ISO 19115 describes a standard for a metadata schema. [ISO 19101]

4.13
metadata section
subset of metadata that defines a collection of related metadata entities and elements [ISO 19115]

4.14
operation
specification of a transformation or query that an object may be called to execute [ISO 19119]

4.15
parameter
variable whose name and value are included in an operation **request** or **response**

4.16
profile
set of one or more base standards and - where applicable - the identification of chosen clauses, classes, subsets, options and parameters of those base standards that are necessary for accomplishing a particular function [ISO 19101, ISO 19106]

4.17
qualified name
name that is prefixed with its naming context

EXAMPLE The qualified name for the road no attribute in class Road defined in the Roadmap schema is RoadMap.Road.road_no. [ISO 19118]

4.18
request
invocation of an **operation** by a **client**

4.19
response
result of an **operation,** returned from a **server** to a **client**

4.20
resource
an object or artefact that is described by a record in the information model of a catalogue

4.21
schema
formal description of a model [ISO 19101, ISO 19103, ISO 19109, ISO 19118]

4.22
server
service instance
a particular instance of a **service** [ISO 19119 edited]

4.23
service
distinct part of the functionality that is provided by an entity through interfaces [ISO 19119]

capability which a service provider entity makes available to a service user entity at the interface between those entities [ISO 19104 terms repository]

4.24
service interface
shared boundary between an automated system or human being and another automated system or human being [ISO 19101]

4.25
service metadata
metadata describing the **operations** and **geographic information** available at a **server** [ISO 19128 draft]

4.26
state
condition that persists for a period

NOTE The value of a particular feature attribute describes a condition of the feature [ISO 19108].

4.27
transfer protocol
common set of rules for defining interactions between distributed systems [ISO 19118]

4.28
version
version of an Implementation Specification (document) and XML Schemas to which the requested operation conforms

NOTE An OWS Implementation Specification version may specify XML Schemas against which an XML encoded operation request or response shall conform and should be validated.

5. Conventions

5.1 Symbols (and abbreviated terms)

All symbols used in this document are either:

1. Common mathematical symbols; or
2. UML 2 (Unified Modeling Language) as defined by OMG and accepted as a publicly available standard (PAS) by ISO in its earlier 1.3 version.

In this document the following abbreviations and acronyms are used or introduced:

BNF	Baukus Naur Form
CSW	Catalogue Services for the Web
HTTP	Hypertext Transfer Protocol
ISO	International Organization for Standardization
MIME	Multipurpose Internet Mail Extensions

OGC	Open Geospatial Consortium, also referred to as OGC®
UML	Unified Modeling Language
XML	Extensible Markup Language

5.2 UML notation

All UML diagrams in this document follow the guidance as documented in OGC OWS Common 2.0 section 5.2.

5.3 XML Schema

The following notations are used in XML Schema fragment presented in this document:

- Brackets ([]) are used to denote constructs that can be optionally specified. In the following example:

  ```
  <xsd:element name="MyElement" minOccurs="0" [maxOccurs="1"]>
  ```

the brackets around *maxOccurs="1"* mean that this construct is optional and can be omitted

5.4 URN notation

All requirements listed in this document are relative to the root URL: http://www.opengis.net/doc/IS/cat/3.0 . Wherever there is a stated requirement and the work "req" is shown, "req" can be replaced with http://www.opengis.net/doc/IS/cat/3.0 to define the complete requirement URL.

6. Catalogue abstract information model

6.1 Introduction

The abstract information model specifies a BNF grammar for a minimal query language, a set of core queryable[2] attributes (names, definitions, conceptual datatypes), and a common record format that defines the minimal set of elements that should be returned in the brief and summary element sets.

The geospatial community is very broad and works in many different operational environments, as shown in the information discovery continuum in **Figure 1 - Information discovery continuum.** On one extreme there are tightly coupled systems dedicated to well-defined functions in a tightly controlled environment. At the other extreme are Web based services that know nothing about the client. This document provides a specification that is applicable to the full range of catalogue operating environments.

[2] That can be queried.

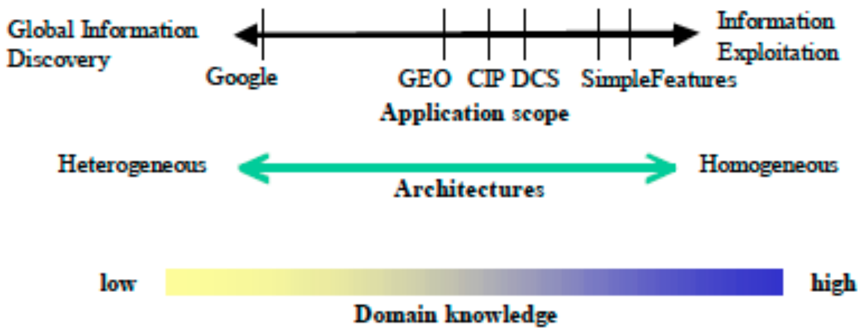


Figure 1 - Information discovery continuum

6.2 Query language support

6.2.1 Introduction

The query capabilities of the OGC General Catalogue Model provide a minimum set of data types and query operations that can be assumed of OGC Compliant Catalogue implementations. In addition, these Query Capabilities provide a high degree of flexibility enabling alternate styles of query, result presentation, and the potential support of any geo-enabled query language. This flexibility is provided by the query operation that contains the parameters needed to select the query result presentation style and to provide a query expression that includes the actual query with an identification of the query language used. The query operation, query expression, and other related operations are further discussed in Clause 7.2.4.

Requirement 1 /req/model/query-language:
A Catalogue service query interfaces shall support and reference a published syntax for processing full text and fielded query.

The interoperability goal is supported by the specification of a minimal abstract query (predicate) language, which shall be supported by all compliant OGC Catalogue Services. This query language supports Boolean queries, text matching operations, temporal data types, and geospatial operators. The minimal query language syntax is based on the SQL WHERE clause in the SQL SELECT statement. The OGC Filter Specification is an implementation of a query language that is transformable to the OGC Catalogue Common Query Language (OGC CommonQL).

This minimal query language assists the consumer in the discovery of datasets of interest at all sites supporting the OGC Catalogue Services. The ability to specify alternative query languages allows for evolution and higher levels of interoperability among more tightly coupled communities of Catalogue Service Providers and Consumers.

Requirement 2 /req/common-query-language:
A Catalogue service query interfaces shall support a catalogue query syntax that is transformable to the BNF[3] *expressed in subclause 6.2.2*

6.2.2 OGC Catalogue Common Query Language (OGC CommonQL)

This sub-clause defines the OGC_Catalogue Common Query Language (OGC CommonQL) (BNF to be found in 9). OGC_CommonQL is the primary query language to be supported by multiple OGC Catalogue Service bindings in order to support search interoperability.

Assumptions made during the development of OGC CommonQL:

a) The query will have syntax similar to the SQL “Where Clause.”

b) The expressiveness of the query will not require extensions to various current query systems used in geospatial catalogue queries other than the implementation of some geo operators.

c) The query language is extensible.

d) OGC CommonQL supports both tight and loose queries. A tight query is defined for the case when a catalogue doesn’t support an attribute/column specified in the query, no entity/row can match the query and the null set is returned. In a loose query, if an attribute is undefined, it is assumed to match.

6.2.3 Extending the OGC CommonQL

The OGC CommonQL BNF can be extended by adding new predicates, operations, and datatypes. The following discussion is an example of extending the BNF to include a CLASSIFIED-AS operator using the patterns identified in OASIS/ebXML Registry Services Specification v2.5. This extension could appear in a protocol binding or an Application Profile.

This standard makes no assumptions about how taxonomies are maintained in a catalogue, or how records are classified according to those taxonomies. Instead, this specification defines a routine, CLASSIFIED-AS, in order to support classification queries based on taxonomies.

The CLASSIFIED-AS routine takes three arguments. The first argument is the abstract entry point whose classification is being checked. The second argument is the key name string that represents a path expression in the taxonomy. The last argument is the key value string that represents the corresponding path expression containing key values that are the targets of the query. In both cases, the first element of the path expression for the

[3] http://en.wikipedia.org/wiki/Backus-Naur_Form

key name argument and key value arguments shall be the name of the taxonomy being used. The normal wildcard matching characters, '_' for a single character and '%' for zero or more characters, may be used in the key value expression which is the last argument of the CLASSIFIED_AS routine.

The following set of productions defines the CLASSIFIED-AS routine.

```
/* The following example:                                      */
/*                                                             */
/* RECORD CLASSIFIED AS CLASSIFICATIONSCHEME='GeoClass'   */
/*      ='/GeoClass/North America/%/Ontario'               */
/*                                                             */
/* Will find all records in all the Ontario's in North America.      */

The following are the required BNF specializations:

<classop argument list> ::= <left paren> <entry_point> <comma>
        <Classification Scheme> <comma><Classification Node> <right
paren>

<entry_point> ::= <identifier>
<Classification Scheme> ::= <identifier>
<classop name> ::= CLASSIFIED_AS

<Classification Node> ::= <identifier> | <solidus><path
element>[<solidus><path element>]…
<path element> ::= <character pattern>

<routine invocation> ::= | <geoop name><georoutine argument list>
                         | <relgeoop name><relgeoop argument list>
                         | <routine name><argument list>
                         | <classop><classop argument list>
```

Consider the following example:

```
CLASSIFIED_AS('RECORD', 'GeoClass', 'GeoClass/NorthAmerica/%/Ontario')
```

In this example, we are searching records classified according to the **GeoClass** taxonomy. Specifically, we are looking for all catalogue records classified as *Continent=NorthAmerica*, *Country=**any country*** and *State=Ontario*. Notice how the wildcard character '%' is used to search for any Country node.

Here is the same example encoded using XML:

```
<ogc:Filter xmlns:ogc="http://http://www.opengis.net/ogc">
  <ogc:ClassifiedAs>
     <ogc:TypeName>csw:Record</ogc:TypeName>
     <ogc:ClassificationScheme>GetClass</ogc:ClassificationScheme>
     <ogc:ClassificationNode>/GeoClass/NorthAmerica/%/Ontario
       </ogc:ClassificationNode>
  </ogc:ClassifiedAs>
</ogc:Filter>
```

In order for catalogue clients to be able to determine which taxonomies are available, a catalogue implementation *should* advertise the list of available taxonomies in its capabilities document. If a query is executed against a non-existent taxonomy, then an exception *should* be raised.

6.2.4 Query language realization

Many OGC service operations have the requirement to pass and process a query as a structure to perform a request. There are several query languages and messaging mechanisms identified within OGC standards. Application Profiles should be explicit about the selected query languages and any features peculiar to a scope of application. The following items should be addressed in the preparation of an Application Profile with respect to query language support.

Support for "abstract" query against well-known queryable entry points (e.g. OGC Core). Some standards promote or require the exposure of well-known field-like objects as common search targets (queryables), allowing interrogation of a service without prior negotiation on information content. The mandatory queryable attributes which shall be recognized by all OGC Catalogue Services are discussed in Subclause 6.3.2.

Selection of a query language. Some standards describe one or more query languages that can be supported. Identify the name and version of required query language(s) anticipated by this Application Profile for use.

Supported data types (e.g. character, integer, coordinate, date, geometry) and operator types (e.g. inequality, proximity, partial string, spatial, temporal). Query languages may be restricted in their implementation or extended with functions not described in the base standard. This narrative should provide lists or reference documents with the enumerated data types and operator types required by this Application Profile. In addition, any description of special techniques (e.g. supporting joins or associations) that are expected by an Application Profile should be described.

6.3 Core catalogue schema

6.3.1 Introduction

Metadata structures, relationships, and definitions -- known as conceptual schemas -- exist for multiple information communities. For the purposes of interchange of information within an information community, a metadata schema may be defined that provides a common vocabulary which supports search, retrieval, display, and association between the description and the object being described. Although this standard does not require the use of a specific schema, the adoption of a given schema within an information-sharing community ensures the ability to communicate and discover information.

The geomatics standardization activity in ISO Technical Committee 211 include formal schemas for geospatial metadata that are intended to apply to all types of information.

These metadata standards, ISO 19115:2003[4] and ISO 19115-1:2014[5] include proposals for core (discovery) metadata elements in common use in the geospatial community. ISO/TS 19139:2007 defines a formal encoding and structure of ISO 19115:2003 metadata for exchange. Where a catalogue service advertises such application schemas, catalogues that handle geographic dataset descriptions should conform to published metadata standards and encodings, e.g. ISO 19115:2003, and support XML encoding per ISO 19139 or profiles thereof. Service metadata elements should be consistent with ISO 19119[6] or 19115:2014[7].

6.3.2 Core queryable properties

The goal of defining core queryable properties is query interoperability among catalogues that implement the same protocol binding and query compatibility among catalogues that implement different protocol bindings, perhaps through the use of "bridges" or protocol adapters. Defining a set of core queryable properties also enables simple cross-profile discovery, where the same queries can be executed against any catalogue service without modification and without detailed knowledge of the catalogue's information model. This requires a set of general metadata properties that can be used to characterize any resource.

Tables 1, 2 and 3 define a set of abstract queryables that binding protocols shall realize in their core queryable schemas. Binding protocols shall further specify a record identifier (ID) based on the native platform ID types. Binding protocols shall also specify how the values of core queryable properties shall be encoded in service requests. Binding protocols may choose to use a single comma-separated list for compound datatypes or may label each sub-element for clarity and order flexibility. Application profiles may further modify or redefine the realization of the core queryables and how their values are encoded.

Requirement 3 /req/common-queryables:

A Catalogue service query interfaces shall support the set of common queryable elements described in Tables 1-3. Services shall perform appropriate mapping of public query terms to internal equivalents to enable general search of catalogues.

NOTE: Queryable items may differ from the response data elements

[4] http://www.iso.org/iso/home/store/catalogue_ics/catalogue_detail_ics.htm?csnumber=53798

[5] hhttp://www.iso.org/iso/catalogue_detail.htm?csnumber=53798

[6] OGC Abstract Specification Topic Volume 12 (http://portal.opengeospatial.org/files/?artifact_id=1221) and ISO 19119 http://www.iso.org/iso/home/store/catalogue_tc/catalogue_detail.htm?csnumber=39890

[7] hhttp://www.iso.org/iso/catalogue_detail.htm?csnumber=53798

Table 1 — Common queryable elements

Name	Definition	Data type
Subject [a]	The topic of the content of the resource [b]	CharacterString
Title [a]	A name given to the resource	CharacterString
Abstract [a]	A summary of the content of the resource	CharacterString
AnyText	A target for full-text search of character data types in a catalogue	CharacterString
Format [a]	The physical or digital manifestation of the resource	CharacterString
Identifier [a]	An unique reference to the record within the catalogue	Identifier
TemporalExtent	Date or period for the content being described in metadata	Date-8601
Modified [c]	Date on which the record was created or updated within the catalogue	Date-8601
Type [a]	The nature or genre of the content of the resource. Type can include general categories, genres or aggregation levels of content.	CodeList [f]
BoundingBox [d]	A bounding box for identifying a geographic area of interest	BoundingBox, See Table 2
CRS [e]	Geographic Coordinate Reference System (Authority and ID) for the BoundingBox	Identifier
Association	Complete statement of a one-to-one relationship	Association, See Table 3

a Names, but not necessarily the identical definition, are derived from the Dublin Core Metadata Element Set, version 1.1:ISO Standard 15836-2003 (February 2003)

b Typically, a Subject will be expressed as keywords, key phrases or classification codes that describe a topic of the resource. Recommended best practice is to select a value from a controlled vocabulary or formal classification scheme.

c DCMI metadata term <http://dublincore.org/documents/dcmi-terms/>.

d Same semantics as EX_GeographicBoundingBoxclass in ISO 19115.

e If not supplied, the BoundingBox CRS is a Geographic CRS with the Greenwich prime meridian.

f A "CodeList" is a CharacterString taken from an authoritative list of CharacterStrings or Identifiers. The authority may optionally be identified in the value.

Table 2 — Composition of compound element "BoundingBox"

Name	Definition	Data type
WestBoundLongitude	Western-most coordinate of the limit of the resource's extent, expressed in longitude in decimal degrees (positive east)	numeric
SouthBoundLatitude	Southern-most coordinate of the limit of the resource's extent, expressed in latitude in decimal degrees (positive north)	numeric
EastBoundLongitude	Eastern-most coordinate of the limit of the resource's extent, expressed in longitude in decimal degrees (positive east)	numeric
NorthBoundLatitude	Northern-most, coordinate of the limit of the resource's extent, expressed in latitude in decimal degrees (positive north)	numeric

Table 3 — Composition of compound element "Association"

Name	Definition	Data type
TargetResourceID	Referenced resource	Identifier
SourceResourceID	Referencing resource	Identifier
Relation	The name of the description of the relationship	CodeList or Identifier

All realizations of the core queryable properties in a binding protocol shall include all the properties listed in Tables 1, 2, or 3 even if the underlying information model does not include information that can be mapped into all properties. Core properties that cannot have a value assigned to them because the information is not available in the information model of the catalogue shall be considered as having a value of NULL.

The properties "Title", "Identifier" and the pseudo-property "AnyText" shall be supported as mandatory queryables in all implementations. Protocol bindings shall describe mechanisms to identify and elaborate on the queryables and operations supported by a given catalogue service.

6.3.3 Core returnable properties

A set of core properties returned from a metadata search is encouraged to permit the minimal implementation of a catalogue service independent of a companion application profile, and to permit the use of metadata returned from different systems and protocol bindings. The core metadata is returned as a request for the Common Element Set. The Common Element Set is a new group of public metadata elements, expressed using the nomenclature and syntax of Dublin Core Metadata, ISO 15836. Table 4 provides some interpretation of Dublin Core elements in the context of metadata for geospatial data and services.

Requirement 4 /req/common-returnables :

Catalogue service query interfaces shall support the set of common returnable elements in result set metadata as described in Table 4. Service shall perform appropriate mapping of internal fields, as necessary, to published returnables to enable interoperable search of catalogues.

Table 4 — List of common returnable properties

Dublin Core element name	Term used in OGC queryables	Definition	Data type
title	Title	A name given to the resource. Also known as "Name".	CharacterString
creator		An entity primarily responsible for making the content of the resource.	CharacterString
subject	Subject	A topic of the content of the resource. This is a place where a Topic Category or other taxonomy could be applied.	CharacterString
description	Abstract	An account of the content of the resource. This is also known as the "Abstract" in other aspects of OGC, FGDC, and ISO metadata.	CharacterString
publisher		An entity responsible for making the resource available. This would equate to the Distributor in ISO and FGDC metadata.	CharacterString
contributor		An entity responsible for making contributions to the content of the resource.	CharacterString
date	Modified	The date of a creation or update event of the catalogue record.	ISO-8601 date
type	Type	The nature or genre of the content of the resource.	CodeList
format	Format	The physical or digital manifestation of the resource.	CharacterString
identifier	Identifier	A unique reference to the record within the catalogue.	Identifier
source	Source	A reference to the full metadata from which the present resource is derived.	URI
language		A language of the intellectual content of the catalogue record.	CharacterString
relation	Association	The name of the relationship that exists between the resource described by this record and a related resource	
coverage	BoundingBox	The spatial and temporal extent or scope of the content of the resource.	Extent
rights		Information about rights held in and over the resource.	CharacterString

The core elements are recommended for a response but do not need to be populated. The support for a common syntax for the returnable properties as a “common” Summary Element Set is defined in the protocol binding clauses.

```
<?xml version="1.0" encoding="UTF-8"?>
<GetRecords
  service="CSW"
  version="2.0.2"
  maxRecords="5"
  startPosition="1"
  resultType="results"
  outputFormat="application/xml"
  outputSchema="http://www.opengis.net/cat/csw/2.0.2"
  xmlns="http://www.opengis.net/cat/csw/2.0.2"
  xmlns:csw="http://www.opengis.net/cat/csw/2.0.2"
  xmlns:ogc="http://www.opengis.net/ogc"
  xmlns:ows="http://www.opengis.net/ows"
  xmlns:dc="http://purl.org/dc/elements/1.1/"
  xmlns:dct="http://purl.org/dc/terms/"
  xmlns:gml="http://www.opengis.net/gml"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.opengis.net/cat/csw/2.0.2
                      ../../../csw/2.0.2/CSW-discovery.xsd">
  <Query typeNames="csw:Record">
     <ElementSetName typeNames="csw:Record">full</ElementSetName>
     <Constraint version="1.1.0">
        <ogc:Filter>
           <ogc:And>
              <ogc:PropertyIsLike escapeChar="\" singleChar="?" 
wildCard="*">
                 <ogc:PropertyName>dc:title</ogc:PropertyName>
                 <ogc:Literal>*Elevation*</ogc:Literal>
              </ogc:PropertyIsLike>
              <ogc:PropertyIsEqualTo>
                 <ogc:PropertyName>dc:type</ogc:PropertyName>
                 <ogc:Literal>Service</ogc:Literal>
              </ogc:PropertyIsEqualTo>
              <ogc:PropertyIsGreaterThanOrEqualTo>
                 <ogc:PropertyName>dct:modified</ogc:PropertyName>
                 <ogc:Literal>2004-03-01</ogc:Literal>
              </ogc:PropertyIsGreaterThanOrEqualTo>
              <ogc:Intersects>
                 <ogc:PropertyName>ows:BoundingBox</ogc:PropertyName>
                 <gml:Envelope>
                    <gml:lowerCorner>14.05 46.46</gml:lowerCorner>
                    <gml:upperCorner>17.24 48.42</gml:upperCorner>
                 </gml:Envelope>
              </ogc:Intersects>
           </ogc:And>
        </ogc:Filter>
     </Constraint>
  </Query>
</GetRecords>
```

The response to such a query, might be:

```
<?xml version="1.0" encoding="UTF-8"?>
<csw:Record
  xmlns:csw="http://www.opengis.net/cat/csw/2.0.2"
  xmlns:dc="http://purl.org/dc/elements/1.1/"
  xmlns:dct="http://purl.org/dc/terms/"
  xmlns:ows="http://www.opengis.net/ows"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.opengis.net/cat/csw/3.0.0
                      ../../../csw/3.0.0/record.xsd">
   <dc:creator>U.S. Geological Survey</dc:creator>
   <dc:contributor>State of Texas</dc:contributor>
   <dc:publisher>U.S. Geological Survey</dc:publisher>
   <dc:subject>Elevation, Hypsography, and Contours</dc:subject>
   <dc:subject>elevation</dc:subject>
   <dct:abstract>Elevation data collected for the National Elevation
Dataset (NED) based on 30m horizontal and 15m vertical
accuracy.</dct:abstract>
   <dc:identifier>ac522ef2-89a6-11db-91b1-7eea55d89593</dc:identifier>
   <dc:relation>OfferedBy</dc:relation>
   <dc:source>http://myserver.com/csw?
SERVICE=CSW&REQUEST=GetRecordById&RECORD=dd1b2ce7-0722-4642-8cd4-
6f885f132777</dc:source>
   <dc:rights>Copyright © 2011, State of Texas</dc:rights>
   <dc:type>Service</dc:type>
   <dc:title>Elevation Mapping Service for Texas</dc:title>
   <dct:modified>2011-03-01</dct:modified>
       <dc:language>en</dc:language>
   <ows:BoundingBox>
      <ows:LowerCorner>-108.44 28.229</ows:LowerCorner>
      <ows:UpperCorner>-96.223 34.353</ows:UpperCorner>
   </ows:BoundingBox>
</csw:Record>
```

6.3.4 Information structure and semantics

Some services that implement OGC Standards expect a rigid syntax for the information resources to be returned, whereas others do not. This subclause allows an Application Profile to be specific about what information content, syntax, and semantics are to be communicated over the service. The following items should be addressed in an Application Profile.

a) Identify information resource types that can be requested. In the case of a catalogue service, the information resources being described by the metadata may include geographic data, imagery, services, controlled vocabularies, or schemas among a wide variety of possible types. This subclause allows the community to specify or generalise the resource types being described in metadata for their scope of application.

b) Identify a public reference for the information being returned by the service (e.g. ISO 19115:2003 “Geographic Information – Metadata “). Include any semantic resources including data content model, dictionary, feature type catalogue, code lists, authorities, taxonomies, etc.

c) Identify named groups of properties (element sets) that may be requested of the service (e.g. “brief,” “summary,” or “full”) and the valid format (syntax) for each element set. Identify valid schema(s) with respect to a given format to assist in the validation of response messages.

d) Specialise the core queryable properties list by making some optional queryable attributes mandatory, deleting other optional attributes and adding queryable attributes that should be standard across all profile users

e) Optional mapping of queryable and retrievable properties against other public metadata models or tags.

f) Expected response/results syntax and content Message syntax and schemas (e.g. brief/full, individual elements).

7. General catalogue interface model

7.1 Introduction

The General Catalogue Interface Model (GCIM) provides a set of abstract service interfaces that support the discovery, access, maintenance and organization of catalogues of geospatial information and related resources. The interfaces specified are intended to allow users or application software to find information that exists in multiple distributed computing environments, including the World Wide Web (WWW) environment.

Implementation design guidance is included in specified protocol binding Parts of this standard. Each protocol binding includes a mapping from the general interfaces, operations, and parameters specified in this clause to the constructs available in a chosen protocol. In most, but not all, protocol bindings, there may be restrictions or refinements on implementation of the General Model agreed within an implementation community. This sub-clause provides an overview of the portions of the GCIM that are realised by implementations described in other Catalogue Service Part documents.

Application profiles are intended to further document implementation choices. An Application Profile is predicated on the existence of one protocol binding as a Part of this standard.

Figure 2 - Reference model architecture shows the Reference Architecture assumed for development of the OGC Catalogue Interface. The architecture is a multi-tier arrangement of clients and servers. To provide a context, the architecture shows more than just catalogue interfaces. The bold lines illustrate the scope of OGC Catalogue.

The Application Client shown in **Figure 2 - Reference model architecture Error! Reference source not found.**interfaces with the Catalogue Service using the OGC Catalogue Interface. The Catalogue Service may draw on one of three sources to respond to the Catalogue Service request: a Metadata Repository local to the Catalogue Service, a Resource service, or another Catalogue Service. The interface to the local Metadata Repository is internal to the Catalogue Service. The interface to the Resource service can be a private or OGC Interface. The interface between Catalogue Services is the OGC Catalogue Interface. In this case, a Catalogue Service is acting as both a client and server. Data returned from an OGC Catalogue Service query is processed by the requesting Catalogue Service to return the data appropriate to the original Catalogue request. See Annex A for more about Distributed Searching.

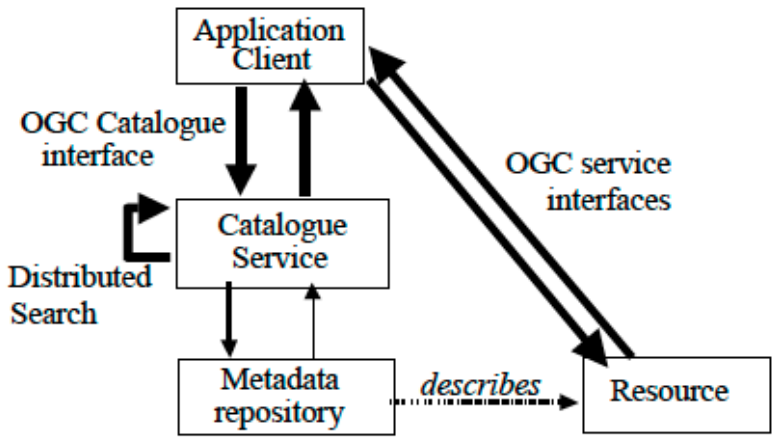


Figure 2 - Reference model architecture

7.2 Interface definitions

7.2.1 Overview

Figure 3 - General OGC catalogue UML static model is a general UML model of OGC catalogue service interfaces, in the form of a class diagram. Operation signatures have been suppressed in this figure for simplicity but are described in detail below. This model shows the Catalogue Service class plus five other classes with which that class are associated. A Catalogue Service is a realization of an OGC Service. Each instance of the Catalogue Service class is associated with one or more of these other classes, depending on the abilities included in that service instance. Each of these other classes defines one or several related operations that can be included in a Catalogue Service class instance. The Catalogue Service class directly includes only the serviceTypeID attribute, with a fixed value for the service type.

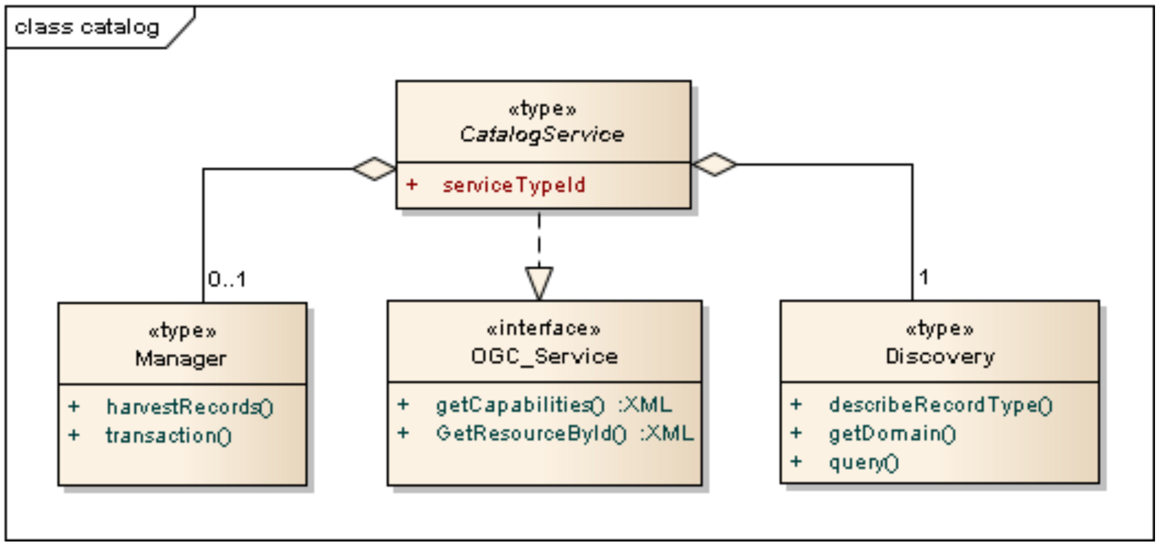


Figure 3 - General OGC catalogue UML static model

In **Figure 3 - General OGC catalogue UML static model,Error! Reference source not found.** an instance of the `CatalogService` type is a composite object that is a high-level characterization of a catalogue service. Its constituent objects are themselves components that provide functional behaviours to address particular areas of concern. A protocol binding may realise specific configurations of these components to serve different purposes (e.g. a read-only catalogue for discovery, or a transactional catalogue for discovery and publication).

The associated classes shown in this figure are mandatory or optional for implementation as indicated by the association multiplicity in the UML diagram. Therefore, a compliant catalogue service shall implement the OGC_Service, CatalogService, and Discovery classes. An application profile or protocol binding can implement additional classes associated with the Catalogue Service class. A catalogue implementation shall recognise all operations defined within each included class, and shall generate a message indicating when a particular operation is not implemented.

The protocol binding clauses of this standard provide more detail on the implementation of these conceptual interfaces. For example, the names of the classes and operations in this general UML model are changed in some of the protocol bindings. The names of some operation parameters are also changed in some protocol bindings.

Application Profiles may further specialise the implementation of these interfaces and their operations, including adding classes. In general, however, the interfaces and operations described here shall have the same semantics and granularity of interaction regardless of the protocol binding used.

The Catalogue Service class can be associated with the following classes.

a) OGC_Service class, which provides the getCapabilities operation that retrieves catalogue service metadata and the getResourceById operation that will retrieve an object by query on its identifier only. This class is always realised by the Catalogue Service class, and is thus always implemented by a Catalogue Service implementation.

b) Discovery class, which provides three operations for client discovery of resources registered in a catalogue. This class has a required association from the Catalogue Service class, and is thus always implemented by a Catalogue Service implementation. The “query” operation searches the catalogued metadata and produces a result set containing references to all the resources that satisfy the query. This operation returns metadata for some or all of the found result set. The optional describeRecordType operation retrieves the type definition used by metadata of one or more registered resource types. The optional getDomain operation retrieves information about the valid values of one or more named metadata properties.

c) Manager class, which provides two operations for inserting, updating, and deleting the metadata by which resources are registered in a catalogue. This class has an optional association from the Catalogue Service class; this interface is implemented by the Catalogue Service implementation. The transaction operation performs a specified set of “insert”, “update”, and “delete” actions on metadata items stored by a Catalogue Service implementation—this enables a “push” style of publication. The harvestResource operation requests the Catalogue Service to retrieve resource metadata from a specified location, often on a regular basis—this behaviour reflects a ‘pull’ style of publication.

The three classes associated with the Catalogue Service class allow different OGC catalogue services to provide significantly different abilities. A particular protocol binding is used by each Application Profile and a particular set of these catalogue service classes is specified by each Application Profile.

Each of the catalogue classes is described further in the following subclauses. These subclauses discuss the operations and parameters of each operation in this general model. Specific protocol bindings or application profiles can define additional parameters. For example, the HTTP Protocol Binding adds the Service, Request, and Version parameters to all operation requests to be consistent with other OGC Web Services.

7.2.2 Catalogue Service class

The Catalogue Service class provides the foundation for an OGC catalogue service. The Catalogue Service class directly includes only the serviceTypeID attribute, as specified in Table 5. In most cases, this attribute will not be directly visible to catalogue clients.

Table 5 — Attribute of Catalogue Service class

Name	Definition	Data type	Multiplicity
serviceTypeID	Identification of catalogue service binding type	String, could be URI, as controlled vocabulary for OGC services	One (Mandatory)

7.2.3 OGC_Service class

7.2.3.1 Introduction

The OGC_Service class allows clients to retrieve service metadata by providing the getCapabilities operation. This class is always realised by the Catalogue Service class, and is thus always implemented by a Catalogue Service instance. Capabilities are described further in OGC Web Service Common Implementation Specification 2.0.

NOTE This getCapabilities operation corresponds to CatalogueService.explainServer operation in OGC Catalogue version 1.1.1.

7.2.3.2 getCapabilities operation

Requirement 6 /req/getcapabilities:
Catalogue service implementations shall include a means to request structured service capability information.

Dependency: OGC Web Service Common Implementation Specification 2.0

The getCapabilities operation is specified in Table 6.

Table 6 — Definition of getCapabilities operation

Definition	Allows clients to retrieve service metadata describing Catalogue Service instance
Receives	Optional identifier(s) of requested parts of the complete service metadata document
Returns	Service metadata document for Catalogue Service instance. Some document contents depend on the set of classes that are associated with the Catalogue Service class, as defined by the specific protocol binding, and on other details of that protocol binding. Other document contents depend on the types of data defined by the specific application profile, and on other details of that profile.
Exceptions	Invalid Parameter Value, Missing Parameter Value
Pre-conditions	None
Post-conditions	Service metadata document returned to requesting client, either complete or including selected parts

The getCapabilities operation is inherited from OWS Common 2.0 and is specialized to describe service capabilities of a catalogue.

The normal GetCapabilities operation response is a service metadata document that includes the “section” attributes listed and defined in Table 7, as selected by the “section” attribute in the operation request.

Table 7 — UML attributes in getCapabilities operation normal response

Name	Definition	Data type	Optionality and use
ServiceIdentification	Metadata about this specific server	SV_ServiceIdentification in ISO 19119	Zero or one (Optional) Include when requested
ServiceProvider	Metadata about the organization operating this server	SV_ServiceProvider in ISO 19119	Zero or one (Optional) Include when requested
OperationMetadata	Metadata about an operations specified by this service, including the URL(s) for operation requests	SV_OperationMetadata in ISO 19119	Zero or more (Optional) Include when requested Repeated for each operation implemented by this server
Content	Metadata about a collection or type of resource catalogued by this server	MD_DataIdentification in ISO 19115 (adapted)	Zero or more (Optional) Include when requested Repeated for each collection and type of resources catalogued
QueryLanguage	Metadata about a query language supported by this server, specifying the query abilities implemented	Character string	Zero or more (Optional) Include when requested Repeated for each query language implemented by this server

NOTE 1 The term "Capabilities XML" document was previously used for what is here called "service metadata" document. The term "service metadata" is now used because it is more descriptive and is compliant with OGC Abstract Specification Topic 12 (ISO 19119).

NOTE 2 This general model assumes that operation failure will be signalled to the client in a manner specified by each protocol binding.

7.2.3.3 getResourceById operation

The getResourceById operation is inherited from OWS Common and supports the request of one or more resources – in this case full, structured metadata records – from the catalogue. Records are discovered through the query operation whose response includes the identifier(s) of the record(s) meeting the conditions of the query. These identifiers are passed via the getResourceById to retrieve records from the catalogue in bulk.

Requirement 7 /req/getresourcebyid:
Catalogue service implementations shall include a means to request catalogue records by their identifiers.

Dependency: OGC Web Service Common Implementation Specification 2.0

7.2.4 Discovery class

7.2.4.1 Introduction

The Discovery class allows clients to discover resources registered in a catalogue, by providing three operations named “query”, describeRecordType, and getDomain. This class has a required association from the Catalogue Service class, and is thus always implemented by all Catalogue Service implementations. All Discovery class operations are stateless.

7.2.4.2 “query” operation

The “query” operation is described in Table 8. Figure 4 provides a UML model of the “query” operation that shows the complete Discovery class with the QueryRequest and QueryResponse classes and the classes they use. The operation request includes the attributes and association role names listed and defined in the following tables. The normal operation response includes the attributes and association role names listed and defined in Table 14.

Requirement 8 /req/query:
Catalogue service implementations shall include a means to formulate a query against a catalogue and return one or more structured results.

Table 8 — Definition of "query" operation

Definition	Allows clients to ask a catalogue to execute a query that searches the catalogued metadata and produces a result set containing (zero or more) references to all the registered resources that satisfy the query. The server may maintain the result set for subsequent retrieval requests. The server may also distribute the request to other Catalogues within a federation.
Receives	Specifications of query constraints and of metadata to be returned
Returns	Number of items in result set, and/or metadata for some or all of the result set. The client can specify the maximum number of records for which metadata is returned. When metadata return is requested, the service implementation shall first sort the result set as specified by the client. Most of the metadata returned depends on the metadata requested and on the types of data defined by the specific Application Profile. The resultset may also include items coming from other Catalogues within a federation.
Exceptions	Missing Parameter Value, Invalid Parameter Value, Nonexistent collection or type
Pre-conditions	The client knows the schema of the information model that the catalogue supports and can thus form valid query expressions.
Post-conditions	Response returned to requesting client, containing number of items in result set and/or selected metadata for some or all of result set
Definition	Allows clients to ask a catalogue to execute a query that searches the catalogued metadata and produces a result set containing (zero or more) references to all the registered resources that satisfy the query. The server may maintain the result set for subsequent retrieval requests. The server may also distribute the request to other Catalogues within a federation.

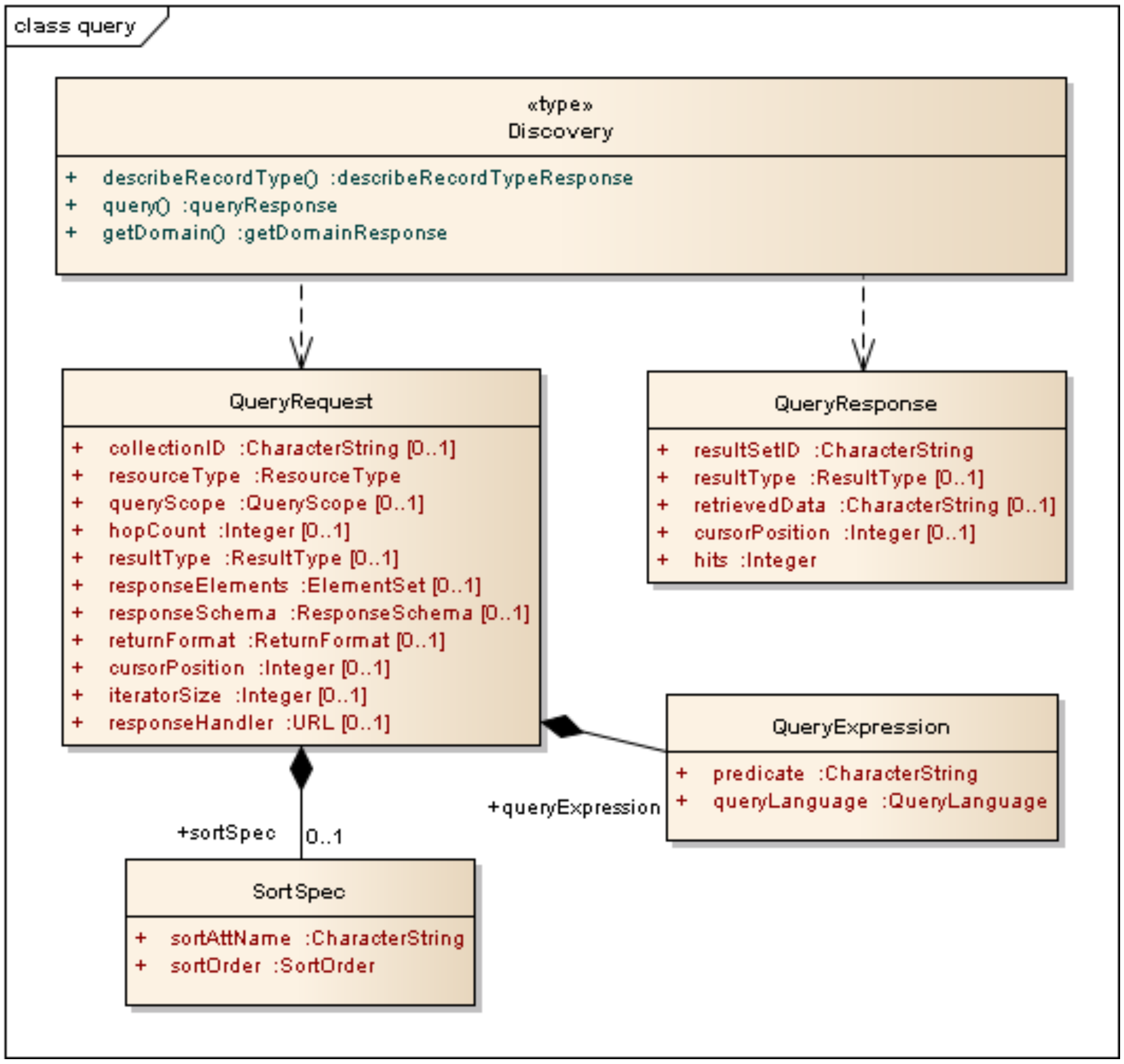

class query
«type»
Discovery
+ describeRecordType() :describeRecordTypeResponse
+ query() :queryResponse
+ getDomain() :getDomainResponse
QueryRequest
+ collectionID :CharacterString [0..1]
+ resourceType :ResourceType
+ queryScope :QueryScope [0..1]
+ hopCount :Integer [0..1]
+ resultType :ResultType [0..1]
+ responseElements :ElementSet [0..1]
+ responseSchema :ResponseSchema [0..1]
+ returnFormat :ReturnFormat [0..1]
+ cursorPosition :Integer [0..1]
+ iteratorSize :Integer [0..1]
+ responseHandler :URL [0..1]
QueryResponse
+ resultSetID :CharacterString
+ resultType :ResultType [0..1]
+ retrievedData :CharacterString [0..1]
+ cursorPosition :Integer [0..1]
+ hits :Integer
QueryExpression
+ predicate :CharacterString
+ queryLanguage :QueryLanguage
+queryExpression
+sortSpec
0..1
SortSpec
+ sortAttName :CharacterString
+ sortOrder :SortOrder

Figure 4 - "query" operation UML static model

Table 9 — UML attributes and roles in “query” operation request

Name	Definition	Data type and value	Optionality and use
queryExpression	The query language and predicate expressing query constraints	QueryExpression, See 6.2	One (Mandatory)
collectionID	Specifies the search space for this query. Search space can be all catalogue holdings or a named subspace of the catalogue holdings	Character String type, not empty Specific values that may be referenced are application profile or protocol binding dependent	Zero or one (Conditional) Include when required by protocol binding, otherwise optional
resourceType	A catalogue may contain references to several different resource types. This parameter provides for the selection of one of those types for retrieval	CodeList type [a]	One (Mandatory)
queryScope [d]	Scope of this query	**QueryScope, see** Table 10	Zero or one (optional) Zero means “local” search
resultType	Specifies how client wants result set presented and the behaviour of the catalogue as to when a response is sent	CodeList Type [b]	Zero or one (Conditional) Default values specified by protocol binding or application profile
responseElements [C]	Specifies set name or list of metadata elements to be returned in the context of a specific metadata structure	Either a list of elements as name/type pairs OR CodeList type named ElementSet with allowed values of “brief,” “summary” “full” and “browse”	Zero or one (Optional) Default value is “summary”
responseSchema [C]	The name of the “well-known” or advertised (in the capabilities) schema of the response	Code List type with one mandatory value of “OGCCORE” that represents the core catalogue schema. Other values may be defined by application profiles. Examples of such values might be: “FGDC”, “ISO-19119”, ISO-19139”, ANZLIC	Zero or one (Optional). If the parameter is not specified then the default value is “OGCCORE”.
sortSpec	Sorting information to the server for formatting data returned to the client	**SortSpec, See** Table 12	Zero or one (Optional) Default is specified by server

Name	Definition	Data type and value	Optionality and use
returnFormat	Specifies format (MIME or Internet media type) for returning result set metadata	CodeList type XML HTML TXT	Zero or one (Optional) Default is “XML” Include when results to be returned
cursorPosition	First result set resource to be returned for this operation request	Positive integer	Zero or one (Optional) Default is “1” Include when results to be returned
iteratorSize	Specifies maximum number of result set resources to be returned	Non-negative integer	Zero or one (Optional) Default is “10” Include when results to be returned
responseHandler	Network location to which the response will be forwarded when operation has been completed, for asynchronous requests	URL	Zero or one (Optional) If not included, process request synchronously

a Values and definitions of resourceType codes:
Data set – the lowest level packaging of Features that have been catalogued
Data set collection – a grouping of data sets that have commonality (ISO 19115: data set series)
Service – a set of interfaces that provide access to or operations on data (e.g. catalogue service)

b Values and definition of resultType codes and behaviours in session based environments:
validate - the QueryResponse is returned as soon as QueryRequest has been determined to be valid. Query processing continues after the QueryResponse is returned.. Reasons for failure are provided in the diagnostic of QueryResponse.
resultSetID - the QueryResponse is returned as soon as the resultSetID is available and the query has completed processing.
hits- the QueryResponse is returned as soon as the query has completed processing and the number of hits has been determined. Metadata records are not returned in the QueryResponse
results - the QueryResponse is returned as soon as the query has completed processing and the results have been formatted for return. Metadata records are returned in the QueryResponse

c The information model of this standard is the core catalogue schema defined in Subclause 6.3. It represents the common part of the information model which all application profiles shall support. This standard only supports 'OGCCORE' as the value of the 'responseSchema' parameter and a value of “brief”, “summary” or “full” for the value of the ‘responseElements’ parameter. Additional values for the responseSchema and responseElements parameters may be defined by application profiles.

d A detailed description of distributed searches can be found in Annex A of this document.

Table 10 — UML attributes in QueryScope data type

Name	Definition	Data type and value	Optionality and use
hopCount	Maximum number of message hops before distributed search is terminated. Each catalogue decrements value by one when request is received, and does not forward request if hopCount=0.	Non-negative integer	Zero or one (optional) Default value is “2” Included only when queryScope has value “distributed”

Name	Definition	Data type and value	Optionality and use
clientId	an Id which uniquely identifies the requestor.	URI	One (Mandatory)
distributedSearchId	an Id which uniquely identifies a complete client-initiated distributed search-sequence/session.	URI	One (Mandatory)
distributedSearchIdTimeout	defines how long the distributedSearchId should be valid, meaning how long a server involved in distributed search should minimally store information related to the distributedSearchId	Long	Zero or one (optional).
federatedCatalogues	To restrict the number of catalogues of a federation which should be searched upon in a distributed query an optional list of those catalogues can be provided here	FederatedCatalogues	Zero or more (Optional)

Table 11 — UML attributes in FederatedCatalogues data type

Name	Definition	Data type and value	Optionality and use
catalogueURL	a catalogue is represented by it's url.	URL	One (Mandatory)
timeout	timeout (in msec) how long a server should wait for a catalogue request to be proceeded before throwing a timeout exception	Long	Zero or one (Optional)

Table 12 — UML attributes in SortSpec data type

Name	Definition	Data type and value	Optionality and use
sortAttName	Identifies the result set attribute type to be sorted on	Character String	Zero or one (Optional) Default is defined by server
sortOrder	How the attributes are to be ordered by the sort	Code List type with allowed values of "ascending" and "descending"	Zero or one (Optional) Default is defined by server

Table 13 –UML attributes in QueryExpression data type

Name	Definition	Data type and value	Optionality
queryLanguage	Specifies the predicate language and version used in a query expression	Code List, known values of "OGC_Common", "Filter, Type-1"	One (Mandatory)
predicate	The constraint expression for selecting entries from a catalogue	CharacterString	One (Mandatory)

Table 14 — UML attributes and roles in "query" operation normal response

Name	Definition	Data type and value	Optionality and use
resultType	How the server responded to the query request.	CodeList type with allowed values of "dataset", "datasetcollection" and "service"	Zero or one (Optional)
retrievedData	A subset of the results of this query request, organised and formatted as specified in the presentation, messageFormat, and sortField parameters.	ReturnData Set of resource descriptions/records	Zero or one (Conditional) Include when resultType = Results
cursorPosition	Last result set resource returned for this operation request.	Positive integer	Zero or one (Conditional) Include when results returned
hits	Number of entries in the result set.	Non-negative integer	One (Mandatory)

NOTE This general model assumes that operation failure will be signalled to the client in a manner specified by each protocol binding.

7.2.4.3 describeRecordType operation

The describeRecordType operation is more completely specified in **Table 15 — Definition of describeRecordType operation**.

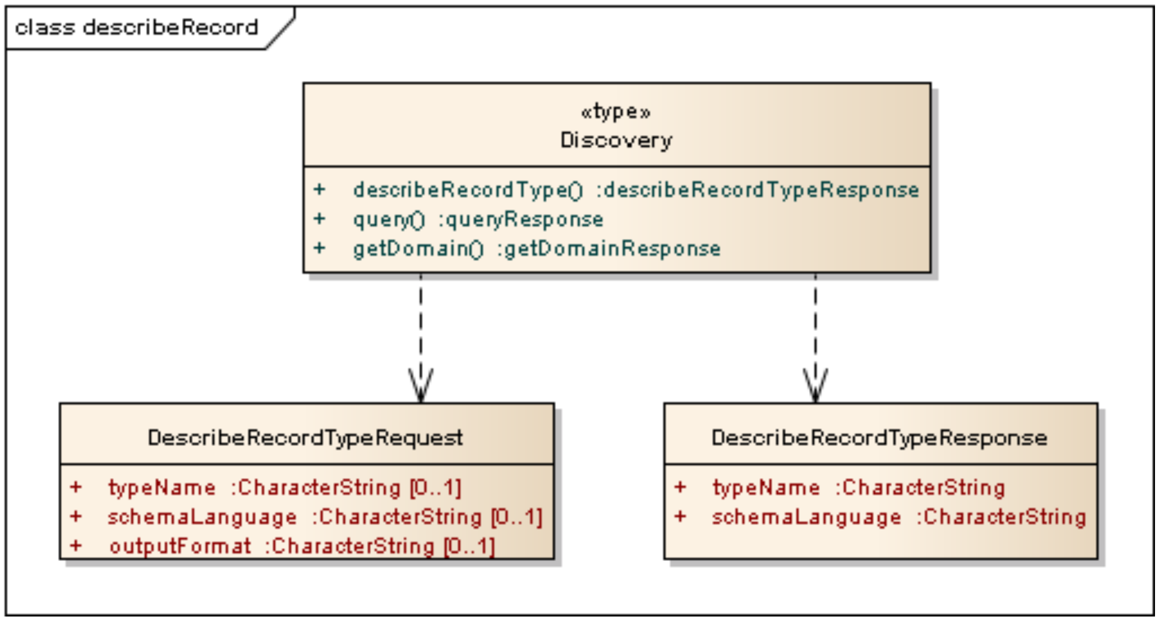
Table 15**Error! Reference source not found.** provides a UML model of the optional describeRecordType operation that shows the complete Discovery class with the DescribeRecordTypeRequest and DescribeRecordTypeResponse classes and the class they use. The operation request includes the attributes and association role name listed and defined in Table 16. The normal operation response includes the attributes and association role name listed and defined in Table 17.

NOTE The describeRecordType operation corresponds to CG_Discovery.explainCollection operation in OGC Catalogue version 1.1.1.

Requirement 9 /req/describe-records:
Catalogue service implementations should include a means to describe or reference the structure (schema), queryables, element sets, and formats of the metadata used for one or more registered resource types.

Table 15 — Definition of describeRecordType operation

Definition	Allows clients to retrieve type definition(s) used by metadata of one or more registered resource types
Receives	Optional identifications of requested record type(s) and of desired format
Returns	Type definition document containing definition(s) of type(s) used by the metadata of one or more registered resource types. This type definition shall include the structure (schema), queryables, element sets, and formats of the metadata used for one or more registered resource types. The contents of the result of this operation depend on the types of metadata that can currently be used by registered resources.
Exceptions	Missing Parameter Value, Invalid Parameter Value, Nonexistent type
Pre-conditions	None
Post-conditions	Type definition document returned to requesting client, containing definition(s) of type(s) used by the metadata of one or more registered resource types



class describeRecord
«type»
Discovery
+ describeRecordType() :describeRecordTypeResponse
+ query() :queryResponse
+ getDomain() :getDomainResponse
DescribeRecordTypeRequest
+ typeName :CharacterString [0..1]
+ schemaLanguage :CharacterString [0..1]
+ outputFormat :CharacterString [0..1]
DescribeRecordTypeResponse
+ typeName :CharacterString
+ schemaLanguage :CharacterString

Figure 5 - describeRecordType operation UML static model

Table 16 — UML attributes and role in describeRecordType operation request

Name	Definition	Data type and value	Optionality and use
typeName	Name of metadata record type(s) for which type information is to be returned	Character String type Values specified by protocol binding	Zero or more (Optional) Return all types when omitted
schemaLanguage	The schema language of the response message	Character String type Values specified by protocol binding	Zero or one (Optional) Use XML Schema when omitted

Name	Definition	Data type and value	Optionality and use
outputFormat	Document format for output	Character String type Value is MIME type	Zero or one (Optional) Use application/xml when omitted.

Table 17 — UML attributes and role in describeRecordType operation response

Name	Definition	Data type and value	Optionality and use
typeName	Name of metadata record type for which type information is returned	Character String Values are names of metadata record types	One or more (Mandatory) Include one for each record type to be returned
schemaLanguage	The schema language used to describe the type	Character String. Non-empty Values specified by protocol binding	One (Mandatory).

7.2.4.4 getDomain operation

The optional getDomain operation is more completely specified in Table 18, which **Error! Reference source not found.** provides a UML model of the getDomain operation that shows the complete Discovery class with the GetDomainRequest and GetDomainResponse classes and the class they use. The operation request includes the attributes listed and defined in Table 19. The normal operation response includes the attributes and association role name listed and defined in Table 20.

Requirement 10 /req/getdomain:
Catalogue service implementations should include a means to retrieve the domain (allowed values) of a metadata property or request parameter at the time the request is invoked.

Table 18 — Definition of getDomain operation

Definition	Allows clients to retrieve the domain (allowed values) of a metadata property or request parameter at the time the request is invoked. The returned information may be static domain information, but may also be dynamic in that the allowed values are determined at runtime. The operation does a *best attempt* at returning information about a metadata property or request parameter.

Receives	Names of one or more requested metadata properties or request parameters.
Returns	Descriptions of domains of one or more requested metadata properties or request parameters
Exceptions	Missing Parameter Value, Invalid Parameter Name
Pre-conditions	None
Post-conditions	Descriptions of domains returned to requesting client, containing the domain descriptions for all the identified metadata properties or request parameters.

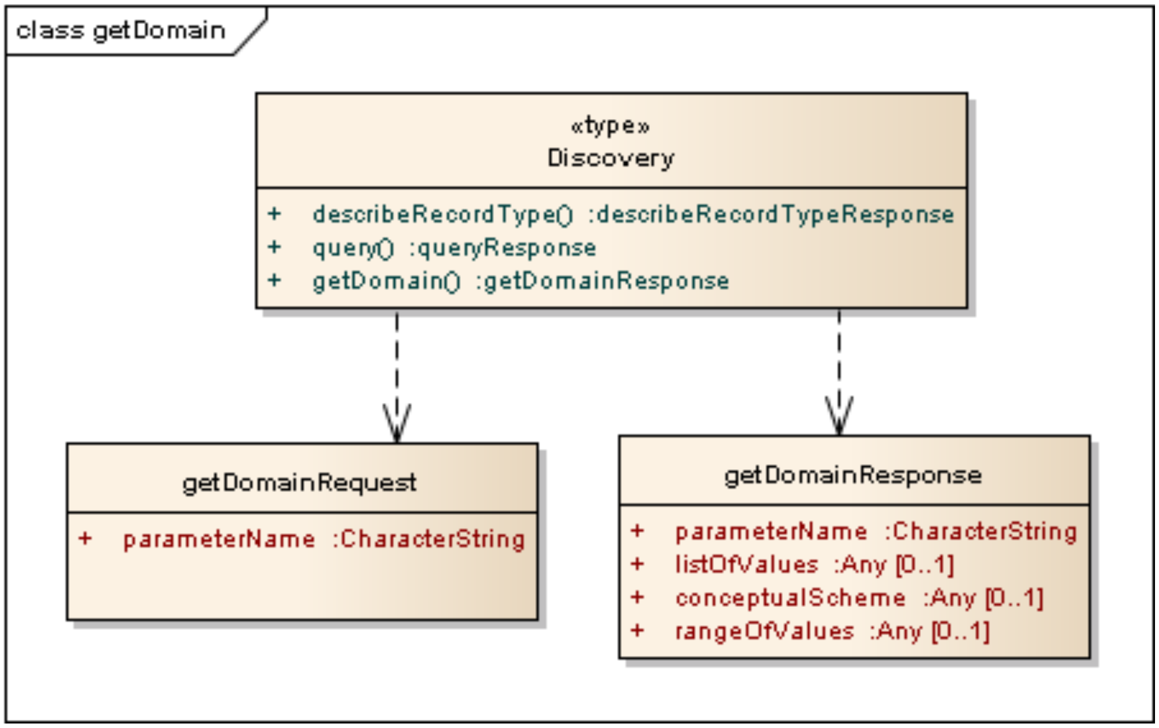

class getDomain
«type»
Discovery
+ describeRecordType() :describeRecordTypeResponse
+ query() :queryResponse
+ getDomain() :getDomainResponse
getDomainRequest
+ parameterName :CharacterString
getDomainResponse
+ parameterName :CharacterString
+ listOfValues :Any [0..1]
+ conceptualScheme :Any [0..1]
+ rangeOfValues :Any [0..1]

Figure 6 - getDomain operation UML static model

Table 19 — UML attribute in getDomain operation request

Name	Definition	Data type and value	Optionality
parameterName	The name of a metadata property or request parameter	Character string. Non-empty Allowed values specified by protocol binding	One (Mandatory)

Table 20 — UML attributes and role in getDomain operation normal response

Name	Definition	Data type and value	Optionality and use
parameterName	Name or identifier of metadata property or request parameter	Character String type, not empty	One (Mandatory)
listOfValues	Unordered list of domain values	Data type of list elements depends on the data type of the parameter whose domain is being described	Zero or one (Optional) [a]

Name	Definition	Data type and value	Optionality and use
conceptualScheme	Reference to an authoritative list of domain values for the specified parameter	Data type of list of values in the authoritative list depends on the data type of the parameter whose domain is being described	Zero or one (Optional) [a]
rangeOfValues	Range of domain values expressed by specifying a minimum and maximum value	Data type of the minimum and maximum values depends on the data type of the parameter whose domain is being described	Zero or one (Optional) [a]
a For any single parameter, only one of listOfValues, conceptualScheme or rangeOfValues should be used to describe the value domain.			

7.2.5 Manager class

7.2.5.1 Introduction

The Manager class allows a client to insert, update and/or delete catalogue content. This class has an optional association from the CatalogueService class; it is not required that a catalogue service implement publishing functionality. Two operations are provided: “transaction” and “harvestResource”. Both are optional operations.

The “transaction” operation allows a client to formulate a transaction, and send it to the catalogue to be processed. The transaction may contain metadata records and elements of the information model that the catalogue understands. To use the transaction operation, the client must know something about the information model that the catalogue implements.

The “harvestResource” operation, on the other hand, directs the catalogue to retrieve an accessible metadata record and processes it for inclusion in the catalogue, perhaps periodically re-fetching the metadata records to refresh the information in the catalogue. The client does not need to be aware of the information model of the catalogue when using the “harvestResource” operation, since the catalogue itself is doing the work required to process the information. The client is simply pointing to where the metadata resource to be harvested is.

Requirement 11 /req/transaction:
Catalogue service implementations shall include a means to request a specified set of “insert”, “update”, and “delete” actions on the content managed by a Catalogue Service instance.

7.2.5.2 ”transaction” operation

The “transaction” operation is more completely specified in Table 21. Figure 7 provides a UML model of the “transaction” operation that shows the complete Manager class with the TransactionRequest and TransactionResponse classes and the classes they use. The operation request includes the attributes listed and defined in Table 22. The normal operation response includes the attributes listed and defined in Table 23.

Table 21 — Definition of "transaction" operation

Definition	Allows clients to request a specified set of "insert", "update", and "delete" actions on the content managed by a Catalogue Service instance.
Receives	Specification of set of "insert", "update", and "delete" actions, plus an optional identifier. At least one action shall be included.
Returns	A summary of the transaction results that identifies newly created entries when applicable. Most contents of the result depend on the types of data defined by the specific protocol binding and Application Profile.
Exceptions	Missing Parameter Value, Invalid Parameter Value, Transaction Failed
Pre-conditions	User is authorized to modify catalogue contents
Post-conditions	Catalogue entries are inserted, updated, and/or deleted as requested, and the integrity and consistency of catalogue contents are preserved..

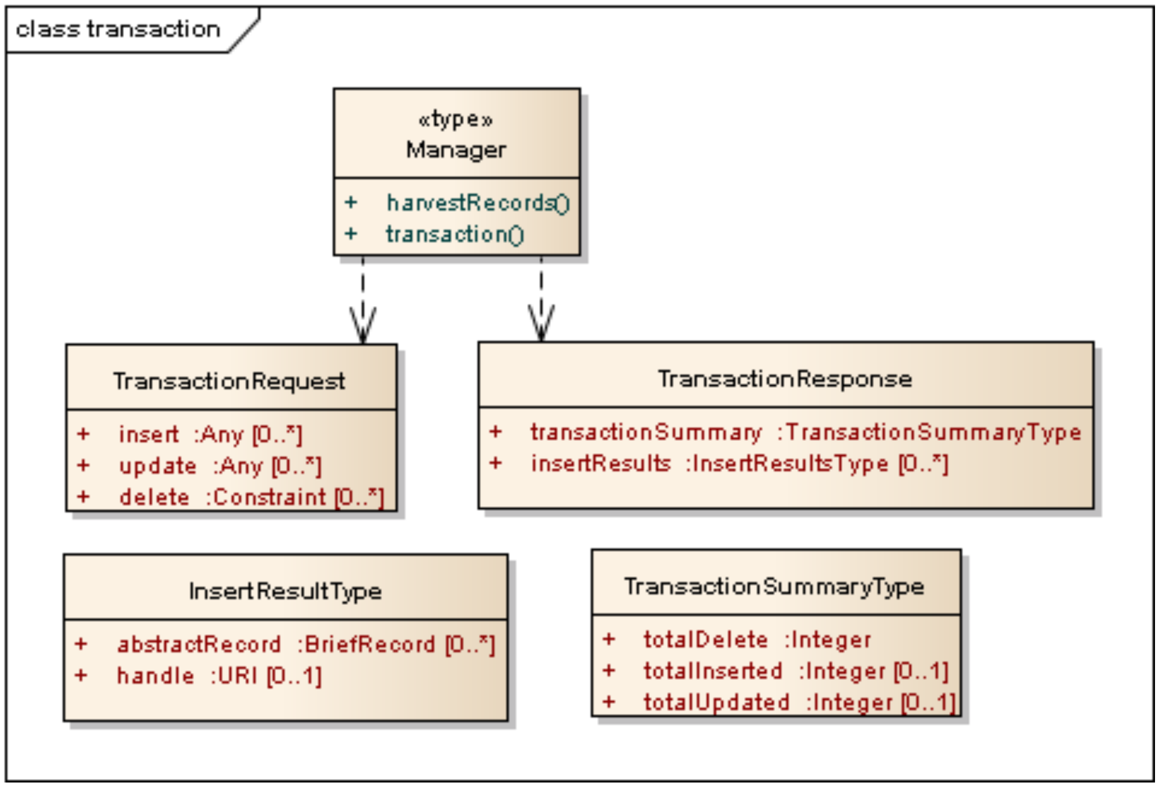


Figure 7 - "transaction" operation UML static model

Table 22 — UML attributes in “transaction” operation request

Name	Definition	Data type and value	Optionality and use
insert	The insert action is used to create new metadata records in a catalogue. Each insert action may contain one or more new metadata record instances that are to be inserted into the catalogue.	Any, a container for one or more metadata record instances The schema for metadata records is defined in the protocol binding and may be extended or redefined in an Application Profile	Zero or more (Optional) Include when client wishes to insert one or more new catalogue records
update	The update action is used to modify existing records in the catalogue. The update action contains a single new metadata record instance and a predicate that defines the set of catalogue records that will be modified. The predicate may identify zero or more records that are to be modified by the update action. The encoding of the predicate is specified in the protocol binding and may be further qualified or extended in an Application Profile.	Any, contains one instance of a metadata record that will be used to update existing records in catalogue The schema of the record is defined in the protocol binding and may be extended or redefined in an Application Profile	Zero or more (Optional) Include when client wishes to modify one or more existing catalogue records
delete	The delete action is used to remove one or more records from a catalogue. The records to be removed are identified by specifying a predicate with the operation. The predicate may identify zero or more records that are to be removed from the catalogue by the delete action. The encoding of the predicate is specified in the protocol binding and may be further qualified or extended in an Application Profile.	The delete action requires a constraint predicate that identifies the records in the catalogue to be removed	Zero or one (Optional) Include when client wishes to delete one or more existing records from a catalogue

Table 23 — UML attributes in “transaction” operation normal response

Name	Definition	Data type and value	Optionality
transaction Summary	Summary of transaction results that includes the numbers of records inserted, updated, and deleted by the actions specified in the transaction	TransactionSummaryType Total number of records inserted, updated, and deleted (Integer)	One (Mandatory)
insertResults	Brief representation of a record created by the transaction, which **shall** include the record identifier May contain a handle that relates newly created record with the insert action that created it	InsertResultType Structure composed of brief record type (application profile or protocol binding dependent) and an optional handle	Zero or more (Optional) Include one for each record created

7.2.5.3 harvestResource operation

The harvestResource operation facilitates the retrieval of remote resources from a designated location and provides for optional transactions on the local catalogue. The harvestResource operation is described in Table 24. Figure 8 provides a UML model of the "harvestResource" operation that shows the complete Manager class with the HarvestResourceRequest and HarvestResourceResponse classes. The operation request includes the attributes listed and defined in Table 25. The normal operation response includes the attributes listed and defined in Table 26.

Requirement 12 /req/harvest:
Catalogue service implementations shall include a means to retrieve a resource from a specified remote location, and to create one or more entries for that resource in the catalogue.

Table 24 — harvestResource operation

Definition	Allows a user to request that a catalogue service attempt to retrieve a resource from a specified location, and to optionally create one or more entries for that resource. A harvest attempt may occur periodically if an interval is specified.
Receives	A request message containing the source of the resource to be harvested
Returns	An acknowledgement that a harvestRequest has been received and validated (if a responseHandler is specified) or a summary of the harvest results that identifies newly harvested records (if a responseHandler is not specified). Most contents of the result depend on the types of data defined by the specific protocol binding and Application Profile.
Exceptions	InvalidRequest, ResourceNotFound
Pre-conditions	The user is permitted to modify catalogue contents, unless the scope of the harvest does not include an insert or update transaction
Post-conditions	One or more records are harvested from a remote system and optionally new catalogue entries are created or existing entries are updated, and the integrity and consistency of the catalogue contents are preserved

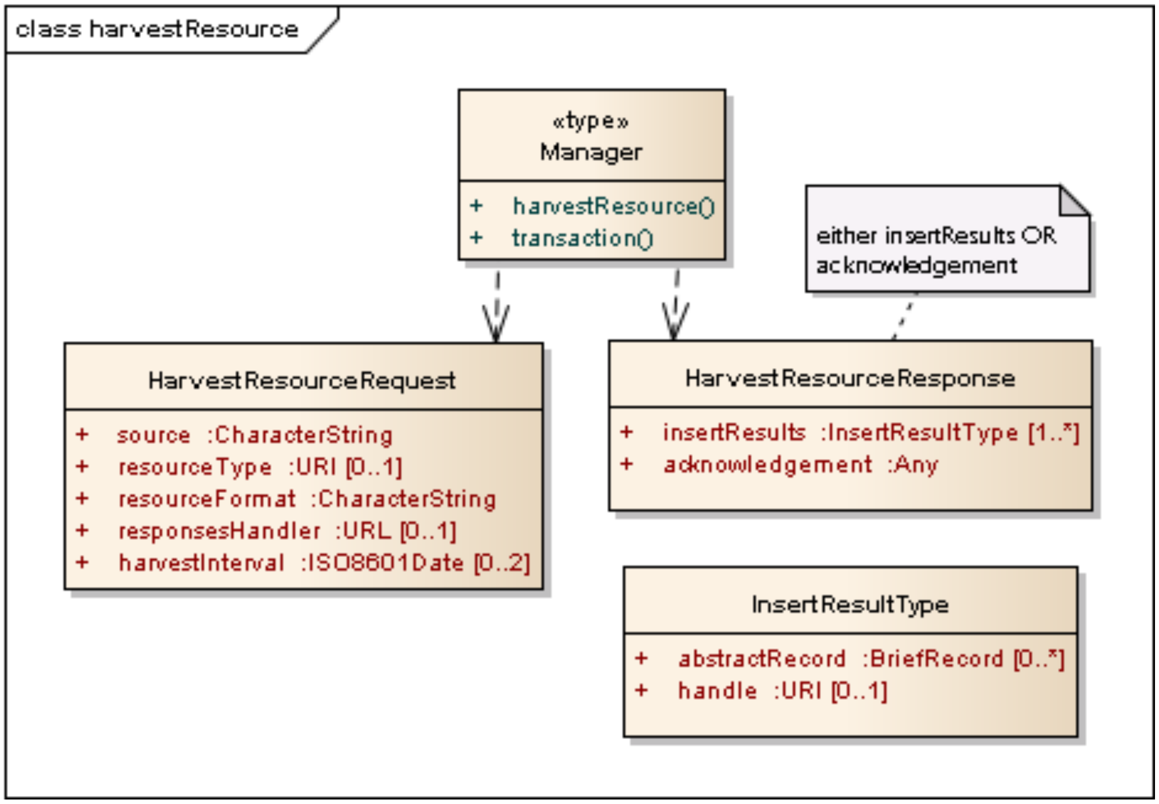

class harvestResource
«type»
Manager
+ harvestResource()
+ transaction()
either insertResults OR acknowledgement
HarvestResourceRequest
+ source :CharacterString
+ resourceType :URI [0..1]
+ resourceFormat :CharacterString
+ responsesHandler :URL [0..1]
+ harvestInterval :ISO8601Date [0..2]
HarvestResourceResponse
+ insertResults :InsertResultType [1..*]
+ acknowledgement :Any
InsertResultType
+ abstractRecord :BriefRecord [0..*]
+ handle :URI [0..1]

Figure 8 - harvestResource operation UML static model

Table 25 — UML attributes in harvestResource operation request

Name	Definition	Data type and value	Optionality and use
source	Location from which resource to be retrieved	URL	One (Mandatory)
resourceType	Identifier of type of resource to be harvested, if known	URI	Zero or one (Optional) If the parameter is not specified then the catalogue should determine the resourceType from the content of the message
resourceFormat	Identifier of media type indicating the format of resource to be harvested	CharacterString Value shall be a media type supported by catalogue	One (Mandatory)
responseHandler	Network location to which the response will be forwarded when operation has been completed, for asynchronous requests	URL	Zero or one (Optional) If not included, process request synchronously
harvestInterval	Time interval between harvest attempts	Period Using ISO 8601 Period syntax (e.g., P6M indicates an interval of six months)	Zero or one (Optional) If the parameter is not specified then the catalogue should harvest the resource once in response to the request.

Table 26 — UML attributes in harvestResource operation normal response

Name	Definition	Data type and value	Optionality and use
acknowledgement	Summary of transaction results, with contents depending on the protocol binding and Application Profile (e.g. total records affected by each action)	Any	One (Mandatory)
insertResults	Brief representation of a record created by the transaction, which **shall** include the record identifier May contain a handle that relates newly created record with the insert statement that created it	InsertResultType A structure composed of the brief record type (application profile or protocol binding dependant) and an optional handle	One or more (Mandatory) Include one for each new record created in catalogue

NOTE This general model assumes that operation failure will be signalled to the client in a manner specified by each protocol binding.

8. Conformance classes and specialisation

8.1 Introduction

This subclause provides an overview of the core elements of the General Catalogue Model and how these may be used in protocol bindings and application profiles.

The General Catalogue Model consists of an abstract model and a General Interface Model. The abstract query model specifies a BNF grammar for a minimal query syntax and a set of core search attributes (names, definitions, conceptual datatypes). The General Interface Model specifies a set of service interfaces that support the discovery, access, maintenance and organization of catalogues of geospatial information and related resources; these interfaces may be bound to multiple application protocols, including the HTTP protocol that underlies the World Wide Web.

Implementations are constrained by the protocol binding parts of this standard, which depend on this general model. Each protocol binding includes a mapping from the general interfaces, operations, and parameters specified in this clause to the constructs available in a chosen protocol. Application profiles are intended to further document implementation choices.

An Application Profile is based on one of the protocol bindings in the base specification. In the case of the Catalogue Services Standard, a profile should reference the HTTP/1.1 protocol binding unless others are defined/recognized. In most, but not all, protocol bindings, there may be restrictions or refinements on implementation agreed within an implementation community. A graphic model of the relationships is shown in Figure 9.

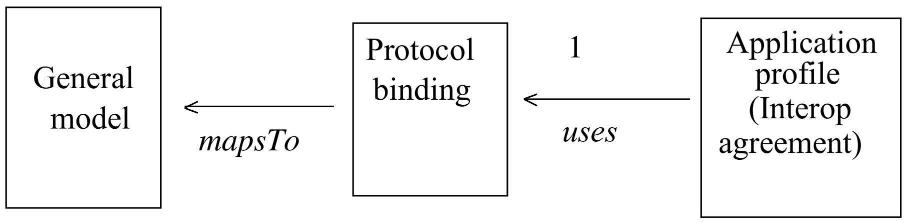


Figure 9 - Relationship of general model, protocol binding, and application profile

8.2 List of requirements and conformance classes

The general model conformance classes are presented in Table 27 — Catalogue Service Requirements (Conformance Classes) along with the corresponding requirement classes. These conformance classes shall be realized in the protocol binding parts of this standard. Test suites are to be included in protocol binding parts of the specification that reference these general conformance clauses (requirements).

Table 27 — Catalogue Service Requirements (Conformance Classes)

No	Conformance class	Requirement Class URI	Description
1	query-language	http://www.opengis.net/spec/cat/3.0/req/base/query-language	Catalogue service query interfaces shall support a published syntax for processing full text and fielded query.
2	common-query-language	http://www.opengis.net/spec/cat/3.0/req/common-query language	Catalogue service query interfaces shall support a catalogue query syntax that is transformable to the BNF expressed in subclause 6.2.2
3	classified-as-operator	http://www.opengis.net/spec/cat/3.0/req/classified-as	Catalogue service query interfaces may be extended to support a classified-as operator to support queries based on taxonomies
4	common-queryables	http://www.opengis.net/spec/cat/3.0/req/base/common-queryables	Catalogue service query interfaces shall support the set of common queryable elements described in Tables 1-3. Services shall perform appropriate mapping of public query terms to internal equivalents to enable general search of catalogues.
5	common-returnables	http://www.opengis.net/spec/cat/3.0/req/base/common-returnables	Catalogue service query interfaces shall support the set of common returnable elements in result set metadata as described in Table 4. Service shall perform appropriate mapping of internal fields, as necessary, to published returnables to enable interoperable search of catalogues
6	get-capabilities	http://www.opengis.net/spec/cat/3.0/req/base/get-capabilities	Catalogue service implementations shall include a means to request structured service capability information
7	getresourcebyid	http://www.opengis.net/spec/cat/3.0/req/base/getresourcebyid	Catalogue service implementations shall include a means to request catalogue records by their identifiers
8	query	http://www.opengis.net/spec/cat/3.0/req/base/query	Catalogue service implementations shall include a means to formulate a query against a catalogue and return one or more structured results

No	Conformance class	Requirement Class URI	Description
9	describe-records	http://www.opengis.net/spec/cat/3.0/req/base/describe-records	Catalogue service implementations shall include a means to describe or reference the structure (schema), queryables, element sets, and formats of the metadata used for one or more registered resource types
10	getdomain	http://www.opengis.net/spec/cat/3.0/req/getdomain	Catalogue service implementations shall include a means to retrieve the domain (allowed values) of a metadata property or request parameter at the time the request is invoked.
11	transaction	http://www.opengis.net/spec/cat/3.0/req/transaction	Catalogue service implementations shall include a means to request a specified set of "insert", "update", and "delete" actions on the content managed by a Catalogue Service instance
12	harvest	http://www.opengis.net/spec/cat/3.0/req/harvest	Catalogue service implementations shall include a means to retrieve a resource from a specified remote location, and to optionally create one or more entries for that resource in the catalogue

The following conformance classes are recommended for Catalogue Services, with respect to the Requirements listed within this standard.

Conformance Class	Requirements
Baseline	1, 2, 4, 5, 6, 7, 8
OpenSearch	1, 6, 7, 8
Transactional	1, 4, 5, 6, 7, 8, 9, 11, 12

8.3 Interface definitions

The various elements of the General Catalogue Interface Model provide functional behaviours and capabilities to address particular areas of concern. A protocol binding may realise specific configurations of these components to serve different purposes (e.g. a read-only catalogue for discovery, a transactional catalogue for discovery and publication).

A compliant protocol binding of the catalogue service is required to implement the OGC_Service, Catalogue Service, and Discovery classes. A protocol binding may also include any of the optional classes associated with the Catalogue Service class. A compliant implementation of a protocol binding shall recognise all operations defined within each class included in the protocol binding, and shall generate a service exception report indicating when a particular operation is not implemented (in such cases the operation is abstract—an implementation is not required).

The protocol binding parts of this standard provides more detail on the implementation of the general interfaces. In effect, each binding maps these interfaces to a particular application protocol. For example, the names of the classes and operations in this general UML model are changed in some of the protocol bindings. The names of some operation parameters are also changed in some protocol bindings. However, the interfaces and operations specified in all Protocol Bindings shall be consistent with the semantics and granularity of interaction specified in the General Interface Model.

Application profiles, which will appear as separate documents may further specialise the implementation of these interfaces and their operations, including adding classes and parameters. However, the application profile is a specialization of the parent protocol binding, in that the names of the operations and the parameters cannot be changed.

8.4 Query model components

8.4.1 Query language/model

Many OGC service operations have the requirement to pass and process a query as a structure to perform a search. There are several query languages and messaging mechanisms identified within OGC standards. Binding protocols and application profiles should be explicit about the selected query languages and any features peculiar to a scope of application. The following items should be addressed in specialization of a Protocol Binding or an Application profile with respect to query language support.

a) Support for “abstract” queries, against well-known access points (e.g. core search properties). Some standards promote or require the exposure of well-known field-like objects as common search targets (queryables), allowing interrogation of a service without prior negotiation on information content. The mandatory queryable attributes which shall be recognised by all OGC Catalogue Services is discussed in Subclause 6.3.

b) Selection of a query language. Identify the name and version of required query language(s) anticipated by this Protocol Binding or Application Profile for use.

c) Supported data types (e.g., character, integer, coordinate, date, polygon) and operator types (e.g., inequality, proximity, partial string, spatial, temporal). Query languages may be restricted in their implementation or extended with functions not described in the base specification. This would need to be done if the base query language did not support a data type required by the OGC CommonQL discussed in Clause 6 such as envelope.

In addition, an application profile may extend the OGC CQL or Filter syntax with functions not described in the base specification through use of the “function “construct in CQL and the “Filter“ language. If an application protocol uses this extension method, the profile documentation should include an updated BNF grammar in addition to lists or reference documents with the enumerated data types and operator types required by this Application Profile. In addition, any description of special techniques (e.g. supporting joins or associations) that are expected by an Application Profile should be described.

8.4.2 Common search and retrieval elements

The abstract information model is discussed in Clause 6; this model consists of a small set of abstract search elements and the specification of a common “summary” element set to allow queries across protocol bindings and even from outside the OGC domain. Each Protocol Binding should specialize this model by:

a) Specify the syntax of the globally unique Identifiers including any registration authority information

b) Map the core search (queryable) elements into a concrete syntax based on the chosen record format(s)

c) Define a “summary” element set that corresponds to the “summary” element set in the Catalogue general model

An application profile is expected to fully specify the conceptual information model adopted by the user community. This process and resulting artefacts are further discussed in Subclause 6.2.5 and the remainder of this clause.

8.5 Catalogue Application Profiles

ISO TR 10000-1:1998 describes a general framework for functional standardization and defines the concept of a profile. A profile identifies the use of particular options available in one or more base standards and it also provides a basis for developing conformance tests; a compliant profile shall not contradict the base specifications or otherwise give rise to non-conforming conditions. An *application profile* specifies the use of an application-layer protocol (e.g. HTTP/1.1) in order to provide for the structured transfer of information between systems (ISO/IEC TR 10000-2:1998).

A catalogue application profile binds a set of functional components (with interfaces specified as part of a protocol binding) to an abstract information model—expressed using UML—that has one or more concrete representations of catalogue content. Each representation is an Internet media type that conforms to a schema defined using some schema language (e.g., ASN.1, XML Schema, RDF Schema). An application profile specifies a set of functional components that are provided by a conforming implementation (Figure 10).

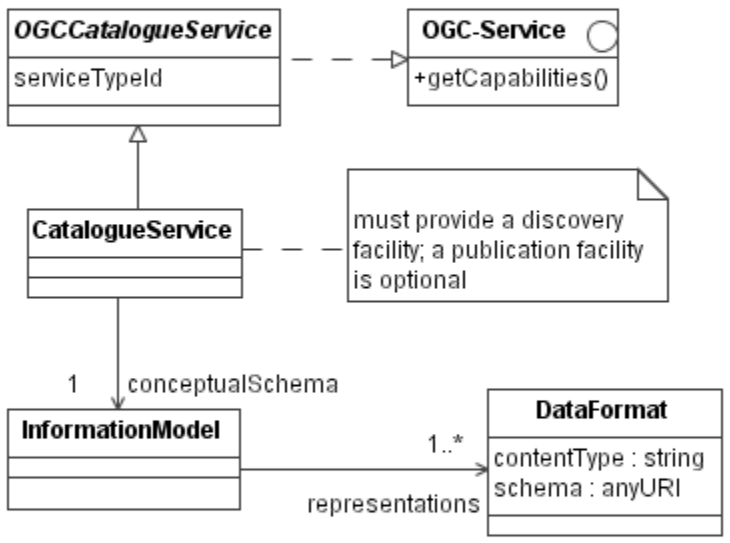


Figure 10 - Application profiles specify concrete catalogue services

An application profile is derived from one or more base specifications in order to address particular needs or requirements. The general OGC catalogue model defines common behaviours and interfaces that have general utility, but in practice there is no single solution that fits everyone's needs. Catalogue application profiles specify refinements or extensions that are targeted toward specific implementation communities; for these communities it is the application profile that represents the standard for conformance. Following ISO 19106, a **Level 1** profile is defined as a pure subset of one or more ISO standards; a **Level 2** profile includes allowable extensions and may also depend on non-ISO standards.

Clause 10 in the ISO 19119 standard distinguishes *platform-neutral* from *platform-specific* specifications and assumes that one of the former will constitute the basis for one or more of the latter. That is, a single platform-neutral specification will give rise to multiple platform-specific specifications each of which is bound to a particular distributed computing protocol (i.e. HTTP). The OGC catalogue framework upholds this basic distinction: the general interface model is a platform-neutral description of catalogue operations; each application profile is platform-specific—it makes use of one of the protocol bindings defined in the catalogue standard.

Note that in **Figure 10 - Application profiles specify concrete catalogue services** application profiles will reflect differing degrees of "thickness". For example, if a profile employs a very simple conceptual model that embodies a limited set of simple properties then its 'native' representation may include little more than the common search and retrieval elements. Profiles that utilize more sophisticated models will define a native representation that provides more information; in this case the common search and retrieval elements shall be mapped to the catalogue information model.

8.6 Structure and format

8.6.1 Introduction

All application profiles shall be structured as shown in Table 28. This organization complies with clause 12.3 of ISO 19106 (*Geographic information – Profiles*). A profile may introduce additional (sub)clauses as required.

Table 28 — Structure of an application profile

Clause	**Title**
(front matter)	Abstract
	Keywords
	Preface
	Document terms and definitions
	Submitting organizations
	Submitters
1	Scope
2	Conformance
3	Normative references
4	Terms and definitions
5	Symbols and abbreviations
6	System context 6.1 Application domain 6.2 Essential use cases
7	Information models 7.1 Capability classes 7.2 Catalogue information model 7.3 Supported data bindings 7.4 Service information model 7.5 Native language support
8	External interfaces 8.1 Imported protocol bindings 8.2 Interface A 8.3 Interface B . . . 8.i Query facilities 8.j General implementation guidance 8.k Security considerations
Annex A	Abstract test suite (normative)
Annex B	Design rationale (informative)
Annex C	Revision history
Annex D	Bibliography

Clauses 6 through 8 convey the particulars of the application profile in terms of three 'views' (these correspond to the following standard ODP viewpoints: Enterprise, Information, and Computational). The three views describe various aspects of the catalogue service with respect to the base specifications; taken together they constitute the basic application architecture. The essential content of these views is summarized in the following subclauses; additional guidance can be found in the annotated profile template (OGC Document 03-101).

8.6.2 System context

This view focuses on the purpose, scope, and policies of the catalogue service (i.e., what is the system used for). It documents special requirements[8] and describes the context of use as suggested in Table 29.

Table 29 — System context: required subclauses

Subclause	Topical content
Application domain	The subject domain being addressed—identify whether this profile has a specific disciplinary focus (e.g. oceanography), or is of interest to a broader community (e.g. research, public access, or libraries) The prospective stakeholders or community of practice
Essential use cases	What the system should be able to do, what it will be used for, who will use it Typical scenarios that encompass a series of interactions between users and the catalogue system being described in order to fulfill the needs of stakeholders. The inclusion of narrative use cases with accompanying interaction and/or sequence diagrams is recommended.

8.6.3 Information models

This view primarily focuses on the information structures and the semantics of information processing (i.e., what the system is about); it describes the public information model that is employed by the catalogue service. The syntax for all supported representations of the catalogued resources shall also be defined (Table 30).

A section describing the MIME-types to be used is mandatory for any standard involving data encodings. If no suitable MIME type exists in http://www.iana.org/assignments/media-types/index.html then this section may be used to define a new MIME type for registration with IANA.

[8] Clause 7 of ISO 19106 stipulates that a profile must clearly identify the specific user requirements that are satisfied by the profile.

Table 30 — Information models: required subclauses

Subclause	Topical content
Capability classes	Capabilities provided by the application profile (and conformance classes/levels if these are distinguished)
Catalogue information model	Kinds of information objects managed by the catalogue using UML notation—a catalogue may offer discovery and publication support for many different types of information resources (services, data sets, schemas, style sheets, reference documents, software components, ontologies, thesauri, etc.) Mappings to the common XML Record format
Supported data formats	Supported representations of the information objects using an appropriate syntax, one of which shall be designated as the default representation Supported element sets (schemas) for each format
Service information model	Content model and syntax for service information
Native language support	How the catalogue service supports multiple languages and character encodings (i.e. internationalization and localization issues)

8.6.4 External interfaces

This view primarily focuses on documenting the externally visible behaviour of the system, including the interfaces provided by its components and the supported protocol binding(s). This view shall define the request and response message structures as part of the operation signatures; it also documents supported query facilities and any relevant security considerations (Table 31). Most of the request and response message elements are imported with the protocol binding, but a (Level 2) profile may introduce extensions to meet more specialized requirements.

Table 31 — Public interfaces: required subclauses

Subclause	Topical content
Imported protocol binding	How the interfaces or functions specified for the profile are related to the imported protocol binding.
Interface specifications	Syntax and semantics of the operations provided by each interface, including relevant preconditions, postconditions, and other usage constraints Formal, language-independent interface specifications that admit multiple programming language bindings (e.g. W3C WSDL, OMG IDL) Error conditions that can be raised and how they're handled Any restrictions or variations on the use of the supported protocol binding (e.g. HTTP/1.1)
Query facilities	Supported query languages (e.g. OGC CQL/Filter, SQL-92, XPath, XQuery, etc.) extensions or restrictions to any of the above languages
Implementation guidance	Any additional information (typically non-normative) that may be helpful to implementers
Security considerations	Information regarding the provision of security functions: authentication, access control, message integrity, confidentiality, non-repudiation, audit trails

The inclusion of a UML diagram is recommended to provide an overview of the interfaces provided by a given service type, where each type provides a different—perhaps overlapping—set of interfaces (e.g. a read-only catalogue, a catalogue that allows a ‘push’ style of publication).

8.7 Compliance

A compliant application profile shall:

a) Include the (sub)clauses indicated in Table 31 (additional clauses MAY also be included);

b) Define the supported catalogue information model using UML as the conceptual schema language;

c) Define a set of mappings for the common XML record format data format;

d) Specify the ‘native’ representation of information model elements (additional representations MAY also be specified);

e) Define any extensions to the imported protocol binding;

f) Indicate how the elements of the general model are related to the corresponding elements of the profile-specific interfaces; and

g) Include a conformance test suite (web-based services can do so using the OGC CITE notation).

9. Annex A - BNF Definition of OGC CommonQL (Normative)

9.1 Introduction

Federation" is "a concept in information technology referring to the lack of central authority over software design or configuration[9]." For the purposes of this standard a catalogue federation can be defined as a loosely coupled union of catalogues that share some common characteristics regarding their content.

Figure 11 identifies some of the relationships between individual OGC Catalogues in a catalogue federation. A "star" symbolizes a Catalogue instance. Federations A, B, C, and D are made up of [2..N] Catalogue instances (only a few are pictured). A Catalogue instance may exist outside any federation, or may be a member of multiple federations.

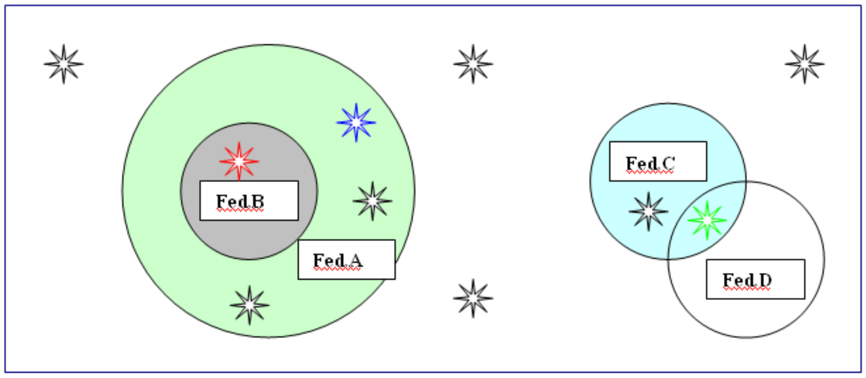


Figure 11 - The Universe of all OGC Catalogues

- A Catalogue instance may be a member of a single federation e.g., the blue instance is a member of Federation A.
- A Catalogue instance may be a member of two federations, one of which is a subset of the other e.g., the red instance is a member of Federation A and its subset Federation B.
- A Catalogue instance may be a member of two federations which share a union set of members e.g. the green instance is a member of Federations C and D.

9.2 Distributed search alternatives

In general, there exist minimally the following options for a distributed search on a catalogue federation.

9.3 Search controlled by catalogue client

The client derives the catalogue topology (the federation) behind one or more known catalogue servers by recursively discovering the "federated catalogues sections" of their

[9] Source: http://en.wikipedia.org/wiki/Federation_(disambiguation)

capability documents and collecting all the catalogues within the federation. Then the client controls the searches on the catalogues itself.

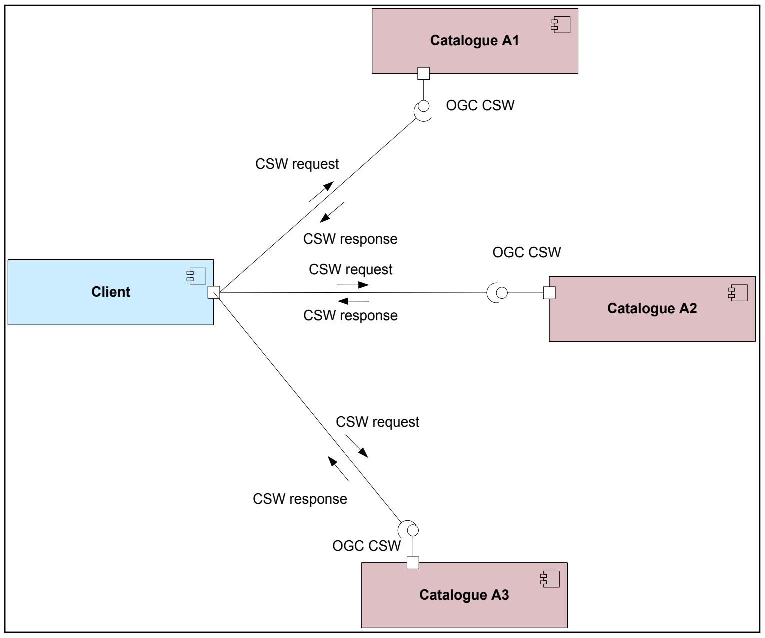


Figure 12 - Distributed search: client controlled

- Disadvantages:
    - Every client has to determine the catalogue topology from time to time.
    - The search control must be processed by every client (it is not transparent to the client).
    - Catalogues which are not directly accessible (e.g. running behind a firewall in an intranet) cannot be accessed.
- Advantages:
    - Search control can be processed by the client: so the client can decide by its own how the search is operated.
    - The response time of a single search request may be more predictable as no hidden requests to third party catalogues are involved.

9.3.1 Search controlled by catalogue server

Distributed Search in this sense allows for a Catalogue Server to accept a request from a client and distribute the request to other Catalogues within a federation. A Catalogue is acting as both: as a server and as a client (for another Catalogue – see Figure 13).

A catalogue can propagate a search request to 0, 1 or N other catalogues within the federation and the distributed catalogues can forward the request to 0, 1 or N other catalogues as well. Data returned from a Catalogue query is processed by the requesting Catalogue to return the data appropriate to the original Catalogue request. With that it becomes possible for a client to start a search from only one known location and to search as many catalogues as possible with the same filter statement. In this case, the metadata entries managed by the other catalogues become available to their own clients.

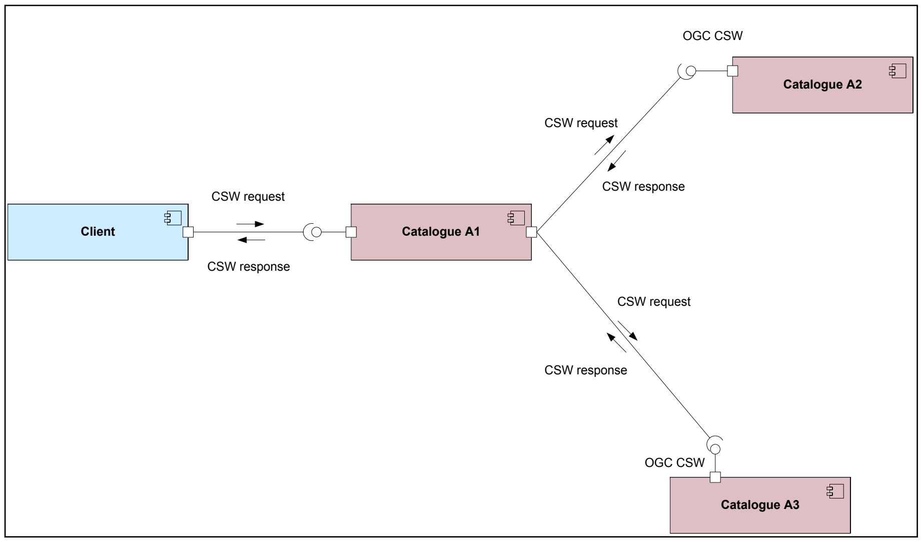


Figure 13 - Distributed search: server controlled

- Disadvantages:
    - More enhanced query request- and response-structures needed.
    - Search control must be processed by every catalogue server (which provides access to federated catalogues).
    - The response time for a single request may be less predictable as possibly hidden requests to (potentially slow) third party catalogues are involved[10].
    - For consecutive requests (belonging together) the catalogue server must handle some state information.
- Advantages:
    - The catalogue must only to know its direct "child-catalogues."
    - Catalogues behind a firewall can be accessed.
    - Search control has not to be processed by every client.

[10] To speedup very slow responding remote catalogues, a catalogue has the possibility to harvest their content from time to time (creating an entire local cache of the metadata) and perform locally all filtering on all cached results of such a catalogue.

In the following we assume a distributed search controlled by a catalogue server.

9.4 Distributed search preconditions

To enable Distributed Searching, the following preconditions must be assured.

- A multi-tier Reference Architecture as defined in Figure 14.
- A data model to define to which federated catalogue server searches can be distributed. A Catalogue may specify those federated catalogue servers in its capabilities document. For a specific catalogue server these are at a maximum all catalogues to which a query will be distributed if no restriction to specific federated catalogues is defined by the client (see below).
- A data model to define how searches are distributed to the federated catalogue servers: The discovery request and response messages define elements that allow for the retrieval and comprehension of a distributed result set. The request and response messages contain elements that allow for understanding the status of distributed searches. These elements (which were already introduced in the general interface model) are explained in the following sections.

  For a substantiation and for a better understanding how problems of distributed searches are solved by these elements and how distributed searches can be implemented, the elements used in the following sections pertain to the HTTP protocol binding.

9.5 Support of Distributed Search within Discovery Request Messages

The GetRecords message contains elements that allow the client to request certain search behaviour with respect to distribution.

The main parameter is *DistributedSearch*.

- The default query behaviour, if the *DistributedSearch* parameter is set to FALSE (or the parameter is not available), is to execute the query on the local server.
- A *DistributedSearch* parameter set to TRUE (or if the distributedSearch substructure is available in a request) indicates that the query should be distributed. In general all catalogues within the federation would be searched upon. Effectively the number and range of requested result items (defined by the entries already found and the *maxRecords*[11]- and *startPosition*-parameters) would limit the search on a few catalogues within the federation.

Distributed searches can cause specific problems that are addressed within this catalogue interface standard. Some problems result from:

[11] The maxRecords is per total and not per catalogue.

- First: the possibility that within a single distributed search the amount of approached catalogue service nodes is very large, causing long response times; and

- Second: the catalogue service node may be approached multiple times, resulting in closed loops causing the whole distributed system to potentially fail. In this case the loop causes infinite recursion – the same query is sent again and again resulting in system failure and/or timeout.

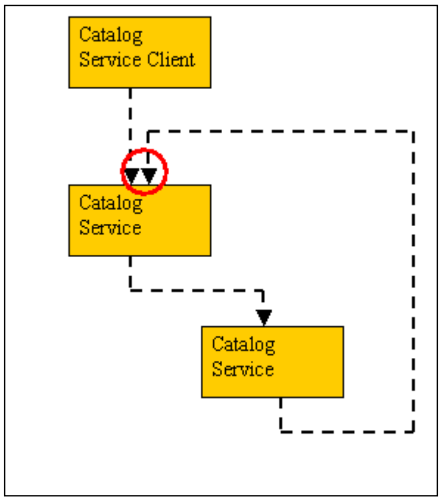


Figure 14 - Query network topology resulting in closed loops

A third potential problem is the duplication of results when a catalogue service node may be approached multiple times. This is especially hard to detect when searching different catalogues in parallel, as the parallel searches have no idea what entries are already in the parallel resultSets. Figure 15 (see also the green node contained in both Fed.C and Fed.D in Figure 11) displays a case resulting in duplicates due to the same catalogue service node being queried twice.

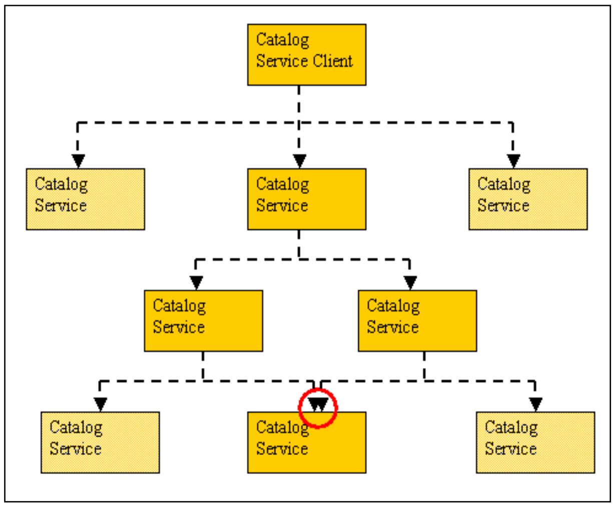


Figure 15- Query network topology resulting in duplicates

Unnecessary duplicates are a nuisance but do not normally cause the system to fail.

A method to discover if any of the problems may occur would be to control the network topology manually. Before a query is issued, the query topology is checked for duplicates or loops.

It is important to notice that the problems depicted so far can be solved by restricting the search hierarchy to two levels: a client queries a catalogue services which is allowed to cascade once. Therefore the *hopCount* parameter was introduced (see **Figure 16 - Extended search request structure**). With the *hopCount* parameter of the discovery request message a specific control is put in place to:

- prevent closed-loops of searches; and
- terminate propagation when a certain number of catalogues have been reached.

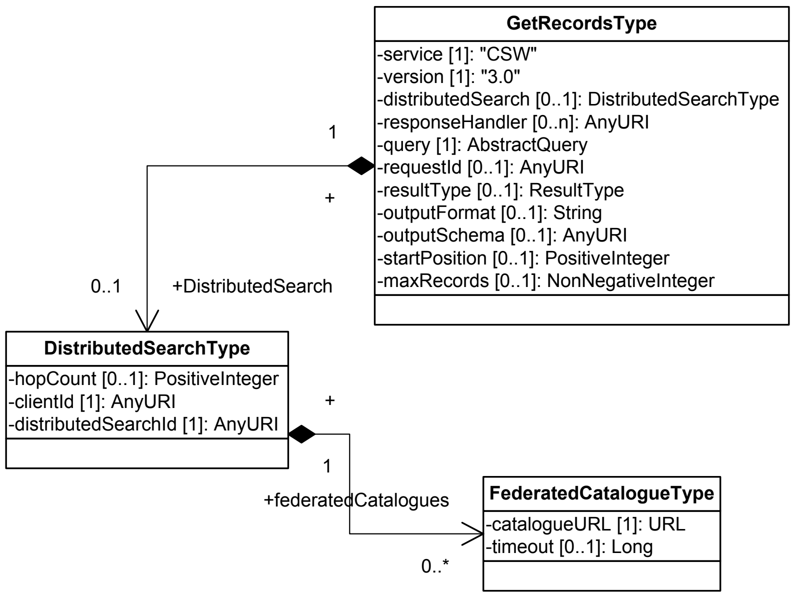

GetRecordsType
-service [1]: "CSW"
-version [1]: "3.0"
-distributedSearch [0..1]: DistributedSearchType
-responseHandler [0..n]: AnyURI
-query [1]: AbstractQuery
-requestId [0..1]: AnyURI
-resultType [0..1]: ResultType
-outputFormat [0..1]: String
-outputSchema [0..1]: AnyURI
-startPosition [0..1]: PositiveInteger
-maxRecords [0..1]: NonNegativeInteger
1
+
0..1
+DistributedSearch
DistributedSearchType
-hopCount [0..1]: PositiveInteger
-clientId [1]: AnyURI
-distributedSearchId [1]: AnyURI
+
1
+federatedCatalogues
0..*
FederatedCatalogueType
-catalogueURL [1]: URL
-timeout [0..1]: Long

Figure 16 - Extended search request structure

If the value of the *hopCount* parameter is greater than 2[12] it cannot prevent the same catalogue service node being queried twice.

To allow an automatic solution to this problem an idea would be to track the nodes already accessed: a cascading catalogue service would make sure that the list of already accessed nodes of the query gets added its own identifier (URI). But this approach does not solve the problem when searching different catalogues in parallel. A better solution

[12] Each catalogue decrements this value by one when the request is received and does not propagate the request if the hopCount=0.

would be that every catalogue server must store the unique *requestId*[13] for every request already processed. In this case a catalogue server can decide if a request (represented by its globally unique *requestId*, i.e. a UUID) was already processed and does not process the query once again. The server would return an empty result set if the request has been seen before. If the client did not include the requestId in the request, the first catalogue server accessed should generate a unique one.

Example:

Server A cascades (in parallel) the client's request (with the requestId 1013) to **Servers B** and **C. Servers B** and **C** process the request and each cascade the request to **Server D**. **Server D** will process the first request and sends a full response to the client. On the second request **Server D** detects that a request with id 1013 was already processed and sends an empty response thus preventing duplication.

Another difficulty is caused by duplicate metadata entries in a resultset that are served by different catalogue servers. But this is not really a problem because every metadata entry is uniquely addressed by the catalogue URL-prefix[14] of the Catalogues getCapabilities HTTP-GET operation from which it originates plus its identifier (which must be unique within the catalogue). So every metadata entry contained in the resultset of a distributed search can be accessed unambiguously by a subsequent getRecordById call on the catalogue which address is defined by its URL-prefix of it´s getCapabilities HTTP-GET operation and defining the Id (which is additionally included in the resultset).

Other problems are as follows:

- a client does not always want to search all catalogues which are listed within the FederatedCatalogues sections of a catalogue server's capabilities documents; and
- it is not possible to control the timeouts of searches on federated catalogues.

To restrict the number of catalogues of a federation which should be searched upon in a distributed query an optional list of those catalogues can be provided within the *federatedCatatalogues* parameter (see Fig 34) of the discovery request message. Every catalogue is represented in the list by its *url* as defined within the FederatedCatalogues constraint of the capabilities document. For every catalogue in this list an optional timeout definition (in msec) can be provided within the *timeout* parameter.

On forwarding the request to a federated catalogue the catalogue should remove its own url from the list of the targeted catalogues (federatedCatalogues).

13 So the requestId becomes mandatory in the case of a distributed search

14 As defined in OWS Common a URL prefix is defined as a string including, in order, the scheme ("http" or "https"), Internet Protocol hostname or numeric address, optional port number, path, mandatory question mark '?'

A big problem for a catalogue server within a server controlled distributed search is to assure that when the client repeats the same request (including the same targeted federated catalogues, the same filter statement and restricting the results by the same startPos and maxRecords values and under the condition that the content of the catalogues has not changed meanwhile) the resultset should be the same. Only if this is guaranteed it can be assured that a client is able to iterate block by block (defined by startPos/maxRecords) through the complete resultset.

If the catalogue servers should additionally have the possibility to provide the fastest results on every request or every sequence of requests that belong together, the protocol must provide the means to relate requests together. For example, with these means it is possible for a catalogue server to temporarily store results, re-order a predefined sequence of requested catalogues involved in a distributed search, record which results were delivered to which client within a distributed, etc. The means to enable this is the introduction of the following three conditional request parameters (which become mandatory in the case of a distributedSearch, see Figure 17:

- *distributedSearchId*: an Id which uniquely identifies a complete client initiated distributed search sequence/session;
- *clientId*: an Id which uniquely identifies the requestor; and
- distributedSearchIdTimout: defines how long (in sec) the distributedSearchId should be valid, meaning how long a server involved in distributed search should minimally store information related to the distributedSearchId.

The disadvantage of the introduced distributedSearchId is that the interface becomes more stateful.

Example:

The following example should clarify how a distributed search with these means can correctly be processed (see Figure 18). The scenario consists of a client (clientId: 'Client') and a network of 6 distributed catalogues which are involved in the distributed search. The client runs a distributed search session (distributedSearchId='123') that consists of a sequence of 4 requests. All requests include the same filter (for a new filter a new distributed search session would have to be initiated). The filter is not further considered here. The maximum hopCount is set to '4', the list of federated catalogues (CSW) that should be considered beside "A" is ["B1","C1","B2","B3","C2"].

The bigger circles in Figure 17 mark the catalogue server nodes itself. The smaller circles alongside of each catalogue node describe how much result items can be provided by the catalogue server concerning the filter statement for the request. The rectangles mark how fast (relative to the other CSWs) the responses of each catalogue are provided. The arrows mark the direction of the searches. Above and below the arrows you´ll find the requestId, startPos, maxRecs and clientId parameter values of the requests. In the boxes you can see the information that is stored locally by a catalogue server to correctly execute subsequent requests.

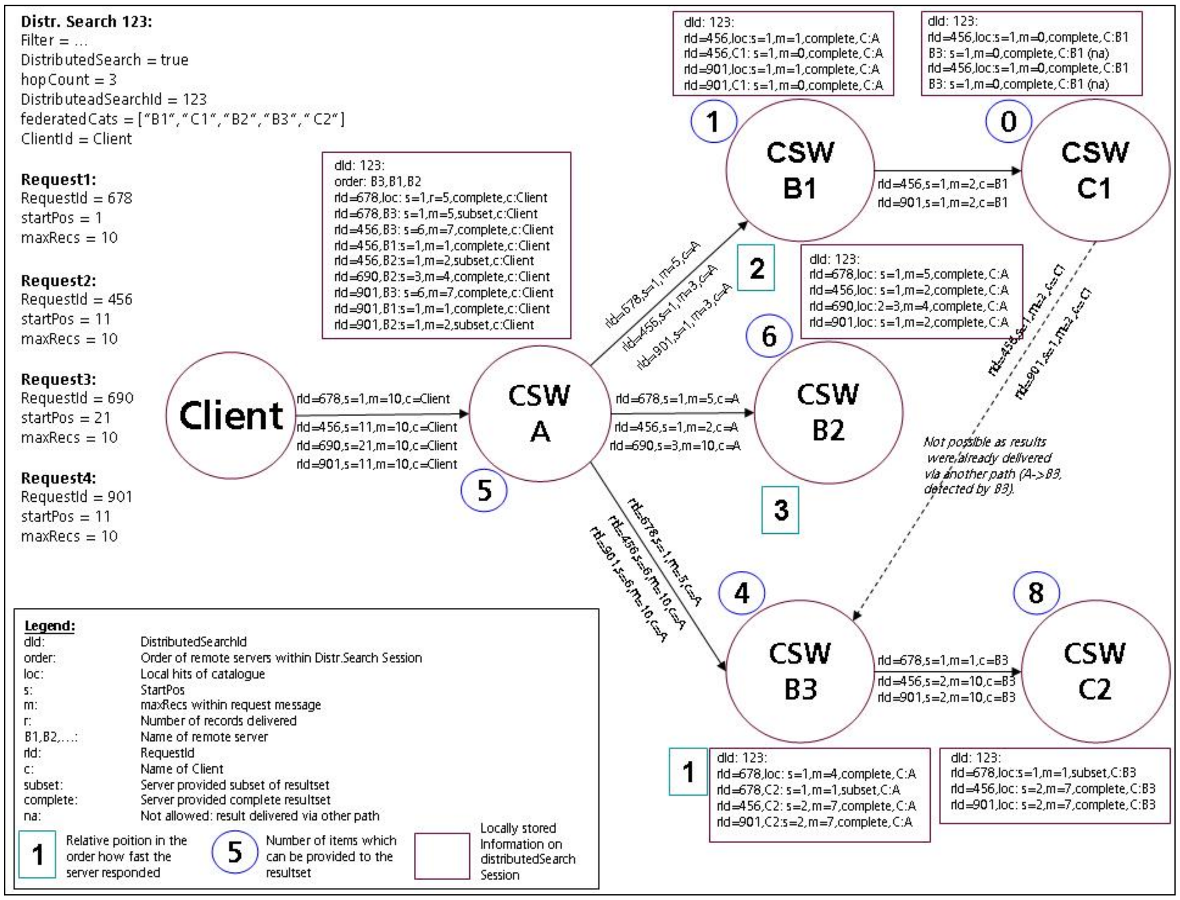

Distr. Search 123:
Filter = ...
DistributedSearch = true
hopCount = 3
DistributeadSearchId = 123
federatedCats = ["B1","C1","B2","B3"," C2"]
ClientId = Client
Request1:
RequestId = 678
startPos = 1
maxRecs = 10
Request2:
RequestId = 456
startPos = 11
maxRecs = 10
Request3:
RequestId = 690
startPos = 21
maxRecs = 10
Request4:
RequestId = 901
startPos = 11
maxRecs = 10
dId: 123:
order: B3,B1,B2
rId=678,loc: s=1,r=5,complete,c:Client
rId=678,B3: s=1,m=5,subset,c:Client
rId=456,B3: s=6,m=7,complete,c:Client
rId=456,B1:s=1,m=1,complete,c:Client
rId=456,B2:s=1,m=2,subset,c:Client
rId=690,B2:s=3,m=4,complete,c:Client
rId=901,B3: s=6,m=7,complete,c:Client
rId=901,B1:s=1,m=1,complete,c:Client
rId=901,B2:s=1,m=2,subset,c:Client
dId: 123:
rId=456,loc:s=1,m=1,complete,C:A
rId=456,C1: s=1,m=0,complete,C:A
rId=901,loc:s=1,m=1,complete,C:A
rId=901,C1: s=1,m=0,complete,C:A
dId: 123:
rId=456,loc:s=1,m=0,complete,C:B1
B3: s=1,m=0,complete,C:B1 (na)
rId=456,loc:s=1,m=0,complete,C:B1
B3: s=1,m=0,complete,C:B1 (na)
1
CSW B1
0
CSW C1
rId=456,s=1,m=2,c=B1
rId=901,s=1,m=2,c=B1
2
dId: 123:
rId=678,loc: s=1,m=5,complete,C:A
rId=456,loc: s=1,m=2,complete,C:A
rId=690,loc:2=3,m=4,complete,C:A
rId=901,loc: s=1,m=2,complete,C:A
rId=678,s=1,m=5,c=A
rId=456,s=1,m=3,c=A
rId=901,s=1,m=3,c=A
6
rId=456,s=1,m=2,c=C1
rId=901,s=1,m=2,c=C1
Client
rId=678,s=1,m=10,c=Client
rId=456,s=11,m=10,c=Client
rId=690,s=21,m=10,c=Client
rId=901,s=11,m=10,c=Client
CSW A
5
rId=678,s=1,m=5,c=A
rId=456,s=1,m=2,c=A
rId=690,s=3,m=10,c=A
CSW B2
3
Not possible as results were already delivered via another path (A->B3, detected by B3).
rId=678,s=1,m=5,c=A
rId=456,s=6,m=10,c=A
rId=901,s=6,m=10,c=A
4
CSW B3
8
CSW C2
rId=678,s=1,m=1,c=B3
rId=456,s=2,m=10,c=B3
rId=901,s=2,m=10,c=B3
1
dId: 123:
rId=678,loc: s=1,m=4,complete,C:A
rId=678,C2: s=1,m=1,subset,C:A
rId=456,C2: s=2,m=7,complete,C:A
rId=901,C2:s=2,m=7,complete,C:A
dId: 123:
rId=678,loc:s=1,m=1,subset,C:B3
rId=456,loc: s=2,m=7,complete,C:B3
rId=901,loc: s=2,m=7,complete,C:B3
Legend:
dId: DistributedSearchId
order: Order of remote servers within Distr.Search Session
loc: Local hits of catalogue
s: StartPos
m: maxRecs within request message
r: Number of records delivered
B1,B2,...: Name of remote server
rId: RequestId
c: Name of Client
subset: Server provided subset of resultset
complete: Server provided complete resultset
na: Not allowed: result delivered via other path
1 Relative poition in the order how fast the server responded
5 Number of items which can be provided to the resultset
Locally stored Infomation on distributedSearch Session

Figure 17 - Distributed search example (requests, node statuses)

The client sends 'request1' of distributedSearch '123' with startPos='1' and maxRecs='10' to CSW **A**. Because CSW **A** can only contribute 5 records to the resultset it starts a search on its associated CSWs. Usually **A** would deliver the results of its associated catalogues in a predefined order (**B1, B2, B3**). To be able to deliver the fastest results first CSW **A** starts the searches on **B1, B2** and **B3** in parallel threads and checks which results are first available. If e.g. **B3** is the fastest responding catalogue, CSW **A** integrates the results of **B3** first (in case of request1 5 items: 4 from **B3**, 1 from **C2**). Therefore request1 is already satisfied and **A** returns the 10 result items. For request1 **B3** provides only 4 local items and started a distributed search on its associated catalogue **C2** with startPos='1' and maxRecs='1'. CSW **C2** delivered 1 item to **B3** which is a subset of **C2**'s resultset. CSW **C2** and CSW **B3** stored the information which result items were delivered to which client.

To allow the client to iterate through the complete resultset **A** must memorize e.g. the order of the distributed catalogues used within distributedSearch ‘123’ (order now: **B3, B1, B2**) and/or memorize which result block (startPos, maxRecs) originates from which distributed catalogue. It may also optionally cache results returned from the distributed catalogues.

For request2 CSW **A** can derive (by the distributedSearchId) that **B3** already delivered its result items 1-5 (resultset status: subset) and starts now requesting CSW **B3** for the results with startPos=’6’ and maxRecs=’10’. CSW **B3** checks that it already delivered its complete items (1-4) as well as the first item of **C2** to **A**. As **B3** knows that **C2** did not deliver its full results it tries to search with startPos=’2’ and maxRecs=’10’. But CSW **C2** could only deliver 7 items which are now bookmarked by **B3** and (later) returned to CSW **A**. **A** checks that 7 items (but not 10) where delivered, memorizes this and starts a search with startPos=’1’ and maxRecs=’3’ (3 items are still missed for request2) on **B1** (which is number 2 in the re-ordered list of associated catalogues of CSW **A**). CSW **B1** could only provide 1 local item and so starts a distributed search on its associated catalogue **C1** with startPos=’1’ and maxRecs=’2’. CSW **C1** could not provide any item and starts a distributed search with startPos=’1’ and maxRecs=’2’ on its associated catalogue **B3**. Now comes the newly introduced parameter clientId into play: CSW **B3** detects by the clientId that within distributedSearch session ‘123’ the results 1-2 where already delivered to Client CSW **A** and denies the response of any items. Because in this case the whole subtree below CSW **B1** could only provide 2 items CSW **A** must start a distributed search with startPos=’1’ and maxRecs=’2’ on **B2** which delivers the two items (resultset of CSW **B2**: subset) and so on….

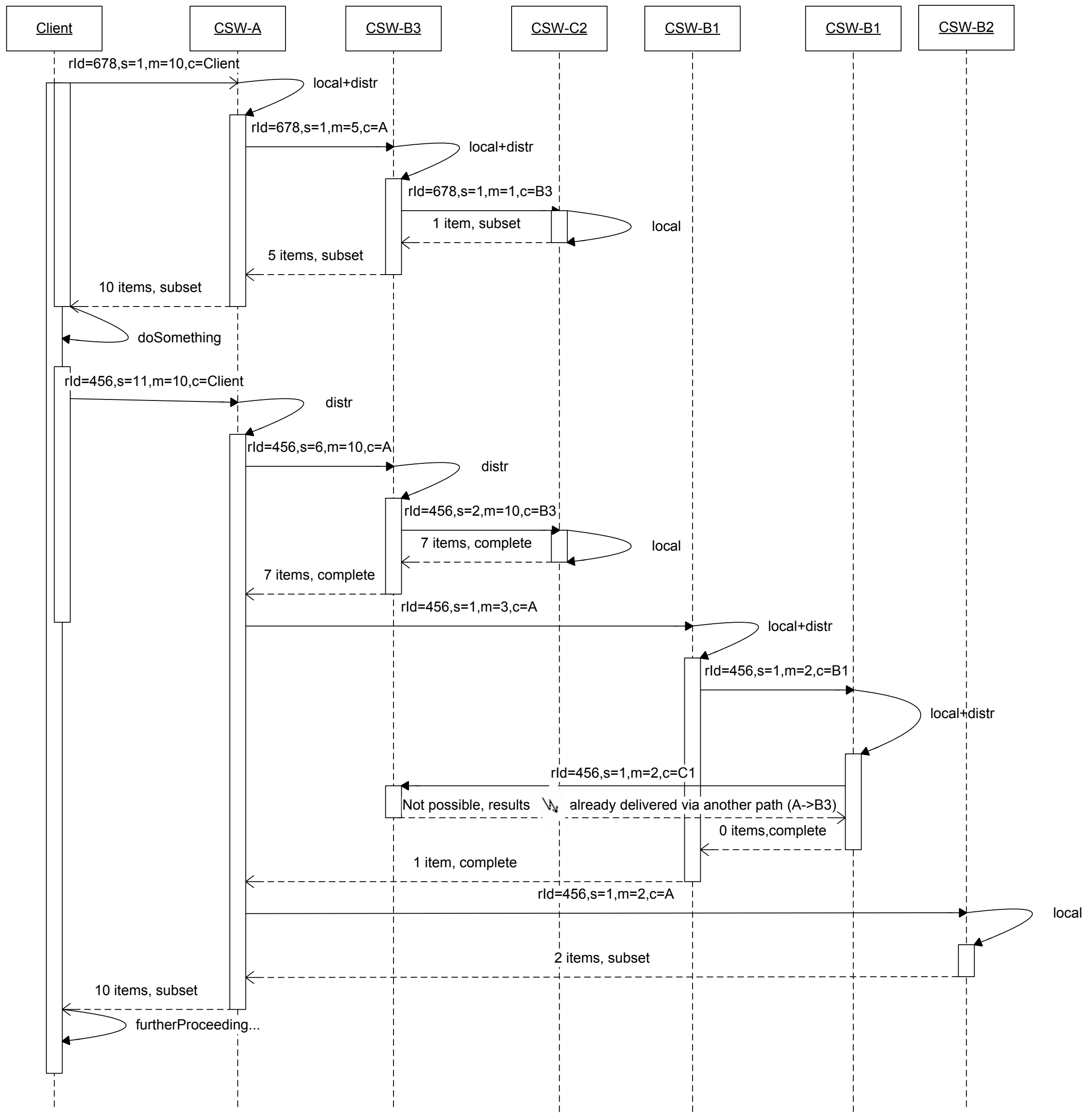


Figure 18 - Distributed Search Example (sequence diagram)

The scenario described in this example is only one possible implementation. Others may cache parts of the results, do not request their associated catalogues in a steady order, harvest the whole content of their associated catalogues, and so forth.

Another point to consider is that in the scenario explained so far the clients wait as long as all requested (available) items are delivered. This makes the client implementation easier. It does also imply that the **status** of every response is either *complete* (no more items are available) or *subset* (more items are available). To further speed up distributed searches the catalogue servers may deliver partial resultsets while further trying to acquire the outstanding result items. In an appropriate scenario CSW **A** could deliver the first 5 items and define the status of the response message as *processing* meaning that for faster response the requested number of items was not fully returned although more items are expected in the requested startPosition/maxRecords range. In this case the client can already display the first 5 items (although requested 10) and the server can continue with processing the distributed searches and temporary caching those which are already acquired. In the next step the client must adjust its request: next n records starting at position 6.... Consequence of this is that clients must generally check the response status and if necessary adjust their startPos and maxRecs request parameters.

For further speeding up the processing it is strongly recommended to not query for hitCount in the case of a distributed search as this would start a full distributed search slowing down the search speed.

9.6 Support of distributed search within discovery response messages

The results of every catalogue involved in a distributed search result are grouped within the *federatedSearchResult* element (which is of type *FederatedSearchResultType*) of the *searchResults*. Every *federatedSearchResult* element includes the *catalogueURL* (the URL-prefix[15] of the getCapabilities HTTP-GET operation of the catalogue). This URL is also required for a subsequent getRecordByID request to be sent by the client.

Further the *federatedSearchResult* element again includes an element of the type *SearchResultsType* (see Fig. xx) so that trees of results can be described. Important information of the SearchResultsType with regard to federated search are:

- The result items (*resultEntry*);
- information how many results are delivered by the catalogue(*numberOfRecordsReturned*);
- information how many results are matched within the catalogue regarding the request (*numberOfRecordsMatched)*; and
- runtime information of the search within the federated catalogue (*elapsedTime*).

If a federated catalogue has thrown an exception an entry of type *FederatedExceptionType* is included instead of type *FederatedSearchResultType. FederatedExceptionType* includes the URL- prefix of the getCapabilities HTTP-GET operation of the catalogue (*catalogueURL*) as well as one or more elements of type ows:ExceptionReport.

[15] As defined in OWS Common a URL prefix is defined as a string including, in order, the scheme ("http" or "https"), Internet Protocol hostname or numeric address, optional port number, path, mandatory question mark '?'

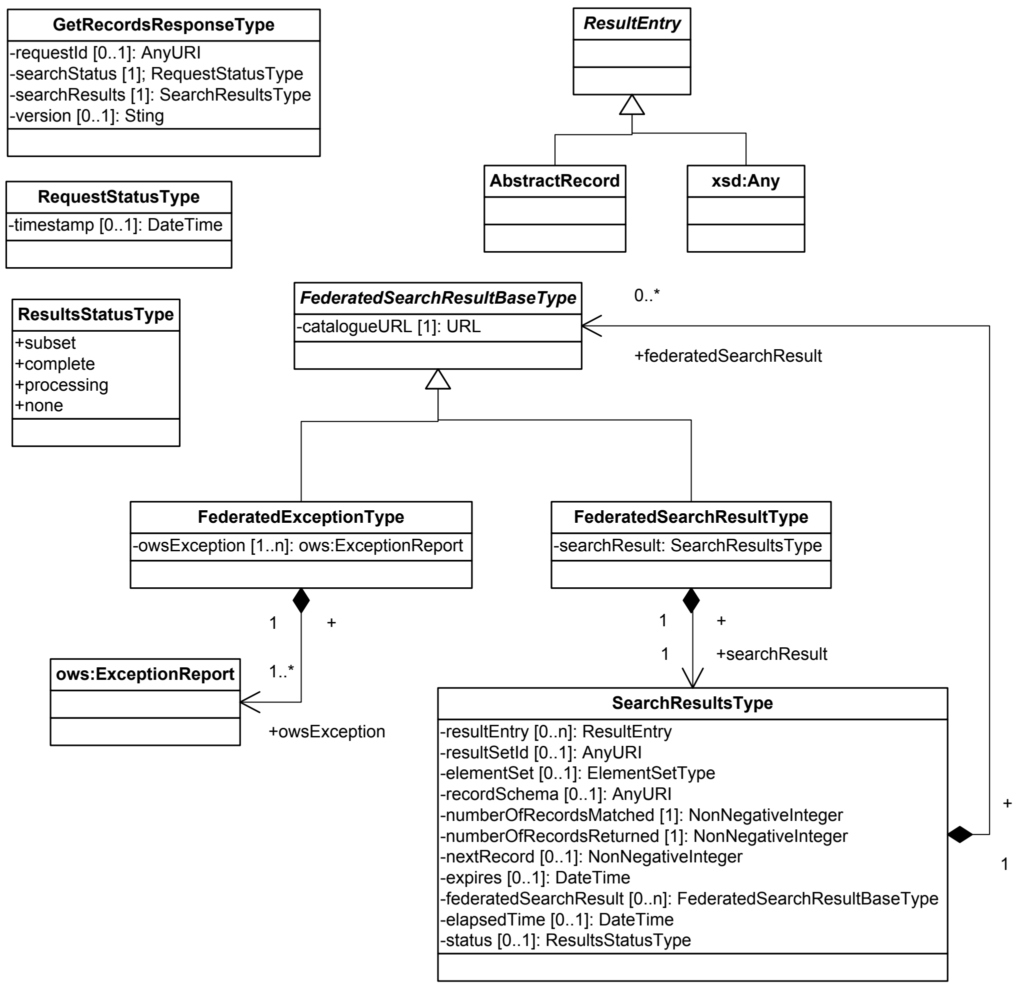

GetRecordsResponseType
-requestId [0..1]: AnyURI
-searchStatus [1]; RequestStatusType
-searchResults [1]: SearchResultsType
-version [0..1]: Sting
RequestStatusType
-timestamp [0..1]: DateTime
ResultsStatusType
+subset
+complete
+processing
+none
ResultEntry
AbstractRecord
xsd:Any
FederatedSearchResultBaseType
-catalogueURL [1]: URL
0..*
+federatedSearchResult
FederatedExceptionType
-owsException [1..n]: ows:ExceptionReport
FederatedSearchResultType
-searchResult: SearchResultsType
1
+
1..*
ows:ExceptionReport
+owsException
1
+
1
+searchResult
SearchResultsType
-resultEntry [0..n]: ResultEntry
-resultSetId [0..1]: AnyURI
-elementSet [0..1]: ElementSetType
-recordSchema [0..1]: AnyURI
-numberOfRecordsMatched [1]: NonNegativeInteger
-numberOfRecordsReturned [1]: NonNegativeInteger
-nextRecord [0..1]: NonNegativeInteger
-expires [0..1]: DateTime
-federatedSearchResult [0..n]: FederatedSearchResultBaseType
-elapsedTime [0..1]: DateTime
-status [0..1]: ResultsStatusType
+
1

Figure 19 - GetRecordsResponse supporting distributed search results

9.7 Distributed search with common information model

Distributed searches are not only possible on OGC CSW base catalogues but also on catalogues implementing an OGC CSW profile (without having to know anything about the profile). This is achieved by using the OGC CSW common profile information model which does not only include a list of core queryable properties but also the common record response schema (a subset of Dublin Core metadata elements). All OGC CSW compliant catalogues must support a mapping of the core queryables to their information model and vice versa a mapping of their information model to the common record response schema.

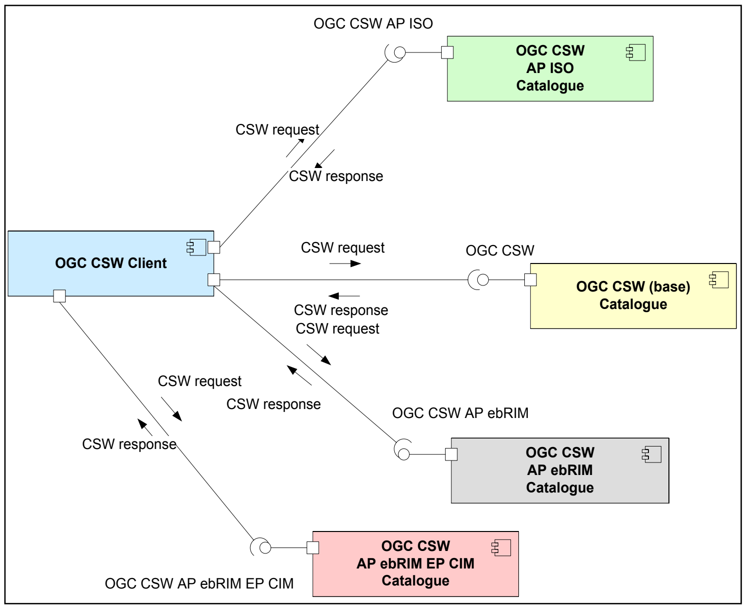


Figure 20 - Distributed search with common information model

Thus an OGC CSW client should be able to query any OGC CSW catalogue, regardless of the underlying information model, using the elements defined in the common record schema and understand the response (common record schema). With this model it is possible for an OGC CSW Client to query an OGC CSW AP ISO Catalogue.

10. Annex B - BNF Definition of OGC CommonQL (Normative)

```
<SQL terminal character> ::= <SQL language character>
<SQL language character> ::= <simple Latin letter>
                           | <digit>
                           | <SQL special character>
<simple Latin letter> ::= <simple Latin upper case letter>
                          | <simple Latin lower case letter>
<simple Latin upper case letter> ::=
     A | B | C | D | E | F | G | H | I | J | K | L | M | N | O
   | P | Q | R | S | T | U | V | W | X | Y | Z
<simple Latin lower case letter> ::=
     a | b | c | d | e | f | g | h | i | j | k | l | m | n | o
   | p | q | r | s | t | u | v | w | x | y | z
<digit> ::=
     0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
<SQL special character> ::= <space>
                          | <double quote>
                          | <percent>
                          | <ampersand>
                          | <quote>
                          | <left paren>
                          | <right paren>
                          | <asterisk>
                          | <plus sign>
                          | <comma>
                          | <minus sign>
                          | <period>
                          | <solidus>
                          | <colon>
                          | <semicolon>
                          | <less than operator>
                          | <equals operator>
                          | <greater than operator>
                          | <question mark>
                          | <left bracket>
                          | <right bracket>
                          | <circumflex>
                          | <underscore>
                          | <vertical bar>
                          | <left brace>
                          | <right brace>
<space> ::= /*space character in character set in use
              In ASCII it would be 40*/
<double quote> ::= "
<percent> ::= %
<ampersand> ::= &
<quote> ::= '
<left paren> ::= (
<right paren> ::= )
```

```
<asterisk> ::= *
<plus sign> ::= +
<comma> ::= ,
<minus sign> ::= -
<period> ::= .
<solidus> ::= /
<colon> ::= :
<semicolon> ::= ;
<less than operator> ::= <
<equals operator> ::= =
<greater than operator> ::= >
<question mark> ::= ?
<left bracket> ::= [
<right bracket> ::= ]
<circumflex> ::= ^
<underscore> ::= _
<vertical bar> ::= |
<left brace> ::={
<right brace> ::=}
<separator> ::= { <comment> | <space> | <newline> }
/* The next section of the BNF defines the tokens available to the
   language. I have deleted the concepts of bit string, hex string and
national character string literal. Keywords have been added to support
the geo literals. */
<token> ::= <nondelimiter token>
         | <delimiter token>
<nondelimiter token> ::= <regular identifier>
                       | <key word>
                       | <unsigned numeric literal>
<regular identifier> ::= <identifier body>
Proposed change:
<regular identifier> ::= <identifier body>
                     | <double quote> {unicode_character} <double
quote>[16]

<identifier body> ::=
<identifier start> [ { <underscore> | <identifier part> } ]
<identifier start> ::= <simple latin letter>
<identifier part> ::= <identifier start>
                   | <digit>
<key word> ::= <reserved word>
```

<reserved word > ::=
```
         AND | NOT |
         POINT | LINESTRING | POLYGON |
         MULTIPOINT | MULTILINESTRING | MULTIPOLYGON |
         EMPTY |
         CURRENT_DATE| CURRENT_TIME | CURRENT_TIMESTAMP|
         FALSE| TRUE | UNKNOWN
         | LIKE| NULL
```

[16] This change is inspired in SQL in order to accept local character sets as property name. Moreover, it resolves the clash problem between cql keywords and property names.

```
<unsigned numeric literal> ::= <exact numeric literal>
                             | <approximate numeric literal>
<exact numeric literal> ::= <unsigned integer>
                           [<period>[<unsigned integer>]]
                           |<period> <unsigned integer>
<unsigned integer> ::= {<digit>}
<approximate numeric literal> ::= <mantissa> E <exponent>
<mantissa> ::= <exact numeric literal>
<exponent> ::= <signed integer>
<signed integer> ::= [ <sign> ] <unsigned integer>
<sign> ::= <plus sign> | <minus sign>
<character string literal> ::=
   <quote> [ {<character representation>} ] <quote>
<character representation> ::= <nonquote character> | <quote symbol>
<quote symbol> ::= <quote><quote>
/*End of non delimiter tokens*/
/* I have limited the delimiter tokens by eliminating, interval strings
and delimited identifiers BNF and simplifying the legal character set
to the characters to a single set so no identification of character set
would be needed decision. */
<delimiter token> ::= <character string literal>
                    | <SQL special character>
                    | <not equals operator>
                    | <greater than or equals operator>
                    | <less than or equals operator>
                    | <concatenation operator>
                    | <double greater than operator>
                    | <right arrow>
                    | <left bracket>
                    | <right bracket>


<character string literal> ::=
   <quote> [ {<character representation>} ] <quote>
<character representation> ::= <nonquote character> | <quote symbol>
<quote symbol> ::= <quote><quote>
<not equals operator> ::= <>
<greater than or equals operator> ::= >=
<less than or equals operator> ::= <=
/*The following section is intended to give context for identifier and
namespaces.  It assumes that the default namespace is specified in the
query request and does not allow any overrides of the namepace */
<identifier> ::=
    <identifier start [ { <colon> | <identifier part> } ]
<identifier start> ::= <simple Latin letter>
<identifier part> ::= <simple Latin letter> | <digit>
<attribute name> ::= <simple attribute name> | <compound attribute
name>
<simple attribute name> ::= <identifier>
<compound attribute name> ::= <identifier><period>
                              [{<identifier><period>}…]
                              <simple attribute name>

/*The rest of the BNF is the real BNF for the query capabilities.*/
```

```
<search condition> ::= <boolean value expression>
<boolean value expression> ::= <boolean term>
               | <boolean value expression> OR <boolean term>
<boolean term> ::= <boolean factor>
                 | <boolean term> AND <boolean factor>
<boolean factor> ::= [ NOT ] <boolean primary>
<boolean primary> ::= <predicate> |
                      <routine invocation> |
                      <left paren> <search condition> <right paren>
<predicate> ::= <comparison predicate>
              | <text predicate>
              | <null predicate>
              | <temporal predicate>
              | <classification predicate>
              | <existence_predicate>

/* This set of productions enables loose or tight queries. For example
the predicate "cloudcover EXISTS" evaluates as true for all record
instances where the attribute cloudcover is a member of the record
schema. Similarly, the predicate "cloudcover DOESNOTEXIST" evaluates as
true for all record instances where the attribute cloudcover is not a
member of the record schema.*/

<existence_predicate> := <attribute_name> EXISTS
                       | <attribute_name> DOES-NOT-EXIST

<comparison predicate> ::= <attribute name> <comp op> <literal>
<text predicate> ::= <attribute name> [ NOT ] LIKE <character pattern>
<null predicate> ::= <attribute name> IS [ NOT ] NULL
<character pattern> ::= <character string literal>
      /* In a character pattern the character percent is used as a
wildcard to represent an arbitrary string. This allows LIKE to
implement the effect of many characters matching operations, such as:
contains, begins with, ends with, not contains, not begins with, not
ends with, and so forth. For example:
         attribute like '%contains_this%'
         attribute like '%begins_with_this%'
         attribute like '%ends_with_this'
         attribute like '%d_ve' will match 'dave' or 'dove'
         attribute not like '%will_not_contain_this%'
         attribute not like '%will_not_begin_with_this%'
         attribute not like '%will_not_end_with_this'   */
<comp op> ::= <equals operator>
            | <not equals operator>
            | <less than operator>
            | <greater than operator>
            | <less than or equals operator>
            | <greater than or equals operator>
<literal> ::= <signed numeric literal>
            | <general literal>
<signed numeric literal> ::= [<sign>] <unsigned numeric literal>
<general literal> ::= <character string literal>
                    | <temporal predicate>
                    | <boolean literal>
                    | <geography literal
```

```
<temporal predicate> =:: <attribute name> <date predicate>
<date predicate> =:: <single data predicate> | <range date predicate>

<single date predicate> =:: <time op> <date-time>
<range date predicate> =:: <range op> <date-time> <date-time>

<time op> =:: "BEFORE" | "EQUALS" | "AFTER"
<range op> =:: "OVERLAPS" | "CONTAINS" | "WITHIN" | "DURING" | "BEFORE"
| "AFTER"

<boolean literal> ::= TRUE
                    | FALSE
                    | UNKNOWN
<routine invocation> ::= | <geoop name><georoutine argument list>
                         | <relgeoop name><relgeoop argument list>
                         | <routine name><argument list>
<routine name> ::= < attribute name>
<geoop name> ::= EQUALS | DISJOINT | INTERSECTS | TOUCHES | CROSSES
               | WITHIN | CONTAINS | OVERLAPS | RELATE
<relgeoop name> ::= DWITHIN | BEYOND
<argument list> ::=
   <left paren> [<positional arguments>]  <right paren>
<positional arguments> ::=
   <argument> [ { <comma> <argument> } ]
<argument> ::= <literal> | <attribute name>
<georoutine argument list> ::=
<left paren><attribute name><comma><geometry literal><right paren>
<relgeoop argument list> ::= <left paren>
   <attribute name><comma><geometry literal><comma><tolerance>
   <right paren>
<tolerance> ::= <unsigned numeric literal><comma><distance units>
<distance units> ::= = “feet” | “meters” | “statute miles” |
                       “nautical miles” | “kilometers”
/*this set of units is just an example. The real list of distance unit
must be developed*/
<geometry literal> := <Point Tagged Text>
                    | <LineString Tagged Text>
                    | <Polygon Tagged Text>
                    | <MultiPoint Tagged Text>
                    | <MultiLineString Tagged Text>
                    | <MultiPolygon Tagged Text>
                    | <GeometryCollection Tagged Text>
                    | <Envelope Tagged Text>
<Point Tagged Text> := POINT <Point Text>
<LineString Tagged Text> := LINESTRING <LineString Text>
<Polygon Tagged Text> := POLYGON <Polygon Text>
<MultiPoint Tagged Text> := MULTIPOINT <Multipoint Text>
<MultiLineString Tagged Text> := MULTILINESTRING <MultiLineString Text>
<MultiPolygon Tagged Text> := MULTIPOLYGON <MultiPolygon Text>
<GeometryCollection Tagged Text> :=
   GEOMETRYCOLLECTION <GeometryCollection Text>
<Point Text> := EMPTY | <left paren> <Point> <right paren>
<Point> := <x><space><y>
<x> := numeric literal
```

```
<y> := numeric literal
<LineString Text> := EMPTY
                   | <left paren>
                     <Point>
                     {<comma><Point >}
                     <right paren>
<Polygon Text> := EMPTY
                | <left paren>
                  <LineString Text>
                  {<comma><LineString Text>}
                  <right paren>
<Multipoint Text> := EMPTY
                   | <left paren>
                     <Point Text>
                     {<comma><Point Text >}
                     <right paren>
<MultiLineString Text> := EMPTY
                        | <left paren>
                          <LineString Text>
                          {<comma><LineString Text>}
                          <right paren>
<MultiPolygon Text> := EMPTY
                     | <left paren>
                       <Polygon Text>
                       {<comma><Polygon Text>}
                       <right paren>
<GeometryCollection Text> := EMPTY
                           | <left paren>
                             <geometry literal>
                             {<comma><geometry literal>}
                             <right paren>
<Envelope Tagged Text> ::= ENVELOPE <Envelope Text>
<Envelope Text> := EMPTY
                | <left paren>
                  <WestBoundLongitude><comma>
                  <EastBoundLongitude><comma>
                  <NorthBoundLatitude><comma>
                  <SouthBoundLatitude>
                  <right paren>
<WestBoundLongitude> := <signed numeric literal>
<EastBoundLongitude> := <signed numeric literal>
<NorthBoundLatitude> := <signed numeric literal>
<SouthBoundLatitude> := <signed numeric literal>

<date-time>   ::= <full-date> | <full-date> "T" <UTC-time>
<full_date>   ::= <date-year> "-" <date-month> "-" <date-day>
<date-year>   ::= <digit><digit><digit><digit>
<date-month>  ::= <digit><digit>
<date-day>    ::= <digit><digit>

<UTC-time>  ::=
      <time-hour>":"<time-minute>":"<time-second> [<time-zone-offset>]
<time-zone-offset> ::= "Z" | <sign><time-hour>

<time-hour>   ::= <digit><digit>
```

```
<time-minute> ::= <digit><digit>
<time-second> ::= <digit><digit>[.{<digit>}]

<duration>    ::= "P" (<dur-date> | <dur-time>)
<dur-date>    ::= <dur-day> | <dur-month> | <dur-year> [<dur-time>]
<dur-day>     ::= {<digit>} "D"
<dur-month>   ::= {<digit>} "M" [<dur-day>]
<dur-year>    ::= {<digit>} "Y" [<dur-month>]

<dur-time>    ::= "T" (<dir-hour> | <dur-minute> | <dur-second>)
<dur-hour>    ::= {<digit>} "H" [<dur-minute>]
<dur-minute>  ::= {<digit>} "M" [<dur-second>]
<dur-second>  ::= {<digit>} "S"

<period>      ::= <date-time>"/"<date-time>

<period >      ::= <date-time> "/" <date-time>  /*between date-times*/
                | "/" <date-time>               /*before date-time*/
                | <date-time> "/"               /*after date-time*/
                | <date-time> "/" <duration>
                | <duration> "/" <date-time>
```

11. Annex C - Revision History

11.1 Revision History: V2.0 and Earlier

The revision history for 2.0 and earlier versions of the OGC Catalogue Services standard is:

Date	Release	Editor	Paragraph modified	Description
12Aug1999	1.0	Nebert	N/A	Original Specification entitled "Catalogue Interface Implementation Specification" OGC Document 00-034
28Mar2001	1.1	Nebert	Made fine-grain CORBA and OLE/COM Annexes to Informative, fixed coarse-grain CORBA IDL	Document only made available to OGC membership pending passage of Version 2.0. (OGC Document 01-040)
11Nov2002	1.1.1	Nebert, Katz,	State diagram changes, renamed specification and changed WWW Profile to Z39.50 Profile, added introductory words as required for new format	Document primarily reflects conversion to newer OGC/ISO document format
22Dec2003	2.0.0	Nebert	All	Reorganised and largely rewrote document.
6Mar2004	2.0.0	Nebert	Clauses 6,7, 9,10,11	Edited CORBA, Z39.50, and HTTP to reflect changes in General Model, other minor revisions to document
29Mar2004	2.0.0	Whiteside	All	
14Apr2004	2.0.0	Whiteside	All	
11May2004	2.0.0	Nebert	Merge updates on Clauses 1-5, 6-8, 9, 10, and 11 from separate editors	Responded to all RWG review comments.

11.2 Revision history: v3.0

The revision history for this general model part of the OGC Catalogue Services specification is:

Date yyyy-mm-dd	Release	Editor	Paragraph modified	Description
2007-05-25	3.0	Whiteside	All	First draft, majority copied from 2.0.2
2008-10-28	3.0	Voges	7.2.4.2, Fig. 5, Table 9,10,11,12, Annex A	Improvement of definition what distributed search is, which

Date yyyy-mm-dd	Release	Editor	Paragraph modified	Description
				alternatives for distributed search are available and what problems can arise. Functional extensions to the discovery request and response messages (which define elements that allow for the retrieval and comprehension of a distributed result set).
2008-10-31	3.0	Voges	Foreword, Introduction, 1 Scope, 7.1, 8.1, 8.3.2, Table 50	Removed any refernces to the Corba Protocol Binding
2010-07-07	3.0	Voges	6.2.2	Fixed CQL BNF provided by Mauricio Pazos: <datetime literal> replaced by <temporal predicate>
2010-11-18	3.0	Voges	6.2.2	Included CQL BNF proposal provided by Mauricio Pazos (Axios)
2011-01-24	3.0	Voges	6.2.2	Included CQL BNF improvements provided by Mauricio Pazos (Axios)
2012-05-01	3.0	Nebert	7.2.1 and tables	Removed reference to stateful and ordering requirements (Brokered Access)
2013-01-30	3.0	Nebert	Throughout	Updated URNs to requirements, General editorial fixes.
2013-07-29	3.0	Voges	Throughout	Review/comment/update
2013-07-30	3.0	Westcott	Throughout	Applied current OGC template; editorial review
2014-02-05	3.0	All	Throughout	Resolution of Outstanding issues; largely editorial
2014-10-28	3.0	All	Throughout	Resolution of Outstanding issues; largely editorial
2014-11-19	3.0	Bigagli	9, 8.6.3, others minor	Rearrangement of some paragraphs. Review/comment/update
2014-12-02	3.0	All	Throughout	URI check, minor editorial
2015-09-10	3.0	All	Voges	Minor editorial, reference to ISO19115-1

11.3 Summary of Changes

Title	Details

Document Structure	CS 3.0 is in 1+2 parts (Core+Extensions): o Part 1: General Model (Core) o Part 2: HTTP Protocol Binding: CSW o Part 3 (referenced): OpenSearch OGC 10-032
Aligned with ISO multi-part document "Part" approach	Included Core and Extension model: 'boxed' requirements mapped to conformance classes
	Set of Conformance Classes and Requirements
	Added ATS
Protocol Bindings	Included OpenSearch with Geo and Time extensions as the "baseline" query operation for all profiles
	Dropped CORBA and z39.50/SRU bindings No more stateful z39.50 CS Interface in General Model (GM): SRU provided as a discussion paper (OGC 12-082)
	CS 3.0 now only supports HTTP bindings
Alignment with other OGC Specs	Aligned with OWS Common 2.0
	Aligned CSW with Filter 2.0: Filter 2.0 can be used with ANY version of GML.
New queryables	New Core Queryables: temporal extent begin and end as core queryables
	Added TemporalExtent as queryable (Type gml:TimePeriod) and returnable (new csw30:TemporalExtentType)
DistributedSearch improved	
CQL BNF updated	
General request structure improvments	Fixed a number of inconsistencies between XML and KVP encodings
	Clarifications on relation: HTTP message headers and request parameters

	Attribute „deleted“ of AbstractRecord defines if metadata item was deleted (time of deletion can be expressed by including an element like dct:modified)
Dropped / Added Operations	DescribeRecord operation dropped
	UnHarvest operation added to the HTTP binding: complementary operation for Harvest
	Removal of OrderOperation
Changes on Operations	Fixed inconsistencies between XML and KVP encodings and clarified relation between HTTP message headers and request parameters
	GetRecords: csw:Record left as mandatory common information model across catalogues BUT ATOM now a mandatory response-only format, improved KVP-encoding, optimized request structures, …
	Get-Capabilities Document: Filter_Capabilities in Capabilities now optional, defined default sorting, …
	GetRecordByID: response (for HTTP/XML/POST and HTTP/KVP/GET) now raw response in its original format, GetRecordByIdResponse only available for SOAP, only 1 id as input, added “outputSchema" parameter
	GetDomain: revised the XML schema types, counterchecked the capabilities with requirements of modern search interfaces, like Autosuggest, DidYouMean, able to interogate any request parameter and info model component, added filter on parameters so one can restrict results to values that satisfy the filter, added ResultType parameter to return all 'possible' (enumeration) versus 'available', response container on 'available' will return number of records with that value.
	Transaction: Harmonized Update-, Insert-, DeleteType: typeName attribute, constraint in the capabilities providing list of URI’s identifying that schemas upon which the Transaction operation may be applied.

	Harvest: Support added for HTTP/POST with attachments for Harvesting of resources that cannot be referenced via URL.
XML Schemas updated	Refactored and updated XML schemas
New WSDL version	Updated the WSDL document

11.4 Changes in Detail

11.4.1 General Model

- No more stateful z39.50 CS Interface in General Model (GM):
    - removed sessionInfo parameters in UML models and tables
    - removed Z39.50 references, including former section 7.2.5 (Session class), 7.3 (Dynamic Model), and Annex C
    - Removed details of Order Operation
- New Core Queryables: temporal extent begin and end as core queryables
- CQL BNF updated
- DescribeRecord operation dropped

11.4.2 Protocol Bindings

- CS 3.0 now only support HTTP bindings
    - Search/Discovery: CSW, OpenSearch
    - Management/Transaction:CSW
- OpenSearch with Geo-/Temporal Extension OGC 10-032 must be supported regardless of profile or even without a profile)
    - OpenSearch (OS) is a HTTP GET Request Interface with key-value parameters to constrain the search
    - OS is flexible, results can be returned as HTML, Atom, XML/RDF, KML, WKT, JSON….
    - Geo-/Temp-Extension specifies parameters to spatially and temporally constrain search results

- Corba, z39.50 (stateful) and SRU (stateless) dropped
    - CORBA protocol binding was very rarely implemented
    - does not correspond with a commonly used protocol for crossing internet domains
    - was not conformant to the latest general interface model.
- Z39.50 (stateful) sections in GM dropped, including:
    - Session class and Dynamic Model
    - parts related to BrokeredAccess class (order!)
    - SRU (HTTP-based Search/Retrieve via URL) is the successor of Z39.50 (same semantics)
        - not included as another HTTP binding
        - was cutted off and provided as a discussion paper (OGC 12-082) to the OGC portal so that the Met/Ocean DWG and others can use this document as starting point for internal discussion and possibly for developing a future CS 3.0 SRU profile.

11.5 Document Structure

- CS 3.0 is in 1+2 parts (Core+Extensions):
    - Part 1: General Model (Core)
    - Part 2: HTTP Protocol Binding: CSW
        - HTTP/GET/KVP (mandatory)
        - Other CSW HTTP bindings optional: POST/XML / + SOAP
    - Part 3 (referenced): OpenSearch OGC 10-032

11.6 Conformance Classes

- Added set of conformance classes with requirements
- Added boxed requirements: Defined requirements within the text and assigned to Conformance Classes

11.6.1 Improved Distributed Search

- Limited Distributed Search in CS 2.0.2:

  - For a description of the problems see presentation: Status, problems and improvements of distributed/federated search of OGC CSW 2007 in Stresa
  - For improved DistributedSearch in CSW 3.0: see presentation: CSW 3.0 – Change request for Distributed Search (Annex B) of OGC CSW 2008 in Potsdam

11.6.2 Queryables

- Added TemporalExtent as queryable (Type gml:TimePeriod) and returnable (new csw30:TemporalExtentType)

11.6.3 Alignment with other OGC Specs

- Aligned with OWS Common 2.0:
  - removed parts already defined there
  - Exception codes are identified for the CSW operations (#240)
- Aligned CSW with Filter 2.0: Filter 2.0 can be used with ANY version of GML.

11.6.4 Definition of a Basic-Catalogue

The Basic-catalogue conformance class is defined as:

- GetCapabilities / KVP-GET

- GetRecordById / KVP-GET

- GetRecords / KVP-GET

- Filter-KVP (basic retrieval parameters) conformance class:

- Implements the Q-, RECORDIDS-, BBOX-, GEOMETRY, GEOMETRY_CRS, RELATION, DISTANCE, DISTANCE_UOM, LAT, LON, RADIUS KVP-parameters and TIME and TRELATION parameters

- Implements CSW- and ATOM-response

- Implements OpenSearch conformance class

11.6.5 OGC CS 3.0 / CSW – XSD

- Refactored schemas to follow common file naming practices
- CSW 3.0 additionally builds upon the following schemas:
  - OGC ows/2.0 (in OGC schema repository)
  - OGC filter/2.0.0 (in OGC schema repository)

- OGC gml/3.2.1 (in OGC schema repository)
  - OGC xlink/1.0.0 (not included)

- New namespace: http://www.opengis.net/cat/csw/3.0
- Extension of discovery request and response messages to support new distributed search concepts
- Elementname now of type xsd:string instead of xsd:Qname
- AbstractRecordType has now optional attribute "deleted"
- XML attributes service ("CSW") and version ("3.0.0") are not more fixed (can be overwritten in profiles) (CR 08-098)
- all elements/types related exclusively to DescribeRecord are deleted
- Synchronized schema fragments in the spec with the refactored schemas.

11.6.6 Changes on operations

- Fixed a number of inconsistencies between XML and KVP encodings
- Clarifications on relation: HTTP message headers and request parameters
- attribute „deleted“ of AbstractRecord defines if metadata item was deleted (time of deletion can be expressed by including an element like dct:modified)

  ➢ **GetRecords:**

- csw:Record left as mandatory common information model across catalogues BUT ATOM now a mandatory response-only format
- CONSTRAINTLANGUAGE parameter now of type anyURI (#281):
- FILTER -> http://www.opengis.net/fes/2.0
- CQL -> http://www.opengis.net/csw/3.0/cql
- Removed resultType parameter since there are other means for performing resultType=hits (i.e. maxRecords=0), etc.
- Now: KVP-encoding for queries
- KVP Params grouped according kind of searching they enable: Query class
  - "Text search": full text queries

  - o “Record search”: allow single records to be retrieved
  - o “Spatial search”: spatial search-ing based on bbox, center-point-radius or geometry
  - o “Temporal search”: temporal searching based on period.

- …bbox=43.6050,-79.4271,43.6915,-79.3162,urn:ogc:def:EPSG::4326&…
- …&time=2012-01-10/2012-12-31&…
- CONSTRAINT and –VERSION parameters may be used for complex filters encoded using CQL and Filter
- In doc: Section on Enabling OpenSearch
  - o Description document auto-discovery
  - o Requirements for an Open- Search enabled CSW
  - o The canonical response to an OpenSearch query is an XML-encoded ATOM document as described in clause 9.3 of OGC 10-032r2
- maxRecords: union to number and the string 'unlimited' (indicating that all records shall be returned) (#221)
- attribute "processContents" of “SearchResultsType -> xsd:any” (see "choice") now changed to "strict“
- +#278, #277: Asynch/Synch behaviour: Multiple values for ResponseHandler
- +#228,#230: Added ElementSetName_TypeName (the typeName to be returned) for KVP encoding
- +#220: Data type of the GetRecords "startPosition" parameter

➢ **Capabilities Document**

- Filter_Capabilities in Capabilities now optional (CR 08-099)
- Default for ows:GetCapabilitiesType->ows:ServiceType now "CSW" (aligned with other OGC specs)
- Sorting: Default sorting
  - o per title OR
  - o by a predefined sorting algorithm

        ▪ defined as link

- Valid sortable terms can be specified
- valid queryable can be defined as well
- Definition of Conformance Classes supported
- updated sample Capabilities in spec

➢ **GetRecordById**

- getRecordByID response (for HTTP/XML/POST and HTTP/KVP/GET) now raw response in its original format,
- conformant with the outputFormat and outputSchema.
- GetRecordByIdResponse only available for SOAP (see WSDL)
- Only 1 id as input, to request more use GetRecords/KVP
- defined “outputSchema" as additional parameter

➢ **GetDomain:**

- Revised the XML schema types
- We counterchecked the capabilities of GetDomain with requirements of modern search interfaces, like Autosuggest, DidYouMean, ..
- able to interrogate any request parameter and info model component
- optional operation, availability of operation can be determined by checking Capabilities
- Added filter on parameters so one can restrict results to values that satisfy the filter.
    - o Permits a sort of type-ahead suggestions solution
    - o Examples: find values between 10 and 20, or start with cat*
    - o Available filters are listed in FilterCapabilities section
- Added a ResultType parameter to return all 'possible' (enumeration) versus 'available' (have such values in the database)
- Response container on 'available' will return number of records with that value.

- **Transaction:**

- Harmonized Update-, Insert-, DeleteType: typeName attribute

- Constraint in the capabilities providing list of URI's identifying that schemas upon which the Transaction operation may be applied.

- **Harvest:**

- Support added for HTTP/POST with attachments for Harvesting of resources that cannot be referenced via URL.

- **UnHarvest:**

- UnHarvest operation added to the HTTP binding: complementary operation for Harvest.

11.7 New WSDL version

- Now single namespace: http://www.opengis.net/cat/csw/3.0/wsdl (important for ws-i basic profile conformance)

- Now two portTypes: cswDiscovery and cswTransaction

- Now two SOAP- and POST-bindings: csw30-Discovery-SOAP/csw-Discovery-POST and csw30-Transaction-SOAP/csw-Transaction-POST

- dropped describeRecord

- redefined GetRecordByIdResponse-message:

```
<wsdl:message name="GetRecordByIdResponse">

    <wsdl:part name="Body"     element="csw30:CSW30GetRecordByIdResponse"/>

</wsdl:message>
```

  - CSW30GetRecordByIdResponse / CSW30GetRecordByIdResponse-Type defined in WSDL (not in CSW30-schema) as only the XML/SOAP-binding shall wrap the metadata record returned.

  - For XML/POST and KVP/GET bindings the metadata record returned should not be wrapped by a CSW30GetRecordByIdResponse.