

# Open Geospatial Consortium Inc.

Date: 2009-04-08

Reference number of this OGC® project document: OGC 09-083

Version: 3.0.0-Draft

Category: OGC® Implementation Specification

Editor: Adrian Custer

## GeoAPI Implementation Specification

### Copyright notice

Copyright © 2009 Open Geospatial Consortium, Inc. All Rights Reserved.  
To obtain additional rights of use, visit <http://www.opengeospatial.org/legal/>.

### Warning

This document is not an OGC Standard. It is distributed for review and comment. It is subject to change without notice and may not be referred to as an OGC Standard.

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

**Document type:** OGC® Publicly Available Implementation Standard  
**Document stage:** Draft  
**Document language:** English



## Contents

<b>i.Preface</b>	<b>iv</b>
<b>ii.Submitting organizations.....</b>	<b>v</b>
<b>iii.Submission contact points.....</b>	<b>v</b>
<b>iv.Revision history.....</b>	<b>v</b>
<b>v.Changes to the OGC® Abstract Specification.....</b>	<b>v</b>
<b>1 Scope</b>	<b>1</b>
<b>2 Conformance.....</b>	<b>1</b>
<b>3 Normative references.....</b>	<b>2</b>
<b>4 Terms and definitions .....</b>	<b>3</b>
<b>5 Conventions.....</b>	<b>3</b>
<b>5.1 Symbols (and abbreviated terms).....</b>	<b>4</b>
<b>5.2 UML Notation.....</b>	<b>4</b>
<b>6 A Geographic API in Java.....</b>	<b>6</b>
<b>7 Annotation package.....</b>	<b>8</b>
<b>8 Utility package.....</b>	<b>9</b>
<b>9 Metadata packages.....</b>	<b>16</b>
<b>10 Geometry packages.....</b>	<b>19</b>
<b>11 Referencing and Parameter packages.....</b>	<b>21</b>
<b>Annex A</b>	
<b>Conformance.....</b>	<b>26</b>
<b>Annex B</b>	
<b>GeoAPI Source Java Archive.....</b>	<b>27</b>
<b>Annex C</b>	
<b>UML Diagram for referencing Operation types.....</b>	<b>28</b>
<b>Annex D</b>	
<b>Departures from the ISO standards.....</b>	<b>29</b>

## i. Preface

This GeoAPI specification grows out of a long effort at the Open Geospatial Consortium (OGC) and in the free software community aiming to develop a library of interfaces defining a coherent data model for the manipulation of geospatial data based on the data model defined in the specifications of the OGC. The library has been developed to facilitate the creation of interoperable, standards compliant, Java language software.

The GeoAPI interface library originates with the publication in January 2001 of the implementation specification OGC 01-009 *Coordinate Transformation Services* Revision 1.00 (Martin Daly, ed.) which included a set of interfaces written in the Java language and in the org.opengis namespace. The GeoAPI project started in 2003 as an effort from several contributors to develop a set of Java language interfaces which could be shared between several projects. The GeoAPI project subsequently considered the interfaces of OGC 01-009 as version 0.1 of the GeoAPI library and started working on GeoAPI 1.0 in collaboration with developers writing the OGC specification *Geographic Objects*. Subsequently, the Open Geospatial Consortium jettisoned its own Abstract Specifications and adopted, as the basis for further work, the standards developed by the Technical Committee 211 of the International Organization for Standardization (ISO) in its ISO 19100 series. The GeoAPI project therefore realigned its library with those standards. In 2003, version 1.0 of the GeoAPI library was released to match the release of the first public draft of the implementation specification OGC 03-064 *GO-1 Application Objects* Version 1.0 (Greg Reynolds, ed.). The standardization effort of GO-1 took a couple of years during which extensive work was made on the GeoAPI library. Release 2.0 of the GeoAPI library was made at the time of the final publication of the GO-1 specification in 2005. This brief historical synopsis explains why this specification adopts the version number 3.0 despite there being no prior OGC specification of the same name. We expect to release version 3.0 of the GeoAPI library with the final version of this specification. *During the preparation of this document, all references to the Java interfaces relate to the current 2.3-SNAPSHOT version of the GeoAPI library.*

The GeoAPI library and its reference implementation provide the OGC dual benefits. The reference implementation demonstrates to the standards writers that it is possible to develop a single, coherent implementation of all the ISO/OGC specifications covered by the standardized API. The API provides the OGC community with a new point of interoperability between client code written to use the API and library code written to implement the API, with this layer of interoperability explicitly based on the interfaces defined by the core standards of the OGC.

## ii. Submitting organizations

The following organizations submitted this Implementation Specification to the Open Geospatial Consortium Inc.:

- a) Geomatys, Arles, France.

## iii. Submission contact points

All questions regarding this submission should be directed to the editor:

Adrian Custer  
Geomatys  
24, Rue Pierre-Renaudel  
13200 Arles, France  
[adrian.custer@geomatys.fr](mailto:adrian.custer@geomatys.fr)

## iv. Revision history

Date	Release	Author	Paragraph modified	Description
04/06/09	3.0.0-Draft	Adrian Custer	All	Initial Public Draft

## v. Changes to the OGC<sup>®</sup> Abstract Specification

The OGC<sup>®</sup> Abstract Specification does not require changes to accommodate this OGC<sup>®</sup> standard.

## Foreword

The GeoAPI interface library is developed by the GeoAPI project whose public web page is available through (<http://www.geoapi.org/>). These interfaces have been developed over a number of years with contributors acting as individual volunteers, as government or institutional workers, or as employees in technology companies. The formal list of contributors is maintained in the project documentation at <http://www.geoapi.org/team-list.html> but many others have contributed to the project through discussions at meetings of the Technical Committee of the OGC, on the project mailing lists and elsewhere, by working on implementations or client code of the GeoAPI interfaces, or by helping with other concerns of the project.

This standard complements existing OGC standards by defining a new, language specific layer of normalization. This standard does not replace the core standards developing the ISO/OGC abstract model but complements those documents for developers who use the Java language by documenting the mapping of types and methods from the abstract model into Java and explaining the use of the GeoAPI library. Because this standard differs in design and ambition from earlier OGC specifications which also included Java language interfaces, this document has been proposed as a new standardization effort in its own right.

— a statement specifying which annexes are normative and which are informative.

The interfaces described in this standard follow directly, without introducing any new concepts, from the previously published standards of the Open Geospatial Consortium and the International Organization for Standardization. Nonetheless, attention is drawn to the possibility that some of the elements of this document may be the subject of artificial constraints granted under patent laws by certain national governments or international institutions. Open Geospatial Consortium Inc. shall not be held responsible for identifying any or all elements potentially or actually subject to claim under such laws. However, to date, no such elements have been claimed or identified. Recipients of this document are requested to submit, with their comments, notification of any relevant patent infringements, copyright violations, or trademark abuses of which they may be aware that might be brought on by any implementation of the specification set forth in this document, and to provide supporting documentation.

## **Introduction**

The GeoAPI Implementation Standard defines the normalized use of the GeoAPI library.

The GeoAPI library contains a series of interfaces and classes in the Java language defined in several packages which interpret into Java the data model and UML types of the ISO and OGC specification documents. The library includes extensive Javadoc code documentation which complement the injunctions of the ISO/OGC specifications by explaining particularities of the GeoAPI library: interpretations made of the specifications where there was room for choice, constraints due to the library's use of Java, or standard patterns of behaviour expected by the library, notably in its handling of return types during exceptional situations.

This specification explains the GeoAPI library and defines its use by library code implementing the API and by client code calling the API. Jointly with the library itself, this work aims to provide a carefully considered interpretation of the OGC specifications for the Java language, to provide a base structure to facilitate the creation of software libraries which implementing OGC standards, and to give application developers a well defined, full documented binding reducing the programming effort of using the OGC abstract model and facilitating the portability of application code between different implementations. The interfaces defined in this standard provide one way to structure the use the Java language to implement software which follows the design and intents of the OGC/ISO specifications. The creators of the GeoAPI interfaces consider this approach as an effective compromise between the OGC specifications, the requirements of the Java language, and the tradition of the core Java libraries.

This version of the standard does not yet propose a complete set of interfaces covering the entire abstract standard of the ISO/OGC but focuses on an initial group of interfaces only. This initial group of interfaces covers enough of the abstract model to permit the definition of geospatial coordinate systems and geodetic anchoring points and to enable the conversion of coordinate tuples between different reference systems. The work writing interfaces matching other OGC specifications has already begun in the 'pending' version of the GeoAPI library. It is expected that these other interfaces will be proposed for standardization in subsequent revisions of this specification but the interfaces must first have been implemented, ideally several times, and then tested extensively by use.

# GeoAPI Implementation Specification

## 1 Scope

The GeoAPI Implementation Specification defines, through the GeoAPI library, a Java language application programming interface (API) including a set of types and methods which can be used for the manipulation of geographic information structured following the specifications adopted by the Technical Committee 211 of the International Organization for Standardization (ISO) and by the Open Geospatial Consortium (OGC). This specification standardizes the informatic contract between the client code which manipulates normalized data structures of geographic information based on the published API and the library code able both to instantiate and operate on these data structures according to the rules required by the published API and by the ISO and OGC specifications.

The normative publication of the library occurs in a Java Archive (JAR) format binary. That binary is distributed along with a ZIP format bundle of the Javadoc comments as HTML files. An online version of the Javadoc comments, which may contain fixes for errata discovered after publication of this specification, will be available at the URL <http://www.geoapi.org/3.0/javadoc/index.html> once this specification has been published. *In the interim, prior to formal acceptance, the proposed API can be seen at the URL <http://www.geoapi.org/snapshot/javadoc/index.html> although all deprecated methods will be dropped for the 3.0 version.*

Version 3.0 of the library covers the base of the OGC Abstract Model for geographic information. GeoAPI 3.0 provides utilities, base types, metadata structures, and geo-referencing data elements which enable the creation of reference systems for spatial coordinates related to the Earth and of mathematical operators to convert coordinates from one coordinate reference system to another. This version of the standard covers the specifications ISO 19103, ISO 19115, ISO 19111, some elements from the closely related OGC™ specification OGC 01-009 and four elements from ISO 19107 necessary to the implementation of ISO 19111. Future versions of this specification are expected to expand this set of interfaces to cover the full model of the OGC Abstract Specification series, including notably Coverage and Feature data structures, with the 'pending' portion of the GeoAPI project already exploring these new areas.

## 2 Conformance

This specification places no conformance constraints on client code which uses this API backed by some implementation. The Java compiler will both ensure that the client code correctly calls the methods which are invoked and ensure type safety for the objects obtained from the method call. Nonetheless, programmers of client code which uses

GeoAPI are urged to follow the best practices for use of the API which are documented in the Javadoc comments of GeoAPI as well as elsewhere, including herein.

This specification makes certain requirements of libraries implementing this API and defines several conformance classes for implementations covering different packages of the API or providing different levels of complexity in their implementations. These requirements and conformance classes are presented in Annex A (normative).

GeoAPI does not currently have any formal test suite through which to establish conformance of GeoAPI implementations. The construction of such a test suite presents several complex challenges which may be tackled over time. However, GeoAPI does include a validation framework which can be used during unit testing as explained in Annex A.

### 3 Normative references

The following normative documents contain provisions which, through reference in this text, constitute provisions of this specification, OGC 09-083, except for any departures from the listed specifications which are explicitly mentioned in this text. For dated references, subsequent amendments to, or revisions of, any of these publications do not apply. However, parties to agreements based on this specification, OGC 09-083, are encouraged to investigate the possibility of applying the most recent editions of the normative documents indicated below. For undated references, the latest edition of the normative document referred to applies.

ISO 19103, *Geographic information --- Conceptual schema language*, 2005.

ISO 19115, *Geographic information --- Metadata*, 2003.

ISO 19115, *Geographic information --- Metadata / Corrigendum 1*, 2006.

ISO 19111, *Geographic information --- Spatial referencing by coordinates*, 2007.

OGC 01-009, *OpenGIS® Implementation Specification: Coordinate Transformation Services*, revision 1.00, 2001 (partially)

*The Java Language Specification, 3<sup>rd</sup> Edition* James Gosling, Bill Joy, Guy Steele, Gilad Bracha, Sun Microsystems, 2005.

JSR-275, *JSR-275 Implementation Specification --- Measures and Units*, The Java Community Process, version DRAFT-0.8, 2007.

The normative reference towards the ISO metadata standard, *ISO 19115*, follows the lead of *ISO 19111* in excluding all references to MD\_CRS and associated types. *ISO 19111* states:

"Normative reference to ISO 19115 is restricted as follows: in this international standard, normative reference to ISO 19111 excludes the

MD\_CRS class and its components classes."

*ISO 19111:2007, section 3 "Normative References"*

Despite this statement here, this is documented as a departure from the standard in the section on the GeoAPI metadata packages below.

The normative reference to JSR-275 will likely be updated to version 1.0 now that the API has been released at that version.

## 4 Terms and definitions

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

### 4.1

#### **Application Programming Interface (API)**

A formally defined set of types and methods which establish a contract between *client* code which uses the API and *implementation* code which provides the API.

### 4.2

#### **Java**

Trademark of Sun Microsystems used to refer to an object oriented, single inheritance programming language whose syntax derives from the C programming language and which is defined by the Java Language Specification.

Rules for the drafting and presentation of terms and definitions are given in the ISO 19104 Terminology.

## 5 Conventions

The conventions in this document follow the model of the ISO 19100 series specifications and standard practice in the fields of geographic information systems and software programming.

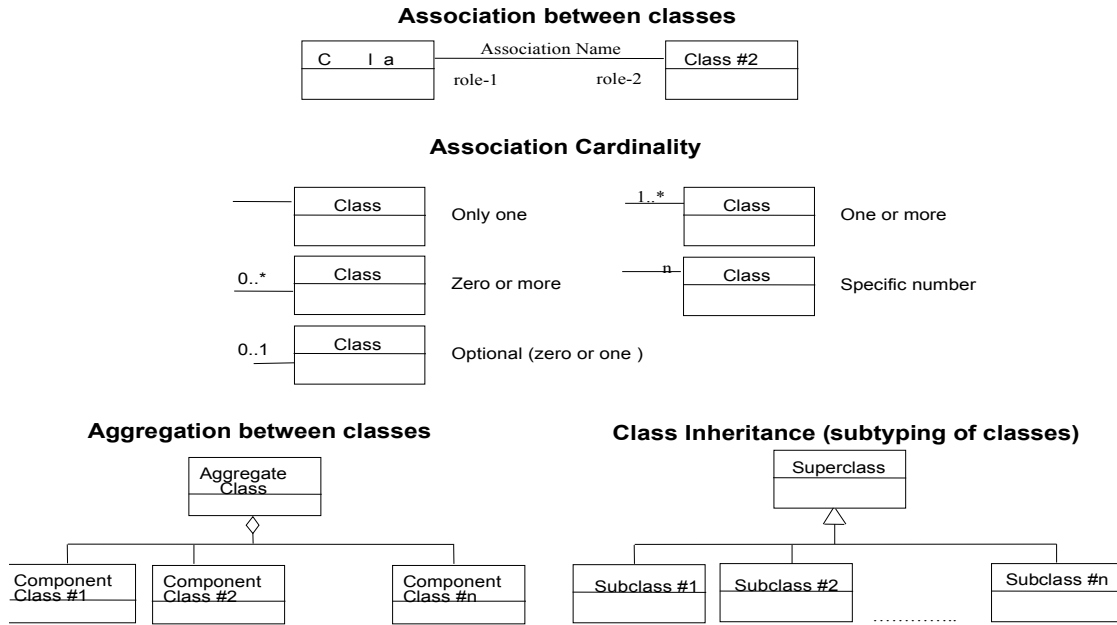
### 5.1 Symbols (and abbreviated terms)

API	Application Program Interface
ISO	International Organization for Standardization
OGC	Open Geospatial Consortium
UML	Unified Modeling Language
XML	eXtended Markup Language
1D	One Dimensional
2D	Two Dimensional

3D            Three Dimensional  
nD            Multi-Dimensional

## 5.2 UML Notation

The diagrams that appear in this standard are presented using the Unified Modeling Language (UML) static structure diagram. The UML notations used in this standard are described in the diagram below.



**Figure 1 — UML notation**

In this standard, the following three stereotypes of UML classes are used:

- <<Interface>> A definition of a set of operations that is supported by objects having this interface. An Interface class cannot contain any attributes.
- <<DataType>> A descriptor of a set of values that lack identity (independent existence and the possibility of side effects). A DataType is a class with no operations whose primary purpose is to hold the information.
- <<CodeList>> is a flexible enumeration that uses string values for expressing a list of potential values.

In this standard, the following standard data types are used:

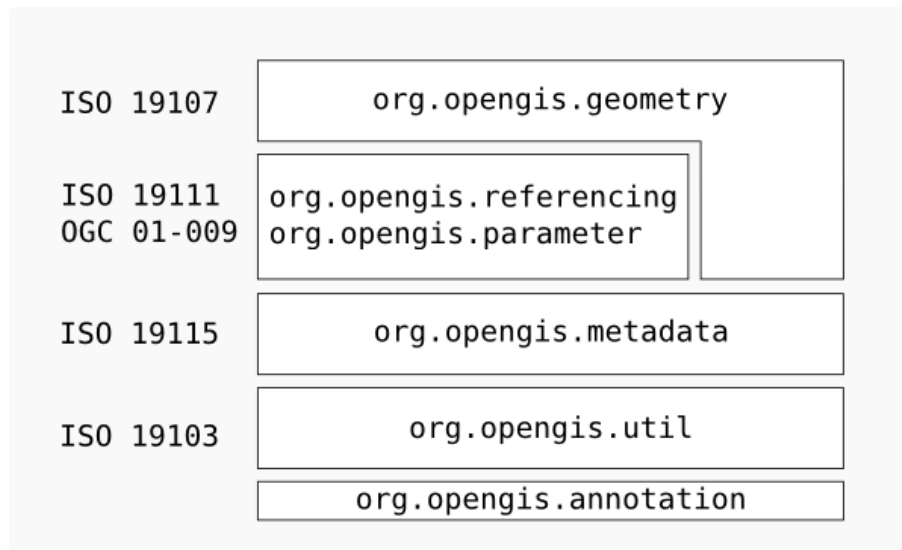
- CharacterString – A sequence of characters

- b) Integer – An integer number
- c) Double – A double precision floating point number
- d) Float – A single precision floating point number

## 6 A Geographic API in Java

The GeoAPI library formalizes the handling of the types defined in the specification documents for working with geographic information adopted by the International Organization for Standardization (ISO) and the Open Geospatial Consortium (OGC). Whereas the specifications define data types, methods and relationships using the general UML notation, the GeoAPI library implements those standards as Java language interfaces or simple classes. The GeoAPI types jointly form an application programming interface (API) which provides two groups of developers with a common point of exchange. Developers wishing to implement code which fulfils the requirements of the ISO and OGC specifications can adopt GeoAPI as a roadmap for their development. Developers wishing to write code which uses the data types defined by the standards can simply call the methods of the interfaces; they also gain a measure of independence from the particular implementation they are using since another implementation of the API can be swapped without breaking any calls made to the GeoAPI interfaces.

The structure of the GeoAPI library mirrors the packaging and separation of the different ISO and OGC specifications by grouping different types and functionality in separate Java language packages.



The library rests on the `org.opengis.annotation` package which provides the annotation system used to document the origin and obligation level of all methods and types in the library. These annotations are available through introspection at runtime for any code which wishes to exploit this information. The base of the library is formed by a formal mapping of the core types used by the ISO and OGC standards to Java equivalents along with extra types not defined in Java which are provided in the `org.opengis.util` package. The packages in the `org.opengis.metadata` namespace cover the data types defined in the ISO 19115 *Metadata* specification which are data structures holding textual references to elements describing other structures. The packages in the `org.opengis.parameter` and `org.opengis.referencing` namespaces implement the types from the ISO 19111 *Spatial Referencing by Coordinates* specification complemented by the mathematical operator types from the OGC 01-009 Implementation specification *Coordinate Transformation Services*. The packages in the `org.opengis.geometry` namespace cover the data types defined

in the ISO 19107 *Spatial Schema* specification, although in version 3.0 of the library only defines the elements from that specification needed by the geo-referencing types defined in the OGC 01-009 specification since these packages are inter-dependent.

## 7 Annotation package

The GeoAPI annotation package uses the `org.opengis.annotation` namespace and implements Java language annotations and supporting classes which enable GeoAPI to document the origin, original name, and necessity of the various types and methods integrated from the various specification documents.

All classes in GeoAPI, including interfaces and enumeration types, which are based on a published standard should have an annotation label `@UML` documenting the standard in which are defined the type or method, the original name of the element and the obligation level of the type if other than the default mandatory level of obligation. Interfaces which have been added by GeoAPI generally carry an annotation label `@Extension` although this is omitted from types, such as Java Exception classes, which do not occur in any standard and are therefore clearly GeoAPI extensions.

### 7.1 Use of the annotation types

As an example, the annotation label for the `ProjectedCRS` interface appears in the source code as:

```
@UML(identifier="SC_ProjectedCRS",
      specification=ISO_19111)
```

which specifies that the type was defined in ISO 19111 standard, in the SC 'Coordinate Reference System' package as the type `"GeographicCRS"` while the method `getCoordinateSystem()` of that class has the annotation:

```
@UML(identifier="coordinateSystem",
      obligation=MANDATORY,
      specification=ISO_19111)
```

which indicates that the method was defined in the same *ISO* 19111 specification but had the name `"coordinateSystem"` in the standard rather than the `"getCoordinateSystem"` name used by GeoAPI and that the method must be present in every `ProjectedCRS` instance.

These annotations are available at runtime by Java introspection. This is useful, for example, when code needs to serialize data types differently based on the package in which they were defined in the ISO standard. At runtime, the annotation of a reference to a GeoAPI interface can be obtained as follows, taking as an example the method `getTitle()` in the `Citation` type:

```
Class<?> type = Citation.class;
Method method = type.getMethod("getTitle", (Class[]) null);
UML annot = method.getAnnotation(UML.class);
String ident = annot.identifier();
Specification spec = annot.specification();
Obligation obl = annot.obligation();
```

Java provides a class instance like the `'Citation.class'` instance used here for every type, either interface or class, defined in the runtime and these references are 'static,' that is they are always available. The `'getMethod(.)'` call uses introspection to obtain a reference

to the method from which the annotation can then be obtained. The annotation system therefore provides access, at runtime, to the original definition of the element.

## 8 Utility package

The GeoAPI utility package uses the `org.opengis.util` namespace and implements the types which are defined in the specification from the International Organization for Standardization *ISO 19103:2005 Geographic Information – Conceptual schema language* but are not already present in the Java language itself or in the standard Java library.

The utility package of GeoAPI completes the GeoAPI type mapping from the UML types used by the 19100 series of ISO standards into Java types by providing the elements missing from the Java language or standard library. The ISO 19103 specification defines types and utilities which are used as building blocks by the other standards in the 19100 series. GeoAPI maps these types either to existing types from the Java language and library or, when needed, to types defined in the utility package. For various practical reasons the mapping is not a one-to-one mathematical function. ISO 19103:2005 defines Primitive types (§6.5.2, of that standard), Collection or Dictionary types (§6.5.3), Enumerated types (§6.5.4), Representational types (§6.5.5), Name types (§6.5.6), and Derived types (§6.5.7). The mapping actually used is explained below. The utility package also includes the extra type `InternationalString` to handle textual sequences which might need to be represented in multiple languages and a factory for the Name constructs which provides a single point of instantiation.

The Java types mapped by GeoAPI or provided in the utility package can be used like regular Java language elements. Most of the types can be instantiated directly through public constructors. Enumeration types provide public access to each of their constants. `CodeList` types provide the static `valueOf(.)` method through which instances can be obtained. The `NameFactory` interface provides public methods for the instantiation of the various Name types. GeoAPI does not specify any extra constraints on the behaviour or use of these types.

### 8.1 Package Mapping

GeoAPI maps the types of ISO 19103 into equivalents from the Java language and library or into types defined in the utility package. However, not all of the types in ISO 19103 have had a mapping defined because the need for these types has not yet appeared since they have not yet appeared in any other specification for which GeoAPI defines interfaces. Such types are listed as 'unimplemented' in the tables below.

**TODO:** We need a note comparing this mapping to the mapping and prescriptions of ISO 11404:2007.

#### 8.1.1 Primitive Types

The Primitive types of the ISO/OGC specifications map to single object structures in GeoAPI. Where the mapping can be made directly to a Java primitive type, such as int and double, the Java primitive is preferred; however, when the value must be able to be set to null, the object wrapper of that primitive is used.

The following table shows the mapping used by GeoAPI to represent the types in the ISO 19100 series.

<b>Primitive Types Mapping</b>		
<b>Type Group</b>	<b>ISO 19103 Type</b>	<b>GeoAPI Type</b>
<b>Numeric</b>	Integer	int / java.lang.Integer long / java.lang.Long
	UnlimitedInteger	<i>unimplemented</i>
	Real	double / java.lang.Double
	Decimal	java.math.BigDecimal
	Number	java.lang.Number
	Vector	<i>unimplemented</i>
<b>Text</b>	CharacterString	java.lang.String org.opengis.util.InternationalString
	Sequence<Character>	java.lang.CharSequence
	Character	char
	CharacterSetCode	org.opengis.metadata.identification.CharacterSet
	LanguageCharacterString	<i>unimplemented</i>
<b>Date and Time</b>	Date	java.util.Date
	Time	java.util.Date
	DateTime	java.util.Date
	DatePrecision	<i>unimplemented</i>
<b>Truth</b>	Probability	<i>unimplemented</i>
	Boolean	boolean / java.lang.Boolean
	Logical	<i>unimplemented</i>
	Truth	<i>unimplemented</i>
	DiscreteTruth	<i>unimplemented</i>
	ContinuousTruth	<i>unimplemented</i>
<b>Multiplicities</b>	Multiplicity	<i>unimplemented</i>
	MultiplicityRange	<i>unimplemented</i>
<b>Enumerations</b>	Sign	<i>unimplemented</i>
	Digit	<i>unimplemented</i>
	Bit	<i>unimplemented</i>

Several of the objects in ISO 19103 have not been implemented since they have not yet been needed during the development of the rest of the interfaces. GeoAPI will consider implementing these types when they become necessary for the implementation of other elements in the ISO and OGC standards.

The class InternationalString is an extension used by GeoAPI to handle Java String objects which may potentially need to be translated for users of different locales. Conceptually

this acts as a String but may, depending on the implementation, provide access to locale specific representations of that String.

### 8.1.2 Collection and dictionary types

GeoAPI implements ISO 19103 collection types using the standard Java Collections Framework. The one major difference is that GeoAPI collections do not implement the TransfiniteSet interface.

Collection and Dictionary Types Mapping	
ISO 19103 Type	GeoAPI Type
Transfinite Set	<i>unimplemented</i>
Collection	java.util.Collection
Set	java.util.Set
Bag	java.util.Collection
Sequence	java.util.List
CircularSequence	<i>unimplemented</i>
Dictionary	java.util.Map
KeyValuePair	java.util.Map.Entry

These collection types are used within GeoAPI qualified with a parametric type, which does not quite follow strictly the template notion which these types have in the ISO standards but is the closest one can conveniently do in the Java language.

**NB** In future revisions of this document which include the geometry interfaces, the TransfiniteSet interface will be added as `org.opengis.geometry.TransfiniteSet`.

### 8.1.3 Enumerated types

GeoAPI distinguishes between two enumerated types depending on whether the complete set of literal types is known when the code is originally created or if the list may be extended at run time or when the code is extended. The Java language provides the Enum language construct for the former case and GeoAPI defines the CodeList interface for the latter case.

Enumerated Types Mapping	
ISO 19103 Type	GeoAPI Type
Enumeration	java.lang.Enum
CodeList	org.opengis.util.CodeList

### 8.1.4 Representation types

These types along with the name types have caused the GeoAPI project some confusion and we are unsure how these interfaces would be used and therefore how they should be implemented. GeoAPI currently defines only a strict minimum of the representation types in order to cover those necessary for the coverage package implementing the types in ISO 19123. It may well be that our understanding of these types is incorrect or incomplete and that we need to change them in future.

<b>Representation Types Mapping</b>	
<b>ISO 19103 Type</b>	<b>GeoAPI Type</b>
Schema	<i>unimplemented</i>
Type	<i>unimplemented</i>
Any	java.lang.Object
RecordSchema	org.opengis.util.RecordSchema
RecordType	org.opengis.util.RecordType
Record	org.opengis.util.Record

There is also no formal factory for representation types and this will have to be considered in future revisions of the standard.

### 8.1.5 Name types

The name types in ISO 19103 have few documentation. At least four of us worked on this issue and have each reached slightly different conclusions. For now, the authoritative explanation for how we interpret this Name system is in the Javadoc for GenericName:

<http://www.geoapi.org/snapshot/javadoc/org/opengis/util/GenericName.html>

which explains our current interpretation of scopes and namespaces. If our understanding is erroneous, we may be forced to change these interfaces once again in the future.

<b>Name Types Mapping</b>	
<b>ISO 19103 Type</b>	<b>GeoAPI Type</b>
NameFactory	org.opengis.util.NameFactory
NameSpace	org.opengis.util.NameSpace
GenericName	org.opengis.util.GenericName
ScopedName	org.opengis.util.ScopedName
LocalName	org.opengis.util.LocalName
TypeName	org.opengis.util.TypeName
MemberName	org.opengis.util.MemberName

The NameFactory is an extension of the GeoAPI project designed to facilitate the construction of instances of these Name types.

### 8.1.6 Derived types

The derived types from ISO 19103 are almost all related to units and measurements. GeoAPI relies for these types on the interfaces defined by the external project the Java Specification Request number 275, *JSR-275 Units Specification*. The JScience project (<http://jscience.org/>) provides the reference implementation of these interfaces.

The JSR-275 interfaces rely extensively on parametrized types to qualify the type of Unit or Measure being used.

<b>Derived Types Mapping</b>	
<b>ISO 19103 Type</b>	<b>GeoAPI Type</b>
Measure	<code>javax.measure.Measure&lt;? extends Quantity&gt;</code>
UnitOfMeasure	<code>javax.measure.unit.Unit&lt;? extends Quantity&gt;</code>
Area	<code>javax.measure.Measure&lt;Area&gt;</code>
UomArea	<code>javax.measure.unit.Unit&lt;Area&gt;</code>
Length	<code>javax.measure.Measure&lt;Length&gt;</code>
Distance	<code>javax.measure.Measure&lt;Length&gt;</code>
UomLength	<code>javax.measure.unit.Unit&lt;Length&gt;</code>
Angle	<code>javax.measure.Measure&lt;Angle&gt;</code>
UomAngle	<code>javax.measure.unit.Unit&lt;Angle&gt;</code>
Scale	<code>javax.measure.Measure&lt;Dimensionless&gt;</code>
UomScale	<code>javax.measure.unit.Unit&lt;Dimensionless&gt;</code>
Time	<code>javax.measure.Measure&lt;Time&gt;</code>
UomTime	<code>javax.measure.unit.Unit&lt;Time&gt;</code>
Volume	<code>javax.measure.Measure&lt;Volume&gt;</code>
UomVolume	<code>javax.measure.unit.Unit&lt;Volume&gt;</code>
Velocity	<code>javax.measure.Measure&lt;Velocity&gt;</code>
UomVelocity	<code>javax.measure.unit.Unit&lt;Velocity&gt;</code>
AngularVelocity	<code>javax.measure.Measure&lt;AngularVelocity&gt;</code>
UomAngularVelocity	<code>javax.measure.unit.Unit&lt;AngularVelocity&gt;</code>
NULL	<code>null</code>
EMPTY	<code>java.util.Collections.EMPTY_SET</code>

GeoAPI uses the Java language keyword `null` to represent the ISO NULL value and the empty set from the Java Collections Framework for the ISO EMPTY. Note that programmers, for type safety when using Java Generics, should call the method `java.util.Collections.emptySet()` rather than refer directly to the constant, since the former will have the parametric type at compile time.

### 8.2 Use of the utility types

Use of the types in the GeoAPI utility package follows directly standard practice in Java.

The `org.opengis.util.InternationalString` interface provides a container for multiple versions of the same text, each for a specific `Locale` – the identifier used in Java for a specific language, possibly in a named territory.

```
NameFactory factory = ...{Implementation dependent}
Map<Locale,String> names = new HashMap<Locale,String>();
names.put(Locale.ENGLISH, "My documents");
names.put(Locale.FRENCH, "Mes documents");
InternationalString localized = factory.createInternationalString(names);
System.out.println( localized.toString() );
System.out.println( localized.toString(Locale.FRENCH));
```

The method to obtain factories is not specified by this standard and therefore depends on the design of the library implementation. Also, the locale used by default depends on the choice of the implementation so the result of the call `toString()` without parameters will depend on the implementation.

The use of `org.opengis.util.CodeList` constructs includes accessing statically defined elements, defining new elements and retrieving any element defined for the code list. Considering, for example, `org.opengis.metadata.distribution.MediumName` used to specify the kinds of physical media on which a data set could be distributed, the following code could be used

```
MediumName cd = MediumName.CD_ROM;
MediumName usbkey = MediumName.valueOf("USB_KEY");
```

where the second locution will create a new value if it does not exist. Special care should be taken to keep such calls consistent throughout the code since the `CodeList` will create a new element if there are any differences between the `String` parameters: for example, the call

```
MediumName med = MediumName.valueOf("CDROM");
```

would return a new value rather than the static default.

The use of `javax.measure.unit.Unit` from JSR-275 and associated types is explained at length in the specification document JSR-275: Units and Measures. Here, only a trivial example is presented:

```
Unit<Length> sourceUnit = NonSI.MILE;
Unit<Length> targetUnit = SI.KILO(SI.METRE);
UnitConverter converter = source.getConverterTo(target);
double source = 123.2;
double target = converter.convert(source);
```

where the initial calls define units of length and then a converter is used to obtain the equivalent length in a new unit.

### 8.3 Departure from ISO 19103

GeoAPI differs from *ISO 19103* in not providing all of the types defined in the standard. The elements that have not been defined have not yet been encountered in subsequent standards implemented by GeoAPI.

The `InternationalString` type provided by the utility package extends the basic `String` type provided by Java for internationalization by enabling the object to hold a separate `String` for every locale it wishes to handle.

The NameFactory type provided by the utility package complements the Name types defined by ISO 19103 by providing a formalized approach to instantiating the objects.

The Collections provided by GeoAPI are the standard Java collections and therefore do not extend TransfiniteSet as required by the *ISO 19103* specification. However, the concept of TransfiniteSet applies most naturally to geometric constructs rather than to sets more generally.

#### **8.4** Future improvements

There are several improvements related to the GeoAPI utility package that are to be expected in future revisions of this standard. The Name system may need another revision since it has proved to be a very difficult system to interpret correctly. Similarly, the Record system remains unclear and may need revision. The mapping of elements to Date might eventually evolve since the Java standard library is gaining its third implementation of data types designed to hold calendar based temporal references; if the new constructs replace the old with much more convenient functionality it might be worth moving to the new constructs in some future revision

## 9 Metadata packages

The GeoAPI metadata packages use the `org.opengis.metadata` namespace and implement the types defined in the specification from the International Organization for Standardization *ISO 19115:2003 Geographic Information – Metadata* along with the modifications of *Technical Corrigendum 1* from 2006.

The metadata packages of GeoAPI provide container types for descriptive elements which may be related to data sets or components. All of these data structures are essentially containers for strings, and the interfaces consist almost exclusively of methods which provide access to the strings or a container. The API defines no methods which manipulate or modify the data structures.

The metadata packages of GeoAPI have been built primarily in support of the geodetic types defined in the referencing packages and therefore consider primarily read access to the data structure contents. The GeoAPI metadata interfaces provide no methods to set the values of the types. Furthermore, because the way that wildcards for Java Generics have been used in the interfaces, the metadata instances are constrained to be read only and it is not possible to implement a fully mutable implementation of GeoAPI using these interfaces.

The GeoAPI rules of method return values have been changed for the metadata packages. Elsewhere in GeoAPI, methods which have a mandatory obligation in the specification must return an instance of the return type and cannot return the Java null reference. However, in the metadata package this rule is relaxed because data sets are encountered so frequently which have not correctly followed the requirements of the specification. In the GeoAPI metadata packages, all methods are considered to have an Optional obligation and must follow the rules for that obligation level. This means that metadata methods **MUST** return the object if present or otherwise either return null or return the empty collection, if the method return type is a Java Collection. This injunction means that implementations must catch any exception they may encounter while satisfying the method call. This modification has been adopted to allow implementations sufficient latitude to handle metadata records which do not correctly conform to the specification. Nonetheless, sophisticated implementations can determine if a metadata record conforms with the specification by inspecting the annotation at runtime.

### 9.1 Package mapping

The mapping of ISO 19115 packages to GeoAPI packages follows an almost perfectly parallel naming scheme.

<b>Metadata Package Mapping</b>	
<b>ISO 19115 Package</b>	<b>GeoAPI Package</b>
Metadata entity set information	<code>org.opengis.metadata</code>
Identification information	<code>org.opengis.metadata.identification</code>
Constraint information	<code>org.opengis.metadata.constraint</code>
Data quality information	<code>org.opengis.metadata.quality</code>

	org.opengis.metadata.lineage
Maintenance information	org.opengis.metadata.maintenance
Spatial representation information	org.opengis.metadata.spatial
Reference system information	org.opengis.referencing.* org.opengis.parameter (see below)
Content information	org.opengis.metadata.content
Portrayal catalogue reference	org.opengis.metadata
Distribution information	org.opengis.metadata.distribution
Metadata extension information	org.opengis.metadata
Application schema information	org.opengis.metadata
Extent information	org.opengis.metadata.extent
Citation and responsible party information	org.opengis.metadata.citation

Several minor packages have been aggregated into the top level package. The Data quality information package has been split into two packages to separate the DQ\_\* types from the LI\_\* types. As explained next, the Reference system information has been replaced by the types from the referencing package.

## 9.2 Use of the GeoAPI metadata packages

The types in the GeoAPI metadata packages are primarily containers of Java String types and have been designed around providing read access to those elements. Metadata elements will be encountered in the data types from the referencing packages and the interfaces enable users to obtain the elements of the data type.

As an example, we want to print a list of all the authors for a document starting with an org.opengis.metadata.citation.Citation element.

```
//We assume this instance is already available
Citation citation = ...;

for (ResponsibleParty rp : citation.getCitedResponsibleParties()){
    if (rp.getRole() == Role.AUTHOR) {
        String author = rp.getIndividualName();
        System.out.println(author);
    }
}
```

The remainder of the metadata packages work in similar ways, where client code must disaggregate an instance to obtain the elements needed.

## 9.3 Departures from standard

The major departure in the GeoAPI metadata packages from the published *ISO 19115* standard come from GeoAPI following the *ISO 19111* standard and replacing the MD\_CRS type from *ISO 19115* with the types in *ISO 19111*. The types from ISO 19111 duplicate the classes present in the metadata specification but with richer, more complete

semantics. GeoAPI does not implement the following classes but substitutes a suitable replacement from the referencing packages.

<b>Mapping of types from the reference system information package</b>	
<b>ISO 19115 type</b>	<b>GeoAPI replacement</b>
MD_ReferenceSystem	org.opengis.referencing.ReferenceSystem
MD_CRS	org.opengis.referencing.crs.CoordinateReferenceSystem
MD_EllipsoidParameters	org.opengis.referencing.datum.Ellipsoid
MD_ProjectionParameters	org.opengis.parameter.ParameterValueGroup
MD_ObliqueLineAzimuth	org.opengis.parameter.ParameterValue
MD_ObliqueLinePoint	org.opengis.parameter.ParameterValue

Note however, that the parameter package of GeoAPI and ISO 19111 is more generic than the explicit types defined in ISO 19115, handling referencing constructs in a map like structure rather than as individual, named data types.

#### 9.4 Future work

Future revisions of the GeoAPI metadata package will add the types and methods defined in the specification *ISO 19115-2 Geographic Information – Metadata – Part 2: Extensions for imagery and gridded data* which was forced to create a number of dummy types to hold elements which naturally could occur directly in the types defined by *ISO 19115*. We expect to integrate such types directly into the existing types rather than adding complexity to the API which exists by historical accident.

Future revisions of these packages may also add factory interfaces through which these types could be instantiated. However, the actual design for such a factory system has not yet been agreed upon by the contributors to GeoAPI.

## 10 Geometry packages

The GeoAPI geometry packages use the `org.opengis.geometry` namespace and implement the types defined in the specification from the International Organization for Standardization *ISO 19107:2003 Geographic Information - Spatial schema*.

The geometry packages of GeoAPI provide spatial types combining coordinates with the reference system used for those coordinates. These types implement a vector based spatial representation of elements. **The geometry packages also include a sophisticated containership hierarchy, objects which know of their boundary, and topological data structures.**

The geometry types defined in this standard include only the two simplest types in the specification along with their abstract parent interface. It is expected that the two concrete types will be instantiated through public constructors.

### 10.1 Defined types

GeoAPI defines a minimal set of four types from the *ISO 19107 Geographic Information - Spatial schema* specification, `DirectPosition`, `Position`, `Envelope`, and `MismatchedDimensionException`, because these types are needed by the referencing package.

Mapping of types from the Coordinate geometry package	
ISO 19107 type	GeoAPI replacement
GM_Position	<code>org.opengis.geometry.coordinate.Position</code>
DirectPosition	<code>org.opengis.geometry.DirectPostion</code>
GM_Envelope	<code>org.opengis.geometry.Envelope</code>

The `DirectPosition` type represents a single location in the anchored coordinate space defined by a `CoordinateReferenceSystem`. Since `DirectPosition` extends the `Position` type that interface was needed as well.

The `Envelope` type represents the lower and upper extreme values along each axis. The type is frequently conflated with a bounding rectilinear box but the two elements differ conceptually in subtle ways. For example, the bounding box of Siberia crosses the anti-meridian and runs from around 60 degrees east of Greenwich to 170 degrees west whereas the `Envelope` for Siberia goes from -180 degrees longitude to 180 degrees longitude. A further possible confusion arises because the `Envelope` type in *ISO 19107* provides methods to obtain the 'corners' of the `Envelope` as `DirectPositions`. However, users should note that these `DirectPositions` might not have any particular meaning individually in their `CoordinateReferenceSystem`; they are acting, for convenience, as data containers for a tuple of ordinates but not as representations of an actual `Position` so the ordinates of the tuple must be considered independent.

GeoAPI also defines a `MismatchedDimensionException` Java Throwable. This type can be used for method calls whose parameters might be nonsensical if they do not share the same, or have the correct, dimension.

## 10.2 Use of the geometry packages

The usage of the data types in the geometry package of GeoAPI follow the standard rules of Java and do not warrant extended explanation here.

**N.B.** This section exists mostly as a placeholder for future revisions.

## 10.2 Departure from Standards

GeoAPI has moved the `DirectPosition` and `Envelope` types from the `geometry.coordinate` package where they are defined in the ISO 19107 specification up to the `org.opengis.geometry` package due to their importance and frequency of use. Conceptually, the ISO 19107 standard considers geometric objects to be collections of `DirectPositions` so that data structure is used throughout the API.

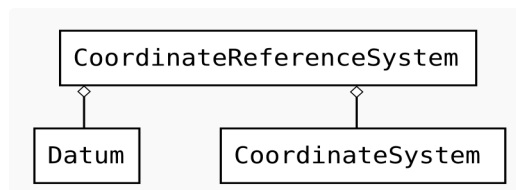
## 10.3 Future work

Future versions of this specification are expected to present a much larger set of interfaces for the types from ISO 19107. For now, the interfaces defined by the GeoAPI project remain experimental with no functional reference implementation.

## 11 Referencing and Parameter packages

The GeoAPI referencing packages and parameter package use the `org.opengis.referencing` and `org.opengis.parameter` namespaces respectively and implement the types defined in the standard from the International Organization for Standardization *ISO 19111:2007 Geographic Information - Spatial referencing by coordinates*. The referencing package also includes the types describing object factories and mathematical transformation operators between reference frames defined in the standard from the Open Geospatial Consortium *OGC 01-009 OpenGIS Implementation Specification: Coordinate Transformation Services* from 2003.

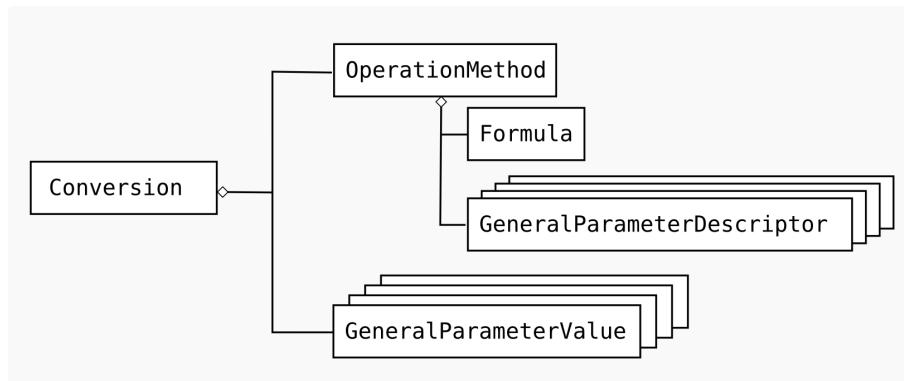
The referencing and parameter packages of GeoAPI provide data constructs and operations for geospatial referencing and coordinate operations.



**Figure 11.1 Components of a CRS**

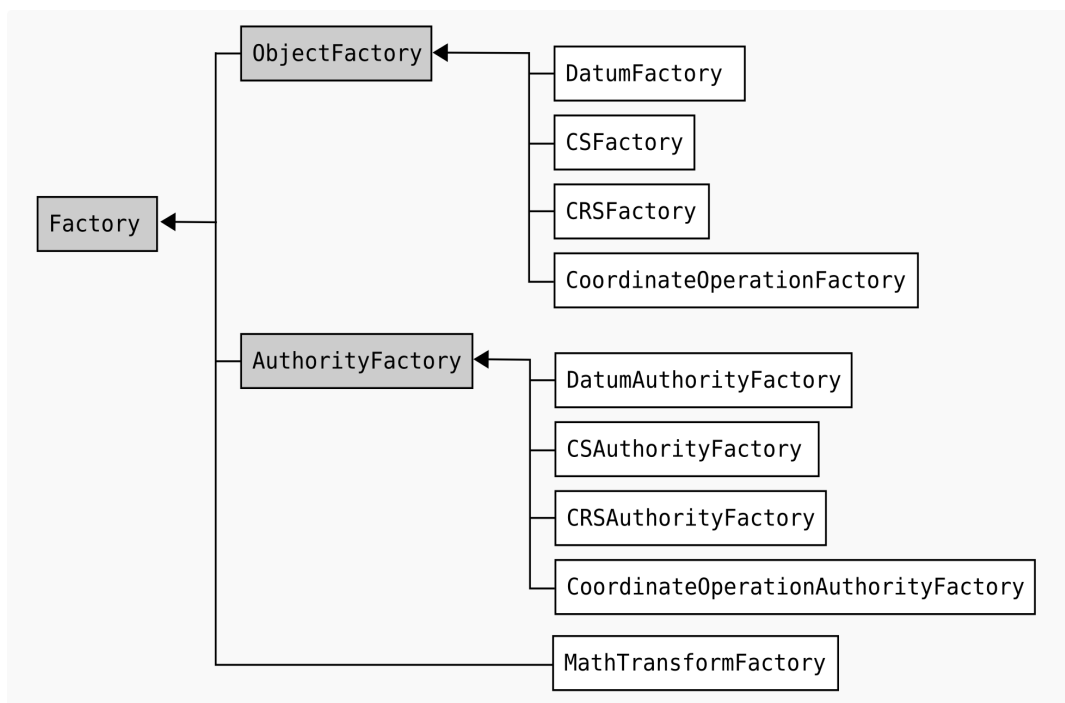
(after Fig.2, *ISO 19111:2007*)

The referencing package types can be used to define geospatial referencing constructs based on the *ISO 19111* specification which can be used to define various engineering and geodetic datums, define various coordinate systems, and combine those to define all the coordinate referencing systems (CRS) generally encountered in geospatial science.



**Figure 11.3 Components of a Mercator projection**

The referencing and parameter packages also provide types from the *ISO 19111* standard through which to define operations, and their parameters, for the calculation of coordinates in a destination CRS based on coordinates given in some source CRS. Additionally, GeoAPI adds the `MathTransform` operator from the *OGC 01-009* standard which can actually perform the calculation.



**Figure 11.3 Referencing Factories**

Finally, the referencing packages include factory types also defined originally in the *OGC 01-009* specification. These object factories define a normalized approach to object instantiation and come in two forms, the *ObjectFactories* which instantiate objects by assembling types passed as arguments and the *AuthorityFactories* which instantiate objects based on the values of some third party database, notably those in the EPSG SQL database of referencing objects assembled by the Surveying & Positioning Committee of the International Association of Oil & Gas producers.

The use of the types defined in the GeoAPI referencing and parameter packages follows the general usage pattern of the library. Since these packages provide factories, so code that needs to instantiate one of the objects defined in these packages should first obtain a reference to the factory in some implementation dependent manner and then use the factory methods to instantiate the desired object instances. These instances can then be used through the interface defined in the GeoAPI library. The only unusual pattern in these packages arises because the *ParameterValue* types provide methods to set the value of the type. In the general use pattern for these types, a *ParameterValueGroup* containing all the named parameters for a method of an operation is first obtained from the *MathTransformFactory* and then each *ParameterValue* type is obtained in turn and its value set. This use pattern ensures that all the needed parameters for an operation method can be obtained as a single block.

## 11.1 Package Mapping

The mapping of ISO 19111 packages to GeoAPI packages follows an almost perfectly parallel naming scheme while the OGC 01-009 packages map to GeoAPI less linearly because the factory system of the OGC standard provides factory types in each GeoAPI package.

Referencing and Parameter Package Mapping	
ISO 19111 (OGC 01-009) Package	GeoAPI Package
IO Identified Object	org.opengis.referencing
RS Reference System	org.opengis.referencing
SC Coordinate Reference System	org.opengis.referencing.crs
CS Coordinate System	org.opengis.referencing.cs
CD Datum	org.opengis.referencing.datum
CC Coordinate Operation	org.opengis.referencing.operation org.opengis.parameter
CS Co-ordinate Systems (OGC 01-009)	org.opengis.referencing org.opengis.referencing.crs org.opengis.referencing.datum
CT Co-ordinate Transformations (OGC 01-009)	org.opengis.referencing.operation
PT Positioning (OGC 01-009)	org.opengis.referencing.operation

Nonetheless, the mapping is fairly straight forward. It should be noted, as was discussed in the section on Metadata, that several types from the ISO 19115 specification also map into the GeoAPI referencing packages.

## 11.2 Use of the referencing and parameter types

The following examples illustrate the use of the referencing and parameter packages of GeoAPI.

### 11.2.2 Creating a Projected Coordinate Reference System

A Coordinate Reference System can be constructed on its own or can be derived from other systems. This example shows how to build a ProjectedCRS based on the Mercator projection. Here we use an Authority which has already defined the method for this projection and then set the parameters to desired values before creating the CRS.

```
//Obtaining factory instances is implementation dependent
CRSFactory crsFactory = ...;
CoordinateOperationFactory opFactory = ...;
CoordinateOperationAuthorityFactory af = ...;

//We assume these instances are already available (used at end)
GeographicCRS baseGeographicCRS = ...;
CartesianCS cartesianCS = ...;

// Get the parameters initialized to their default values
OperationMethod method = af.createOperationMethod("Mercator (1SP)");
ParameterValueGroup pg = method.getParameters().createValue();

// Set the parameter values
pg.parameter("semi-major axis").setValue(6377397.155);
pg.parameter("semi-minor axis").setValue(6377397.155 * (1 - 1/299.15281));
pg.parameter("Latitude of natural origin").setValue(0.0);
pg.parameter("Longitude of natural origin").setValue(110.0);
pg.parameter("Scale factor at natural origin").setValue(0.997);
```

```

pg.parameter("False easting").setValue(3900000.0);
pg.parameter("False northing").setValue(900000.0);

// Create the defining conversion
Map<String, Object> properties = new HashMap<String, Object>();
properties.put(Conversion.NAME_KEY, "Makassar / NEIEZ");
Conversion def = opFactory.createDefiningConversion(properties, method, pg);

// Create the projected CRS
properties.clear();
properties.put(Conversion.NAME_KEY, "Makassar / NEIEZ");
ProjectedCRS projectedCRS = crsFactory.createProjectedCRS( properties,
                                                         baseGeographicCRS,
                                                         def,
                                                         cartesianCS);

```

This gives us a ProjectedCRS with the appropriate parameters for our needs.

### 11.2.2 Build an operation

In this usage example we build an operation using a sophisticated factory.

```

//Obtaining factory instances is implementation dependent
CoordinateOperationFactory opFactory = ...;

//We assume these instances are already available (taken from above)
CoordinateReferenceSystem sourceCRS = baseGeographicCRS;
CoordinateReferenceSystem targetCRS = projectedCRS;

CoordinateOperation op = opFactory.createOperation(sourceCRS, targetCRS);

```

The factory has done all the work of establishing which parameters should be used and correctly instantiating the operation.

### 11.2.3 Convert a coordinate between coordinate reference systems.

In this example, we use the operation we just created to calculate the coordinates in a destination coordinate reference system equivalent to the coordinates in a source coordinate reference system.

```

//We assume these instances are already available
CoordinateOperation op = ...;
double[] sourceOrdinates = ...;

//create the destination array
double[] targetOrdinates = new double[sourceOrdinates.length];

MathTransform mt = op.getMathTransform();
mt.transform(sourceOrdinates, 0, targetOrdinates, 0, sourceOrdinates.length)

```

with the user needing to guarantee that the length of the ordinate arrays are the same integer multiple of the number of dimensions in their respective coordinate reference systems.

### 11.3 Departure from Standards

The major departure of GeoAPI from the ISO 19111 standard comes from the inclusion, directly in the `CoordinateOperation` type, of a method providing access to the `MathTransform` construct from the older OGC specification. This departure fundamentally alters the function of these packages: under the ISO 19111 standard the classes only describe coordinate reference systems and the operations which convert between them, under GeoAPI the classes also provide an object which can actually calculate the coordinates in a destination CRS equivalent to given coordinates in a source CRS. For reasons of informatic efficiency and compactness, the method providing access to the `MathTransform` has been directly integrated into the `CoordinateOperation` interface so that users can obtain the mathematical object directly from the object that defines the operation. GeoAPI further departs in defining its own 1D and 2D `MathTransforms`, for speed, convenience and interoperability with the Java2D graphics library.

The second major departure of GeoAPI from the ISO 19111 standard comes from the addition of the factory system defined in the OGC 01-009 standard. This departure adds two factory hierarchies, a default factory hierarchy in which new instances are obtained by providing the content as parameters to the method calls and an 'authority' factory hierarchy in which instances are obtained based on some code identifier of the object desired specific to the particular authority supported by the factory instance. The factories provide a common basis for object instantiation and, if used exclusively, simplify the work of switching between implementations. The interfaces describe two type hierarchies for factory types: the hierarchy rooted in the `ObjectFactory` type all instantiate objects by given the necessary content elements whereas the factories rooted in `AuthorityFactory` instantiate objects based on some identification code and some data source mapping the code to object contents. GeoAPI focuses especially on the few authority codes provided by the OGC in the CRS and AUTO namespaces and the authority codes provided by the EPSG database of the Surveying & Positioning Committee of the International Association of Oil & Gas producers (OGP).

One minor departure from the ISO 19111 specification comes from GeoAPI defining an `Ellipsoidal VerticalDatumType`. The ISO specification does not allow distances above an ellipsoid independent of the longitude and latitude coordinates in order to prevent users from misusing the vertical ordinate during conversion. However, this separation is not inherently incorrect, but merely dangerous, and is necessary to handle older constructs such as the coordinate reference systems defined in the Well-Known Text textual format. GeoAPI has therefore elected to integrate this vertical datum type.

### 11.4 Future work

The referencing and parameter packages are not expected to change fundamentally in subsequent revisions of this standard. The only changes which might arise would come from unforeseen conflicts during the integration of the temporal types from the *ISO 19108 Geographic Information - Temporal Schema* standard which defines its own `TemporalReferenceSystem` and `TemporalCoordinateSystem` which are expected to be dropped in favour of the types already defined in the referencing packages.

## **Annex A**

(normative)

### **Conformance**

Libraries implementing GeoAPI are enjoined to follow certain requirements to claim conformance with this standard. The standard does permit implementations with different levels of coverage of the library by providing, below, a number of conformance classes for implementation libraries.

#### **A.1 Fundamental requirements**

All implementing libraries must follow the requirements made in this clause.

Implementing libraries must satisfy all paragraphs in this standard and in the library Javadoc that use the keywords "required," "shall," "shall not," or "must."

Java libraries which provide code implementations of the GeoAPI interfaces and which wish to claim conformance with this standard shall follow the dictates both of the Javadoc comments in the API and of the language of the OGC specifications which define each Java method.

Conformant libraries shall respect the following general pattern for method return values unless countermanded by the Javadoc code documentation for a particular method. Methods which generate new instances, such as Factory methods or Operations, are expected either to return the desired value or to throw a checked exception such as a `FactoryException` or a `TransformException`. 'Setter' methods, methods which set the value of an object, are expected either to succeed or to throw an `UnsupportedOperationException` if the method is either not implemented or illegal in that implementation. 'Getter' methods, methods which obtain a value from an object, are documented through annotations to the Javadoc as mandatory or optional. Mandatory 'getter' methods are expected to return the requested value unless the value is missing in which case they shall throw the runtime exception, `IllegalStateException`. (An exception is made to this rule in the metadata packages because of the extensive existence of incomplete metadata. In those packages, all methods are treated as optional.) Optional 'getter' methods are expected to return the requested value unless the value is missing or the method is not implemented in which case they shall return `null`. Exceptions to these general rules occur occasionally but are documented in the Javadoc comments.

All the instances of GeoAPI interfaces which are generated by a conformant library shall be valid according to the test validator, whenever a validator exists for the instance type. This does not require that all instances be tested but merely that if the instances were tested, they would validate.

## A.2 Conformance levels.

This standard provides several levels of conformance for libraries that wish to claim conformance with this standard.

All implementations must necessarily provide a fully functional implementation of the base types required by the library. This means that all implementing libraries must provide a fully working implementation of the JSR-275 standard, possibly by including the reference implementation directly. All implementing libraries must also provide functional implementations of the types defined in the `org.opengis.util` package.

### A.2.1 Conformance Level **M** – Metadata

The first level of conformance, **M**, requires the implementing library to provide a functional implementation of all the types defined in the `org.opengis.metadata` packages.

### A.2.2 Conformance Level **R-A** – Referencing Base

Libraries implementing the types defined in the `org.opengis.referencing` and `org.opengis.parameter` packages can reach several different levels of conformance depending on the coverage and complexity of their implementation.

The simplest conformant status for the Referencing level, Status **R-A1** provides code, including the `ObjectFactory` types, which can instantiate all the objects in the `org.opengis.referencing.datum`, `org.opengis.referencing.cs`, and `org.opengis.referencing.crs` packages but may be limited to the creation of coordinate referencing systems which are not compound.

The next status for this level, Status **R-A2** provides the types in level R-A1 but includes all the types necessary for compound coordinate reference systems. At this conformance level, the implementation must be able to construct any `CoordinateReferenceSystem` which is legal under the ISO 19111 standard, including all of the projected systems.

### A.2.3 Conformance Level **R-B** – Referencing Authority Factories

This conformance level requires implementations to be able to instantiate types from the Authority factories.

The simplest conformance status for this level, Status **R-B1** requires being able to instantiate the most common objects from the OGC authority. The factory must be able to handle the following identifiers:

- CRS:1 (computer display)
- CRS:84 (geographic, WGS 84)
- CRS:83 (geographic, NAD83)
- CRS:27 (geographic, NAD27)
- CRS:88 (NAD vertical datum)
- AUTO2:42001 (Universal Transverse Mercator)
- AUTO2:42002 (Transverse Mercator)
- AUTO2:42003 (Orthographic)
- AUTO2:42004 (Equirectangular)

- AUTO2:42005 (Mollweide)

which are defined by the OGC for other implementation specifications. The factory should also be able to handle the URN form of these identifiers, such as <urn:ogc:def:crs:epsg:4326>, and the URL form, such as <http://www.opengis.net/gml/srs/epsg.xml#4326>.

The next conformance status for this level, Status **R-B2** requires being able to instantiate valid instances from any Well-Known Text (WKT) string. WKT is defined in OGC 01-009.

The final conformance status for this level, Status **R-B3** requires being able to instantiate a valid instances of the Datum, CoordinateSystem, or CoordinateReferenceSystem interfaces based on the codes and values in the EPSG database. The database is maintained by the Surveying and Positioning Committee of the International Association of Oil and Gas Producers and can be found at the URL <http://www.epsg.org/>

#### A.2.4 Conformance Level **R-C** – Referencing Operations

This conformance level requires implementations to be able to create the types in the `org.opengis.referencing.operation` and `org.opengis.parameter` packages.

The simplest conformance status for this level, Status **R-C1** requires implementations to provide the `CoordinateOperationFactory` type and be able to instantiate any of the types in the two packages.

The second conformance status for this level, Status **R-C2** requires a `CoordinateOperationAuthorityFactory` able to instantiate the `CoordinateOperation` instances based on the codes and values in the EPSG database.

#### A.2.5 Conformance Level **R-M** – MathTransform

This conformance level requires that the `CoordinateOperations` provided by the implementations be able to create the appropriate `MathTransform` instance for the `OperationMethod` of the `CoordinateOperation`. The `MathTransform` will then permit the calculation of coordinates in a target coordinate reference system from the values of a coordinate in a source coordinate reference system. The different status categories for this level are distinguished by the mathematical complexity of the `OperationMethod` which are supported.

The first conformance status for this level, Status **R-M1** requires that conformant implementations be able to instantiate the appropriate `MathTransform` instance for any `CoordinateOperation` which uses one of the `OperationMethod` types identified below:

- Affine general parametric transformation (EPSG:9624)
- Longitude rotation (EPSG:9601)
- Equidistant Cylindrical (EPSG:9842, 9823)
- Mercator (1SP) (EPSG:9804)
- Mercator (2SP) (EPSG:9805)

These `MathTransform` instances involve no shift in Datum and the most basic mathematical treatment.

The next conformance status for this level, Status **R-M2**, requires that conformant implementations be able to instantiate the appropriate MathTransform instance for any CoordinateOperation which uses one of the OperationMethod types identified below:

- Transverse Mercator (EPSG:9807)
- Transverse mercator (South Orientated) (EPSG:9808)
- Lambert Conic Conformal (1SP) (EPSG:9801)
- Lambert Conic Conformal (2SP) (EPSG:9802)
- Lambert Conic Conformal (2SP Belgium) (EPSG:9803)

These operations involve no shift in Datum but require more advanced mathematics.

The third conformance status for this level, Status **R-M3**, requires that conformant implementations be able to instantiate the appropriate MathTransform instance for any CoordinateOperation which uses one of the OperationMethod types identified below:

- Molodensky transformation (EPSG:9604)
- Abridged Molodensky transformation (EPSG:9605)
- Geographic/geocentric conversions (EPSG:9602)
- Geocentric translation (EPSG:9603)
- Position Vector 7-parameters (EPSG:9606)
- Coordinate Frame rotation (EPSG:9607)

These operations perform a shift in Datum but the shifts require only a small number of parameters.

The final conformance status for this level, Status **R-M4** requires that conformant implementations be able to instantiate the appropriate MathTransform instance for any CoordinateOperation which uses one of the OperationMethod types identified below:

- Ellipsoid to Geoid
- North American Datum Conversion (EPSG:9613)

These operations require a shift in Datum based on an extensive set of parameters using a numerical Grid or a set of spherical harmonic parameters.

### **A.3 Validation**

The GeoAPI source bundle, in the test packages of the conformance modules, contains a number of validator which can be used in JUnit test cases to test compliance of the objects created in an implementation. This is not as sophisticated as a full conformance test suite. Nonetheless, the GeoAPI validators can establish that certain instances are invalid and therefore can readily be integrated into the test suite of any implementation library.

#### **A.3.1 Example of a validation test**

The following code demonstrates an example which uses the validators contained in the GeoAPI binary distribution to evaluate an instance object created by the implementation within a unit test. This test would require the JUnit library, version 4 or later, on the Java Classpath.

```
import static org.opengis.test.Validators.*;
```

```
public class ValidationTests{  
  
    @Test  
    public void testCRS(){  
  
        //The implementation would build this CRS  
        CoordinateReferenceSystem crs = ...;  
  
        validate(crs);  
    }  
}
```

If the validation fails, the JUnit library would throw an `AssertionError`. Also, the GeoAPI binary JAR archive must be on the Java CLASSPATH for the library to be linkable at runtime.

## **Annex B**

(Informative)

### **GeoAPI Source Java Archive**

In addition to this document, this specification includes the normative GeoAPI Java archive file:

`geoapi-3.00-sources.jar`

That archive contains the authoritative Javadoc code documentation for the types and methods.

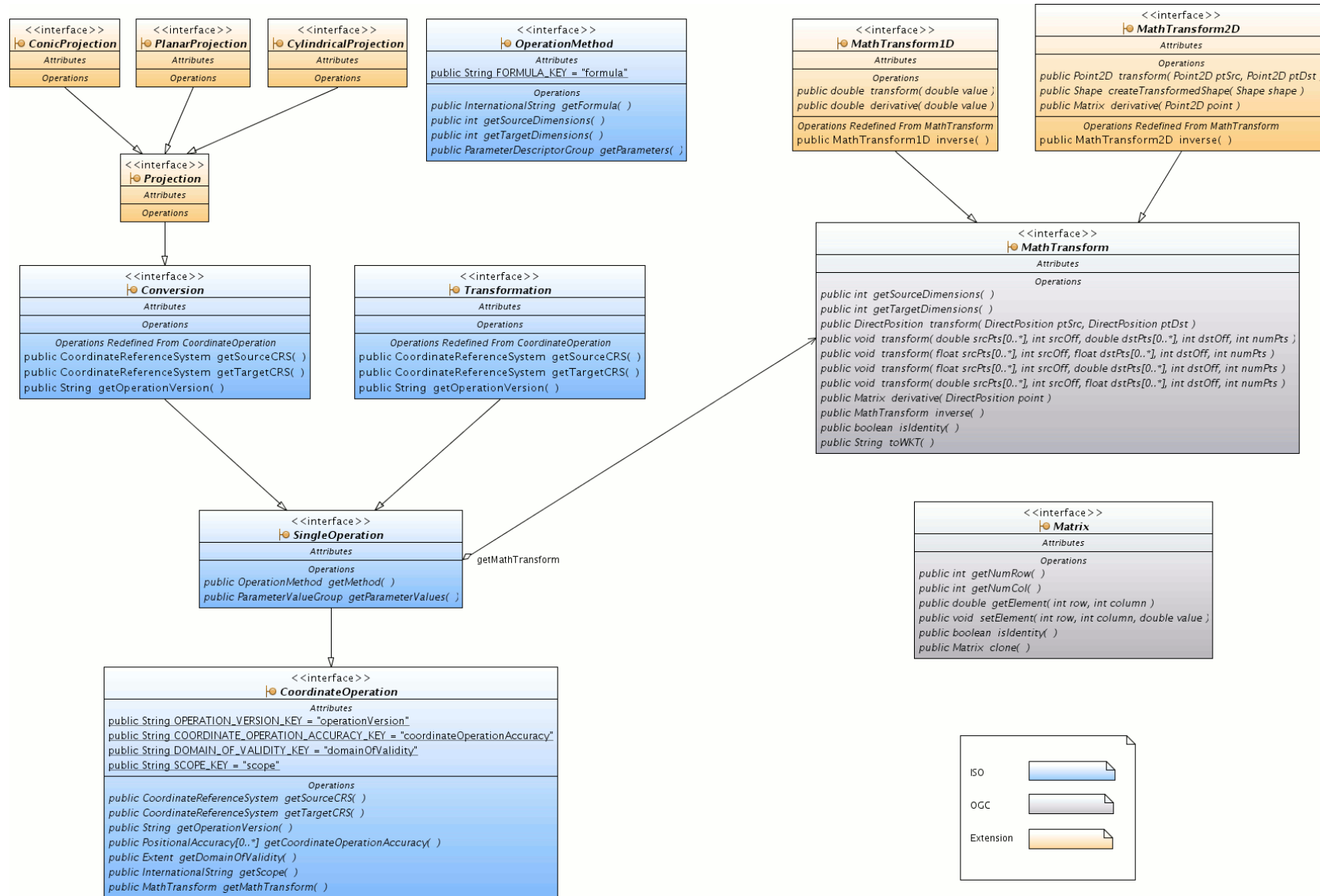
The Java archive file contains the following elements:

- META-INF/MANIFEST.MF
- org.opengis.annotation/
- org.opengis.geometry/
- org.opengis.geometry.coordinate/
- org.opengis.metadata/
- org.opengis.metadata.citation/
- org.opengis.metadata.constraint/
- org.opengis.metadata.content/
- org.opengis.metadata.distribution/
- org.opengis.metadata.extent/
- org.opengis.metadata.identification/
- org.opengis.metadata.lineage/
- org.opengis.metadata.maintenance/
- org.opengis.metadata.quality/
- org.opengis.metadata.spatial/
- org.opengis.parameter/
- org.opengis.referencing/
- org.opengis.referencing.crs/
- org.opengis.referencing.cs/
- org.opengis.referencing.datum/
- org.opengis.referencing.operation/
- org.opengis.util/

with each directory holding Java source files (.java extension) and some directories having documentation directories holding text or image files.

# Annex C (Informative)

## UML Diagram for referencing Operation types



**Annex D**  
(Informative)

**Departures from the ISO standards**

The following table lists all the departures from the ISO standards taken by the GeoAPI interface library.

**(TBD)**

## **Annex E**

(Informative)

### **Extensions to the ISO standards**

The following table lists all the extensions to the ISO standards added by the GeoAPI interface library.

**(TBD)**

## **Annex E**

(Informative)

### **Reference Implementation**

The GeoAPI library is released along with a Reference Implementation to demonstrate its viability and ensure that functional client code can be written with the release of this specification.

The Reference Implementation is provided by the Geotoolkit project (<http://www.geotoolkit.org/>). The implementation library itself is called `geotk-bundle-referencing-3.00.pack.gz` and is available from the <http://download.geotoolkit.org/> server.

The Reference Implementation is free software, licensed to all under the terms of the GNU Lesser General Public License, version 2.1, and therefore open for study, modification and redistribution, the latter under some constraints.

## Bibliography

- [1] ISO 31 (all parts), *Quantities and units*.
- [2] IEC 60027 (all parts), *Letter symbols to be used in electrical technology*.
- [3] ISO 1000, *SI units and recommendations for the use of their multiples and of certain other units*.
- [4] ISO 19103, *Geographic information – Conceptual schema language* 2005.
- [5] ISO 19115, *Geographic information – Metadata* 2003.
- [6] ISO 19115, *Geographic information – Metadata / Corrigendum 1* 2006.
- [7] ISO 19103, *Geographic information – Spatial referencing by coordinates* 2<sup>nd</sup> Edition, 2007.
- [8] Nordgren, Bryce, *Tools from ISO 19103: A GeoAPI Interface Proposal* USDA Forest Service, [????](#)
- [9] Nordgren, Bryce, *An ISO-19109 Primer (and comparison to the ComplexFeature effort in GeoTools)* USDA Forest Service, [????](#)
- [10] Nordgren, Bryce, *An ISO19123 Coverage Primer (and GeoAPI/GeoTools integration guide)* USDA Forest Service, [????](#)
- [11] Nordgren, Bryce, *The ISO TC/211 Image Concept: An integrated review and definition*. USDA Forest Service, [????](#)
- [12] Daly, Martin, ed. OGC 01-009 *Coordinate Transformation Services* Revision 1.00
- [13] Reynolds, Greg, ed. OGC 03-064 *GO-I Application Objects* Version 1.0