OGC® DOCUMENT: 00-027

OGC®
Making location count.

CONFORMANCE TEST GUIDELINES FOR OPENGIS CATALOG SERVICES SPECIFICATION FOR CORBA

TEST SUITE

PUBLISHED

Version: 1.0
Submission Date: XXX
Approval Date: XXX
Publication Date:
Editor: Laura Diaz, Bart van der Eijnden, Barend Gehrels

OGC®
Making location count.

License Agreement

Use of this document is subject to the license agreement at https://www.ogc.org/license

Copyright notice

Note

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.

CONTENTS

I. SECURITY CONSIDERATIONS v

II. SUBMITTING ORGANIZATIONS vi

III. SUBMISSION CONTACT POINTS vi

IV. EDITORS vi

V. DOCUMENT CONVENTIONS vii

1. OVERVIEW 9

2. BACKGROUND 11
2.1. Catalog Services Implementation Specification 11
2.2. CG_CatalogService Interface 12
2.3. CG_Discovery Interface 12

3. ENVIRONMENT AND SETUP OF THE CATALOG CERTIFICATION PROGRAM 14
3.1. Setting up the environment 14
3.2. Setting up the server 14
3.3. Running the client 14

4. DESCRIPTION OF THE TEST PROCEDURES 16
4.1. Introduction 16
4.2. Operations of the CG_CatalogService interface 16
4.3. Operations of the CG_Discovery interface 17

ANNEX A (INFORMATIVE) METADATA DATABASE IN XML FORMAT 26

LIST OF FIGURES

Figure 1 — The Coarse-Grained (CG) General Interface Model. 12

Figure 2 — Outline of the parameters (boxes) involved in the Query operation and their possible values (circles). 18

Figure 3 — Parameters (in boxes) and their possible values (in circles) of the present operation of the CG_Discovery interface. 22

Figure 4 — The parameters (in boxes) and their possible values (in circles) of the ExplainCollection method. 23

I SECURITY CONSIDERATIONS

No security considerations have been made for this document.

II SUBMITTING ORGANIZATIONS

The following organizations submitted this Document to the Open Geospatial Consortium (OGC):

- Geodan Holding bv, the Netherlands

III SUBMISSION CONTACT POINTS

All questions about the submission should be directed to:

Barend Gehrels
Geodan SDT bv, the Netherlands
tel: +31 73 692 5151
fax: +31 73 692 5150
E-mail: barend@geodan.nl
http://www.geodan.nl

IV EDITORS

The following editors contributed to this document

Laura Diaz, on behalf of Geodan Holding bv
University of Valencia, Spain
E-mail: laudiaz2@alumni.uv.es
http://www.uv.es

Bart van der Eijnden
Geodan SDT bv, the Netherlands
tel: +31 73 692 5151
fax: +31 73 692 5150
E-mail: bart@geodan.nl
http://www.geodan.nl

Barend Gehrels
Geodan SDT bv, the Netherlands
tel: +31 73 692 5151
fax: +31 73 692 5150
E-mail: barend@geodan.nl
http://www.geodan.nl

DOCUMENT CONVENTIONS

The Courier New font has been used to indicate code segments.

1

OVERVIEW

1 OVERVIEW

This document describes the guidelines that the Open GIS Consortium will use for testing software implementations with respect to conformance to its specification entitled Revision 1.0.

The Open GIS Consortium, Inc. (OGC) maintains a brand (in the form of a certification mark) that cannot be used in connection with a software product by any organization unless it has submitted a software product to OGC's conformance test, successfully completed this test, and received a certificate stating such success. Organizations that have earned the certification mark may use it in ways defined within this document. This set of rules ensures that users who buy products that are branded can be sure that the products carrying the certification mark have been submitted to a test. The primary purpose of the conformance test is to protect the value of the OpenGIS® brand as an element of OGC's program to promote interoperability between diverse geoprocessing systems.

This document contains the following:

- An outline on the background of Catalog Services;
- A general description of how to configure the environment for the Catalog Certification Program;
- A description of the testing procedures of the Catalog Certification Program.

2

BACKGROUND

2 BACKGROUND

2.1. Catalog Services Implementation Specification

According to the OpenGIS Catalog Services Implementation Specification, OGC Catalog servers have three kinds of services:

- **Discovery Service;**
- **Access Service;**
- **Management Service.**

Discovery Services are those services which allow a client to locate metadata that describe data. Access Services provide the client with methods to request services on the data. Access Services have been divided into two types: Direct Access and Brokered Access. Management Services define methods for a client to change the metadata held by a catalog.

The Discovery Service has to be provided by all Application Servers claiming compliance with the OGC Catalog Interface. The Access and Management Services are optional for an OGC compliant catalog.

The Catalog Services General Model provides the standards of compliance for specific implementation communities, in a profile-based way. There are different profiles for each Distributed Computing Platform, like CORBA, OLEDB and WWW. The test program is specific for CORBA in combination with a coarse-grained interface. This means that the client and server have very little knowledge of each other. Therefore, the interface in this environment must be flexible, well-defined and simple.

As specified in the Coarse-Grained General Interface Model there are four major interfaces, `CG_CatalogService`, `CG_Discovery`, `CG_Access` and `CG_Manager`. These interfaces allow the discovery, access and management of geospatial data and services. The above-mentioned model has been based on the concept of interface operations passing Request-Response Message Pairs between a client and a server. Stated another way, the Coarse-Grained architecture uses a messaging-based structure to describe the access and invocation of Catalog services.

The Catalog Certification Program is centered on the `CG_Discovery` interface, it will test all the functionality supported by the discovery service as well as all the functionality of the `CG_CatalogService` interface. The Catalog Certification Program focuses on these two interfaces because as said before the Access and Management Service are optional.

2.2. CG_CatalogService Interface

Server level interfaces provide access to the services that support the establishment and management of a user session through operations. The operations included in this interface that have to be supported by all servers are listed in Figure 1. The `InitSession` operation generates a unique identifier used to track the context of a session. To terminate a session, the `TerminateSession` operation can be used. The `Status` operation checks the status of a currently pending request. To terminate a request, the `Cancel` operation should be used. The `ExplainServer` operation lists all conventions and services available during the current session.

In the current version of the Catalog Certification Program only synchronous requests are considered. This rules out the possibility to test for the `CancelRequest` and `Status` operation.

2.3. CG_Discovery Interface

The `CG_Discovery` Interface provides users with a way to discover what data, services and other resources are available to them. This interface does not provide access to the resources themselves. The operations provided by this interface can be deducted from Figure 1. The `Query` operation searches for data on or services from a given catalog server and may return records from the result set. The `Present` operation retrieves records from a result set created by issuing a query. To get an explanation of the data model of the catalog the `ExplainCollection` operation can be used.

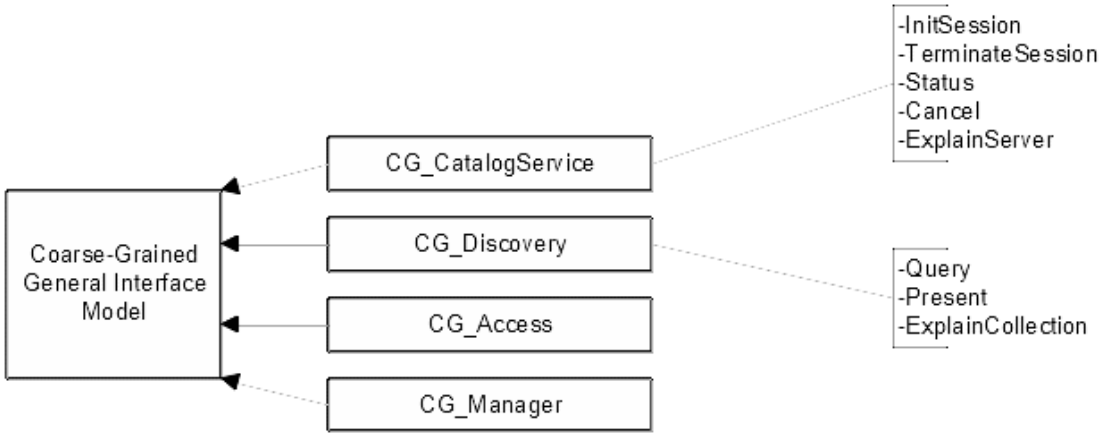

Figure 1 — The Coarse-Grained (CG) General Interface Model.

3

ENVIRONMENT AND SETUP OF THE CATALOG CERTIFICATION PROGRAM

3 ENVIRONMENT AND SETUP OF THE CATALOG CERTIFICATION PROGRAM

3.1. Setting up the environment

The Catalog Certification Program has been developed with JDK 1.2.2. The code is in a single file called `catalogCertification.java`. The module `OGC_CatalogService` has been defined in one `idl` file (`ogc_cat.idl`), the idl file contains all definitions, i.e. the message definitions and the required interfaces.

To translate the idl file to Java classes, the file ij.bat has been written. This batch file uses `idl2java` of VisiBroker for Java 3.4. To compile all Java classes of the module `OGC_CatalogService` and the `catalogCertification.java` file, the `cj.bat` file has been provided. These two compilation procedures are only necessary if the version of ORB for Java differs from that in VisiBroker for Java 3.4.

3.2. Setting up the server

For performing the conformance test, the server should be started and it should write an IOR file. The IOR file is used by the Catalog Certification Program to obtain a reference to the server. For a distributed query two servers should be started.

3.3. Running the client

The Catalog Certification Program functions like a client. To start the Catalog Certification Program, run the `catalogCertification` file.

4

DESCRIPTION OF THE TEST PROCEDURES

4 DESCRIPTION OF THE TEST PROCEDURES

4.1. Introduction

This section details the tests that have to be executed to test all the operations mentioned in Clause 2. In order to test the operations, a default query is sent to the catalog server being tested. The database is a known XML (eXtendable Markup Language) file. The retrieved results will be compared with the expected results. A series of queries, applying multiple aspects of the Query Language, will be tested.

To test for a distributed search, two servers are used using both the sample XML file.

An example of the database with metadata is given in Annex A. If a server is not capable of using a database in XML format, the XML file should be ported to a supported format.

4.2. Operations of the CG_CatalogService interface

4.2.1. Testing the InitSession operation

In order to test whether or not the InitSession operation is supported by the server, the Catalog Certification Program follows the following steps:

1. It tries to read the default IOR (Interoperable Object Reference) file, which contains the address of the catalog server;
2. It initializes the ORB (Object Request Broker);
3. It creates a reference to the catalog server;
4. If step 3 has been successful, it creates an `InitSession` request and invokes this method on the server;
5. It retrieves the identifier, which will be used in the current session.

If any of these steps fail, the `InitSession` operation is not supported by the server being tested.

4.2.2. Testing the `TerminateSession` operation

The Catalog Certification Program tests whether or not the identifier of the session and the reference to the catalog server exist. If these two constraints are met, the `TerminateSession` method is invoked and a `TerminateSession` response is sent by the server.

If the status of the `TerminateSession` response is successful and no exception has been raised, the `TerminateSession` operation is considered supported. In order to be compliant with the specification, the server should send a successful status when the session has been terminated in a correct way.

4.2.3. Testing the `ExplainServer` operation

For invoking this method on the catalog server, the `capabilities` field in the request is filled with all the `ctAllSupportedRequest`. After invoking this method, the server returns a response, which should contain a list of all the capabilities supported by the server. There is no *status* in the ExplainServer response. The Catalog Certification Program works as follows: if the catalog server sends an empty list or if an exception is raised during the process of testing, the ExplainServer operation will be considered not supported. For a full compliance the server should return a list of all capabilities that are supported, the correct capabilities are specified in the OpenGIS Catalog Services Implementation Specification as `CG_CapabilityType`. At least the `OGC_Common QueryLanguage` should be supported, the contents of the other capability types can not be tested.

4.3. Operations of the CG_Discovery interface

4.3.1. Query

A query is built by combining a set of parameters in a request. For some parameters there is no viable way to test if their different values are working or they are part of an asynchronous operation. To overcome this problem, their value is fixed in the request (`iteratorsize=100`, `cursor=0`, `asynchronous=false`, `maxlevel=2`, `presentation=full`, `entrytype=collection`).

There is a general approach for testing the operations, although some of them differ slightly from this general approach. The general approach can be described as:

- if a general exception occurs during the *query* call, the capability which is being tested at that moment will be considered *not tested*;
- if a successful status of the response has been retrieved, the capability will be considered *supported*;

- if an unsuccessful status has been retrieved, it depends on the capability if it is considered not supported or not tested.

If, during the initial invocation of the query operation, a general exception is raised or an invalid status is returned, the query operation will be considered as not supported. This test checks the results retrieved in NV format (name-value pairs). If the NV format is not supported by the server, the query method will only be considered supported if no exception has been raised during a call with another format and if the retrieved status is successful. The parameters tested and their possible values are shown in Figure 2:

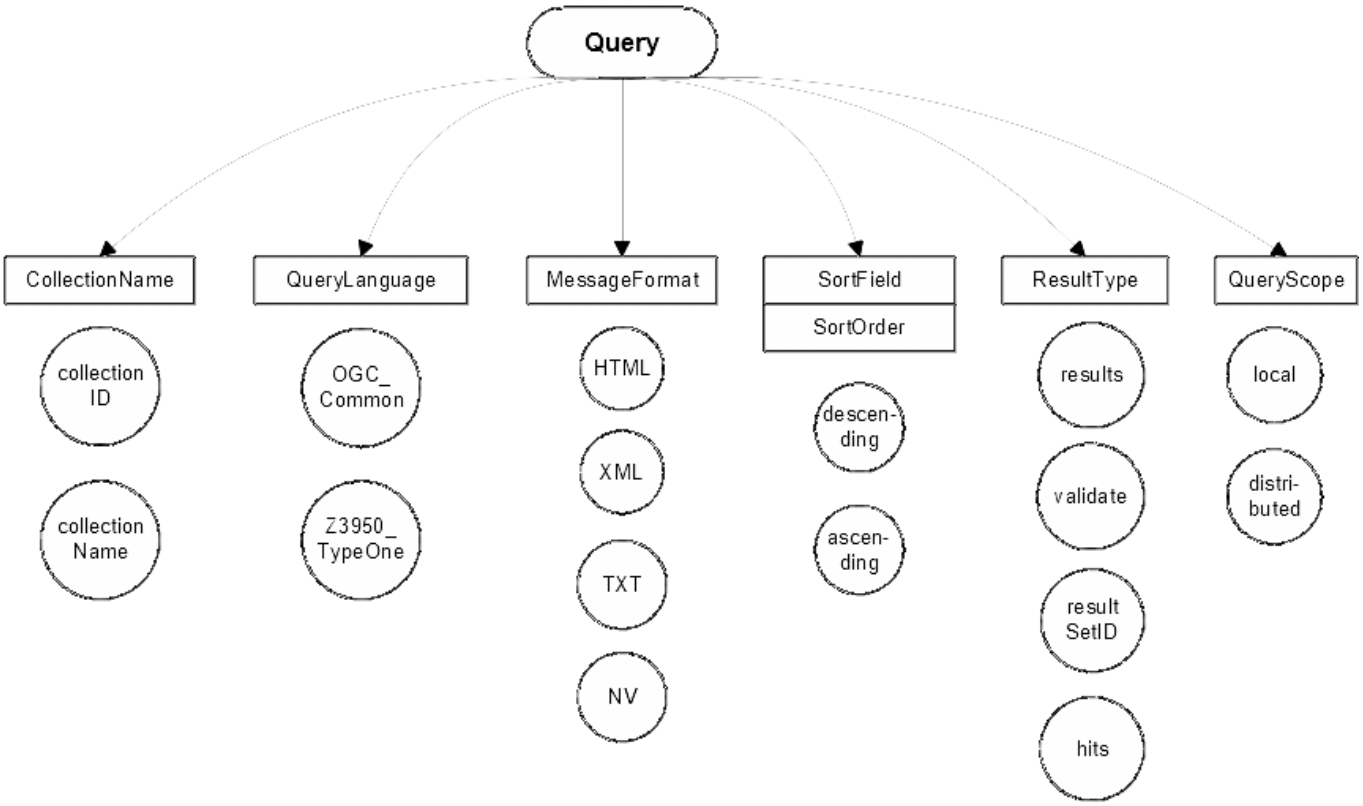

Figure 2 — Outline of the parameters (boxes) involved in the Query operation and their possible values (circles).

4.3.1.1. CollectionName

The `CollectionName` parameter can have two possible values:

- `CollectionID`: this parameter value combination is considered successful if, after invoking the query method, the status returned is success and there are no problems with reading the string that this field contains;
- `CollectionName`: the same procedure is followed as with CollectionID.

When the query method is invoked there are several parameters in the request and there is no way to know which parameter is failing if an unsuccessful status is retrieved. If a general

exception occurs during the remote call this capability will not be tested, as we do not know for sure which parameter is causing the exception.

4.3.1.2. QueryLanguage

While testing this parameter, the `ReturnFormat` is set to `NV`, the `QueryScope` is `local` and the `ResultType` equals `results`. This means this part of the test depends strongly on the success of the NV format. For the `QueryLanguage` parameter two values are valid:

- `OGC_Common`: if an exception is returned, this capability will be considered as not tested. If the response status is unsuccessful, the capability is considered not supported. If none of the above failures occurs, and if the retrieved number of hits is correct, the capability will be considered supported;
- `Z3950_TypeOne`: the same procedure applies to this language as to the `OGC_Common` language.

4.3.1.3. MessageFormat

In order to test the different values of this parameter, the `QueryScope` is set to `local`, the `ResultType` is `results` and the `QueryLanguage` `OGC_Common`. The series of queries is invoked in combination with the following values of MessageFormat:

- XML: if an exception is raised during invocation of the query method or if the status is unsuccessful, this capability (`MessageFormat=XML`) will not be tested. Otherwise, the program will test if the return format is the correct one. First of all the program will compare if the `returnFormat` field in the request matches the `retrieveData.encoding` field in the response. After this, the retrieved string will be read and searched for `<?xml`. If both conditions are met, the capability will be considered supported;
- HTML: the same applies to this format, only in this case the string is searched for `<html>` and `</html>`;
- TXT: the same applies here, but in this case, the string cannot be searched for a specific substring, so it can only be tried to read the string;
- NV: the same applies, it will be tried to read the retrieved NV values, if the number of NV pairs matches the correct number of hits, this capability is considered supported.

4.3.1.4. Number of hits retrieved

To test this parameter, the `QueryScope` is `local`, the `ResultType` equals `results`, the `MessageFormat` is `NV` and the `QueryLanguage` is set to `OGC_Common`. If the NV format is not supported by the server, it's not possible to test if the field containing the number of hits in the response is working. The only thing that is tested for is whether or not this field contains the number of hits found in the search which should equal the number of NV pairs (it doesn't have

to be the correct number of hits which is to be expected, but it can be). If an exception occurs or if the status is unsuccessful, this capability will not be tested.

4.3.1.5. SortOrder

Again, the `QueryScope` is set to `local`, the `ResultType` is `results`, the `MessageFormat` equals `NV` and the `language` is `OGC_Common`. The `SortOrder` has two possible values: ascending or descending. The sort order should always be used in conjunction with a `SortField`, like in the following example:

```
sortField[0] = OGC_CatalogService.CG_SortField ("title", OGC_CatalogService.CG_
SortOrder.ascending)
```

In this example the Catalog Certification program will test whether or not the results are ordered by title in an ascending way. The possibility of testing this capability depends on the support of the NV format. If an exception occurs, this capability will not be tested. If the status is successful and the NV format is supported, the sorting order will be checked using an array.

4.3.1.6. ResultType

There is no easy way to test this parameter, it is difficult to know if a query call is failing due to the query method itself or due to another capability. There are four different values possible for the parameter `ResultType`, they are:

- Results: if an exception is raised or a status is unsuccessful, the results capability will be considered not supported. In order to be considered supported, the test should work for one or more of the formats. When the Catalog Certification Program invokes the query method for testing the NV format, it will test the results too. If a general exception occurs, the status obtained will be the status obtained with another format. If the status is unsuccessful or if the retrieved number of hits doesn't match the correct number of hits, this capability will be considered unsupported;
- Validate: if an exception is raised or if an unsuccessful status is retrieved, this capability will be considered not supported. Otherwise, the program will test if the `retrieveData` field in the response is empty (this should be the case because only a validation should be performed). Testing if the field is empty is done using the txt format and looking for an empty string `""`;
- Hits: testing this capability is almost equivalent to the Validate test. In addition, the number of hits is checked;
- `ResultSetID`: this capability is tested equivalent to the Validate test with one addition: the program also checks whether or not the `collectionID` field is filled.

4.3.1.7. QueryScope

The `MessageFormat` is fixed as `NV`, the `ResultType` as `results`. The two options for `QueryScope` are:

- Local: if an exception occurs during the invocation of the query method, this capability will not be tested. If the status is successful and the correct number of hits is retrieved, this capability will be considered as supported;
- Distributed: because of the use of 2 servers during the test of the distributed query, the correct number of hits should be 2 times the correct number of hits using a local scope.

4.3.1.8. GeographicBoundingBox

The `QueryScope` is local, the `MessageFormat` equals `NV` and the `ResultType` `results`. The `queryExpression` used is given by:

```
queryExpression=OGC_CatalogService.CG_QueryExpression
("intersects(GeographicBoundingBox,envelope(40.71,-24.17,71.26,27.63))","",
OGC_CatalogService.CG_QueryLanguage.OGC_Common)
```

The correct number of hits for this query, with respect to the standard metadata datbase, is 2. If an exception occurs, this capability will be considered not tested. If the status of the response is invalid or if the returned number of NV pairs doesn't match the expected number of hits, this capability is considered not supported.

4.3.1.9. Invalid query request

This capability will show if the server is able to detect an invalid query and act in the right way. The `MessageFormat` is again of the `NV` type. The query which is invoked is: `abstract lkdd 'river%'`. The server has to return an exception or an invalid status in order to consider this capability as supported.

4.3.2. Present

In order to test for this operation of the `CG_Discovery` interface, some parameters have to be fixed in the request, these are: `Iteratorsize(100)`, `Cursor(0)`, `Presentation(full)` and `ResultSetID(ResultSetID sent by the query)`. Other parameters and their possible values are shown in Figure 3.

The general approach which applies to the testing of the capabilities of present is equal to the general approach of the query method (see Clause 4.3.1). First of all the Catalog Certification Program will set the `Present` capability to not supported but in case one of the calls was successful the capability will be considered supported. If a successful status is retrieved, the last check will be using the NV format, i.e., the program will check if the functionality of this method is working testing the results retrieved with NV format. If the NV format is working the

present capability will be shown as supported if the number of hits retrieved is the number of hits expected by the test.

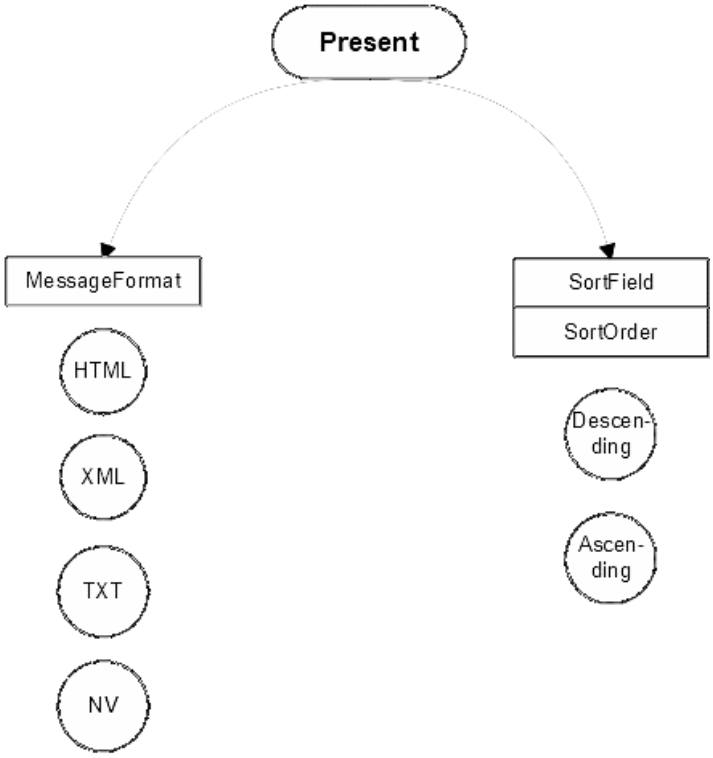

Figure 3 — Parameters (in boxes) and their possible values (in circles) of the present operation of the CG_Discovery interface.

4.3.2.1. MessageFormat

The possible values for this parameter are equivalent to those of the query method:

- XML: the same applies here as for the MessageFormat of the query method (Clause 4.3.1.3);
- HTML: idem;
- TXT: idem;
- NV: idem.

4.3.2.2. Number of hits retrieved

The field containing the number of hits has to have a value equivalent to the number of NV pairs. After invocation of the present method, no exception should be raised and a successful status should be returned. In this case, this capability will be considered supported.

4.3.2.3. SortOrder

There are two options for this parameter: `ascending` and `descending`. If the NV format in present is working, this parameter can be tested. The same applies here as to `SortOrder` of the query method (Clause 4.3.1.5).

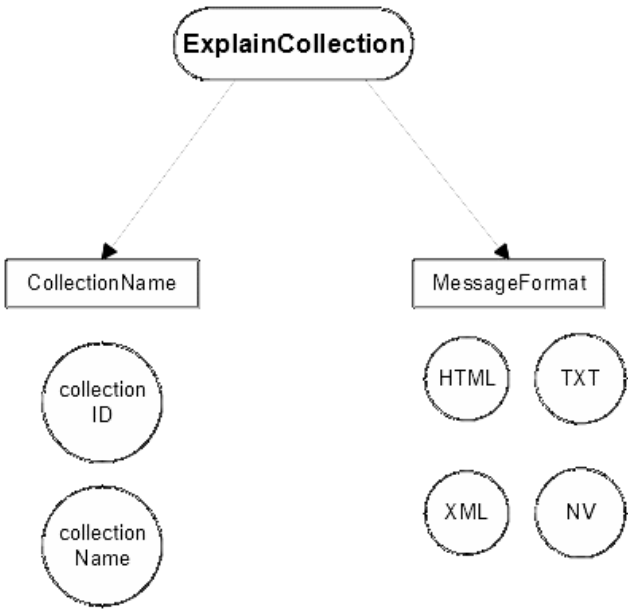

Figure 4 — The parameters (in boxes) and their possible values (in circles) of the ExplainCollection method.

4.3.3. ExplainCollection

The `ExplainCollection` is the last operation of the `CG_Discovery` interface to be detailed. In order to test this operation, the following parameters are fixed:

- `collectionID.collectionID("")`
- `AttributeCategory = OGC_CatalogService.CG_AttributeCategory.both`

There is no viable way to test all the parameters of this method. Both the `CG_ExplainCollectionRequest` and the `CG_ExplainCollectionResponse` contain parameters which cannot be tested. The parameters that are tested are shown in Figure 4. If a general exception is raised during the remote call the different parameters will be considered not tested. The overall `ExplainCollection` capability will first be set to not supported, if one of the submethods is supported, the overall capability is also set to supported. Because of the combination of parameters it's difficult to render a probable cause of a failure.

4.3.3.1. CollectionName

As with the query method, the `CollectionName` can have a value of:

- `CollectionID`: if it's possible to read the string that this field contains and if the status is successful, this capability will be considered supported;
- `CollectionName`: the same applies here as to `CollectionID`.

4.3.3.2. Format

As with `Query` and `Present`, the format can be one of the following options:

- XML: in this case there is no `retrievedData` field in the response sent by the server, so the only way to test for this capability is to check whether or not the status is successful;
- HTML: the same applies here as to XML;
- TXT: idem;
- NV: idem.

4.3.3.3. Parameters not tested

The `attributeCategory` could be tested in the same way the format is, i.e., testing if the status is successful but there is no way to know if the `queriable` or `presentable` attributes are working in a correct way. The `dataModel` attribute is not tested for because it's impossible to test the information each database contains.

A

ANNEX A (INFORMATIVE) METADATA DATABASE IN XML FORMAT

A

ANNEX A (INFORMATIVE) METADATA DATABASE IN XML FORMAT

The XML file required for a good operation of the Catalog Certification Program is:

```
<?xml version="1.0"?>
<Database>
  <Metadata>
    <Title>Provinces of the Netherlands</Title>
    <Abstract>This dataset contains the 12 provinces which make up the
Netherlands</Abstract>
    <GeographicBoundingBox>
      <westBoundLongitude>1.34</westBoundLongitude>
      <eastBoundLongitude>5.00</eastBoundLongitude>
      <northBoundLatitude>70.50</northBoundLatitude>
      <southBoundLatitude>20.70</southBoundLatitude>
    </GeographicBoundingBox>
  </Metadata>
  <Metadata>
    <Title>Communities of the Netherlands</Title>
    <Abstract>This metadata describes the geographic file containing the
municipalities of the netherlands</Abstract>
    <GeographicBoundingBox>
      <westBoundLongitude>3.34</westBoundLongitude>
      <eastBoundLongitude>7.22</eastBoundLongitude>
      <northBoundLatitude>53.54</northBoundLatitude>
      <southBoundLatitude>50.74</southBoundLatitude>
    </GeographicBoundingBox>
  </Metadata>
  <Metadata>
    <Title>Railroads of the Netherlands</Title>
    <Abstract>This dataset contains the railroads of the Netherlands</Abstract>
    <GeographicBoundingBox>
      <westBoundLongitude>2.50</westBoundLongitude>
      <eastBoundLongitude>10.00</eastBoundLongitude>
      <northBoundLatitude>25.75</northBoundLatitude>
      <southBoundLatitude>50.75</southBoundLatitude>
    </GeographicBoundingBox>
  </Metadata>
  <Metadata>
    <Title>Railroad stations of the Netherlands</Title>
    <Abstract>This dataset contains the railroad stations of the Netherlands</
Abstract>
    <GeographicBoundingBox>
      <westBoundLongitude>1.90</westBoundLongitude>
      <eastBoundLongitude>3.60</eastBoundLongitude>
      <northBoundLatitude>73.55</northBoundLatitude>
      <southBoundLatitude>70.75</southBoundLatitude>
    </GeographicBoundingBox>
  </Metadata>
  <Metadata>
    <Title>Nuts areas of Europe</Title>
```

```
    <Abstract>This dataset contains the NUTS-regions of Europe</Abstract>
    <GeographicBoundingBox>
      <westBoundLongitude>-25.15</westBoundLongitude>
      <eastBoundLongitude>25.70</eastBoundLongitude>
      <northBoundLatitude>50.25</northBoundLatitude>
      <southBoundLatitude>10.65</southBoundLatitude>
    </GeographicBoundingBox>
  </Metadata>
  <Metadata>
    <Title>Countries of Europe</Title>
    <Abstract>This dataset contains the countries of Europe</Abstract>
    <GeographicBoundingBox>
      <westBoundLongitude>-24.17</westBoundLongitude>
      <eastBoundLongitude>40.71</eastBoundLongitude>
      <northBoundLatitude>71.26</northBoundLatitude>
      <southBoundLatitude>27.63</southBoundLatitude>
      </GeographicBoundingBox>
  </Metadata>
  <Metadata>
    <Title>Rivers of Europe</Title>
    <Abstract>This dataset contains the Rivers of Europe</Abstract>
    <GeographicBoundingBox>
      <westBoundLongitude>-24.17</westBoundLongitude>
      <eastBoundLongitude>40.71</eastBoundLongitude>
      <northBoundLatitude>71.26</northBoundLatitude>
      <southBoundLatitude>27.63</southBoundLatitude>
    </GeographicBoundingBox>
  </Metadata>
  <Metadata>
    <Title>Mountains of Italy</Title>
    <Abstract>This dataset contains the mountains of Italy</Abstract>
    <GeographicBoundingBox>
      <westBoundLongitude>2.15</westBoundLongitude>
      <eastBoundLongitude>8.70</eastBoundLongitude>
      <northBoundLatitude>20.25</northBoundLatitude>
      <southBoundLatitude>1.05</southBoundLatitude>
    </GeographicBoundingBox>
  </Metadata>
</Database>
```