Open Geospatial Consortium Inc.

OpenGIS® Web Map Server Cookbook

November 4, 2004

Editor: Kris Kolodziej

OGC Document Number: 03-050r1

Version: 1.0.2

Stage: Draft

Language: English

Note: This document is not an OGC Standard. Internal and external documents cannot refer to it as such. Drafts are distributed for review and comment and are subject to change without notice.

Document Contact Information

If you have questions or comments regarding this document, you can contact:

Name	Organization	Contact Information
WMS Cookbook Editor	*OGC*	*editor@opengeospatial.org +1 (812) 334-0601*

Future Work

Table of Contents

Preface **1**
Submitting Organizations (The Contributors) 1
Organization of the Cookbook 2
Acknowledgments 3
Revision History 3
1. WMS Implementation: Overview **4**
1.1. Introduction: Web Mapping & Interoperability 4
1.2. Web Mapping Compliance: The WMS Service Interface 7
1.2.1. The WMS Interface Implementation Specification as an API 8
1.2.2. The WMS Interface Operations 9
1.2.3. WMS SLD Enabled Operations 10
1.2.4. Supported Distributed Computing Platform (DCP) 11
1.2.5. Relation of WMS to other OGC Web Services 12
1.2.6. Relation of OGC Web Services to the ISO Reference Model 13
1.3. Describing Your WMS Server: The GetCapabilities Request 16
1.3.1. Purpose of GetCapabilities: Allows a Map Server to Describe Itself 16
1.3.2. Implementing GetCapabilities: Request Parameters 17
1.3.3. GetCapabilities Response: Capabilities XML Document 18
1.4. Serving a Map: The GetMap Request 20
1.4.1. Purpose of GetMap 20
1.4.2. Implementing GetMap: Required Parameters 20
1.4.3. GetMap Response 23
1.4.4. Exception Handling 24
1.4.5. Cascading WMS Servers 24
1.5. Optional Operation: The GetFeatureInfo Request 25
1.5.1. Purpose of GetFeatureInfo 25
1.5.2. Implementing GetFeatureInfo: Required Parameters 26
1.5.3. GetFeatureInfo Response 26
1.6. Connecting drivers to OGC WMS Services 27
1.7. On What Technologies Does the WMS Specification Depend? 27
1.7.1. Extensible Markup Language (XML) 27
1.7.2. Validate XML - DTD (Document Type Definition) & XML Schema 28
1.7.3. Transform and Format XML:XSL (Transforming the Web) 29
1.7.4. Navigate in XML: XPointer, Xlink 31
1.7.5. Managing XML-Extensibility: Namespaces 31
1.7.6. Parse XML: DOM and SAX 32
1.8. Implementing WMS Compliance 33
1.8.1. Writing a WMS-Compliant Translator for a Map Server 33
1.8.2. Making a Map Server WMS-Compliant 33
1.8.3. Making a Web Client WMS Compliant 33
1.9. WMS Web Applications Development Technologies 33
1.9.1. CGI and Servlets 34
1.9.2. ASP versus JSP 34
1.10. Performance Criteria 36
1.11. Additional Information 37
1.11.1. OpenGIS Web Map Context Implementation Specification 37
1.11.2. OGC Compliance Testing 37
1.11.3. Conformance and Interoperability Test and Evaluation (CITE) Initiative 38

Preface

This OGC Cookbook is for the OpenGIS® Web Map Server (WMS) Interface Implementation Specification. This "Cookbook" is conceived as a means to share the current experiences in using the WMS interface for developing interoperable Web mapping applications. Developers around the world are implementing this specification in interfaces on a wide variety of commercial-off-the-shelf, government-off-the-shelf, open source, custom and legacy geoprocessing software products. We encourage these developers to share what they have learned by submitting implementation "recipes" to include in this cookbook. Providing a recipe is a community-spirited thing to do, but it also brings the submitter good publicity and, just as every telephone on the planet adds to the value of every other telephone, so every Web Map Server (and client) adds to the value of all the others.

Overall, this Cookbook provides the basic understanding and steps needed for implementing and exploiting the WMS interface and related technologies. In addition, the Cookbook includes examples of implementations, applications, and related helpful information for beginners and more advanced users. The Cookbook includes "recipes" or step-by-step instructions and recommendations on developing OpenGIS infrastructures using the WMS interface. It is an implementation guide.

It is assumed that the reader has reviewed the OpenGIS Web Map Server (WMS) Interface Implementation Specification[2]. Where appropriate, specific references to the WMS Specification are outlined within this Cookbook.

Other OpenGIS Cookbooks to follow this cookbook are the Web Feature Server (WFS), Web Coverage Server (WCS), and Geography Markup Language (GML) Cookbooks. These cookbooks are intended to be living documents, updated from time to time. Please email your suggested changes to t WMS Cookbook Editor at email@opengeospatial.org.

Many geoprocessing software vendors offer products that implement the OpenGIS Web Map Server Implementation Specification and other OpenGIS Specifications. Many of the vendors list these on OGC's "Implementing Products" page: http://www.opengeospatial.org/testing/product/index.php .

Submitting Organizations (The Contributors)

Contributions to this OGC Cookbook are from a wide variety of OGC members, including software vendors, integrators, universities, local and national government agencies, and non-governmental organizations who are users of the WMS interface. The cookbook is intended to draw from many sources and satisfy readers working in a wide variety of application settings.

This OGC Cookbook was made possible through the contributions of those listed below. All questions, remarks, and updates regarding these submission should be directed to the Cookbook editor or to the specific submitter of a particular section:

Submission Contact Points

CONTACT	*COMPANY*	*CONTRIBUTIONS*	*PHONE/FAX*	*EMAIL*
Kris Kolodziej	*OGC*	*Chapter 1* *Chapter 2: Preface,*	*+1-646-244-2731*	*KKolodziej@opengeospatial.org*

[2] *WMS 1.1.1 Adopted OpenGIS Specification http://www.opengeospatial.org/docs/01-068r3.pdf*

[Editor]		*System Architecture Example, XSL/XSLT Example*		
David Danko	*ESRI*	*Chapter 3: ArcIMS WMS Adapter Recipe*	*+1-703-506-9515*	*Ddanko@esri.com*
Markus Muller	*Bonn University & lat/lon*	*Chapter 2: DTD & XML Examples Chapter 3: lat/lon WMS Recipe*	*+49-228-732098*	*Markus_Muller@bug.hamburg.de*
Jeff McKenna	*DM Solutions Group, Inc*	*Chapter 3: UMN MapServer WMS Recipe (Client & Server)*	*+1-613-565-5056*	*mckenna@dmsolutions.ca*
Roland Stahl	*CSC Ploenzke AG & Wupperverband*	*Chapter 2: WMS User Experience*		*rstahl2@cscploenzke.de*
Karel Charvat	*WirelessInfo*	*Chapter 2: WMS User Experience & Software Architecture*		*kch@volny.cz*
Roger Harwell	*Intergraph*	*Chapter 3: GeoMedia WebMap WMS Adapter Kit Recipe*		*rdharwel@ingr.com*
Lucia Lovison	*Harvard University*	*Chapter 2: WMS User Experience*		*lovison@seismology.harvard.edu*
Allan Doyle	*International Interfaces*	*Chapter 3: Making a WMS of out Free Parts*		*adoyle@intl-interfaces.com*
Vincent Tao	*York University*	*Chapter 3: GSN 3D OGC Client Recipe*	*+1-416-736-5221*	*Tao@yorku.ca*
Rob Raskin	*NASA/Ocean ESIP, JPL*	*Chapter 2: GetCapabilties & `GetMap` Request Examples*		
Richard Pascoe, Hao Ding, Neville Churcher	*University of Canterbury*	*Chapter 2: Implementing OGC Web Map Service Client Applications Using JSP, JSTL and XMLC*		richard@cosc.canterbury.ac.nz

Organization of the Cookbook

The Cookbook is organized into three chapters that correspond to levels of detail and application:

Chapter 1: WMS Implementation: Overview establishes the background and context of the WMS interface implementation specification. In addition, WMS client and server development technologies (XML, XSL/XSLT, ASP/JSP, etc.) and approaches are reviewed. The chapter is a general orientation for all readers, including technical managers and developers.

Chapter 2: **WMS Examples** addresses the design architecture of software systems that implement the WMS interface. In addition, user experiences provide an explanation of the practical use of the WMS Implementation Specification. Also included are WMS request examples as well as DTD/XML and XSL/XSLT stylesheet examples that show how these technologies/tools are used as part of WMS client and server implementation. The chapter is aimed at technical managers and developers, but suitable for all readers (especially the User Experience section).

Chapter 3: WMS Implementation Recipes addresses the implementation of existing WMS interface compliant software (Web clients and map servers) and includes step-by-step instructions (recipes) for deploying them. The chapter is for technical personnel who want to deploy these systems for end-users, or for developers who want to implement similar systems.

Acknowledgments

In addition to each of the contributors, special thanks go to the following people:

- Professor Stephan Winter of the Technical University of Vienna for thorough feedback, and to Prof. Winter's students at Carinthia Tech Institute (Villach, Austria) in his Geoinformation class (5th semester) for doing a hands-on testing of this Cookbook by implementing one of the recipes.

- Lucia Levison of Harvard University for general feedback on the Cookbook (in addition, to their User Experience contribution).

Revision History

Version	Date	Author/Editor	Comments
1.0.0	*April 14, 2003*	*Kris Kolodziej*	
1.0.1	*June 4, 2003*	*Kris Kolodziej*	*Emphasize that others are invited to submit examples, experiences, and recipes.*
1.0.2	*November 4, 2004*	*Kris Kolodziej*	*Editorial fixes.*

1. WMS Implementation: Overview

This chapter establishes the background information of Web mapping, system interoperability, and the OpenGIS® WMS Specification implementation.

This chapter also presents the technologies and tools for WMS client and server development. WMS Specification version 1.1.1 is directly referenced to the corresponding "WMS requests" sections (GetCapabilities, GetMap, GetFeatureInfo) of the specification for the information needed when implementing the WMS interface.

1.1. Introduction: Web Mapping & Interoperability

Organizations and companies have been providing online mapping services for years. These Web mapping systems[3] have been implemented as a set of proprietary systems. As a result of this isolated development, online mapping services from different vendors cannot interoperate. The current status of the lack of geographic information standardizations is shown in Figure 1 which depicts non-interoperable web mapping systems.

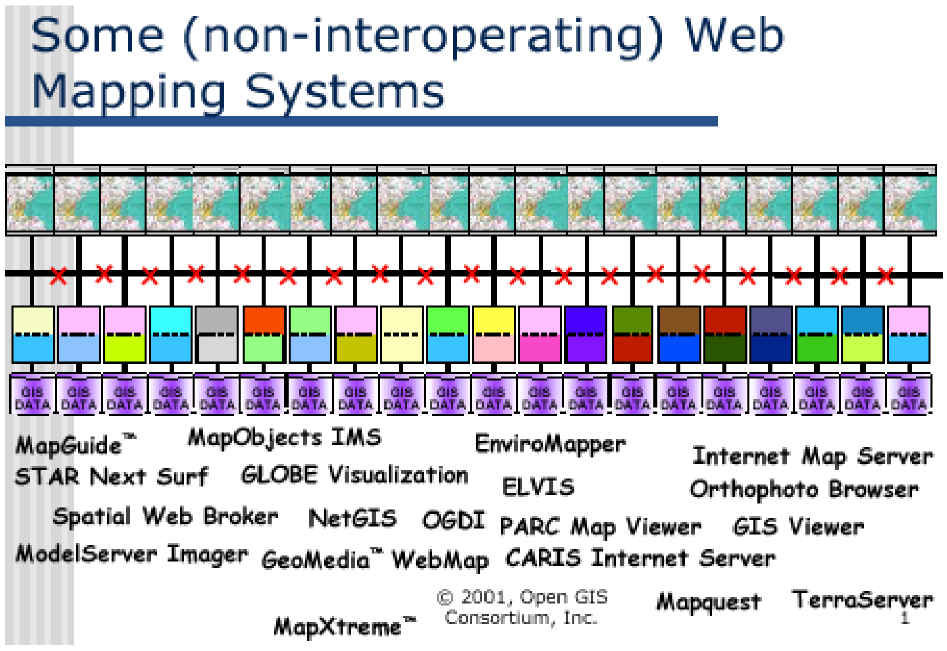


Figure 1: Current Status of Non-Interoperable Web Mapping Systems

This diagram is actually overoptimistic considering that the map views (top of diagram) are identical. In reality, they are different because the majority of today's technology does not provide common map views across GIS platforms. The lack of interoperability of either data or services is indicated by the red x's. In

[3] *Web mapping is the set of products, standards and technologies that enable access to geographic information, usually portrayed as maps, via the Web.*

practice, it means that many technology islands are created and preserved, and that many users are locked into single-vendor solutions. This situation (lack of interoperability) is slowly improving but, unfortunately, most Web mapping applications today are still inseparably tied to a specific server implementation. In other words, the Web client is hard-coded to interact with a particular vendor's proprietary map server implementation.

Figure 2 below shows a scenario where the user must run three different Web applications in order to access the data and functionality provided by three different server implementations. In this situation, there is very little interoperability or reuse of the Web client and server implementations.

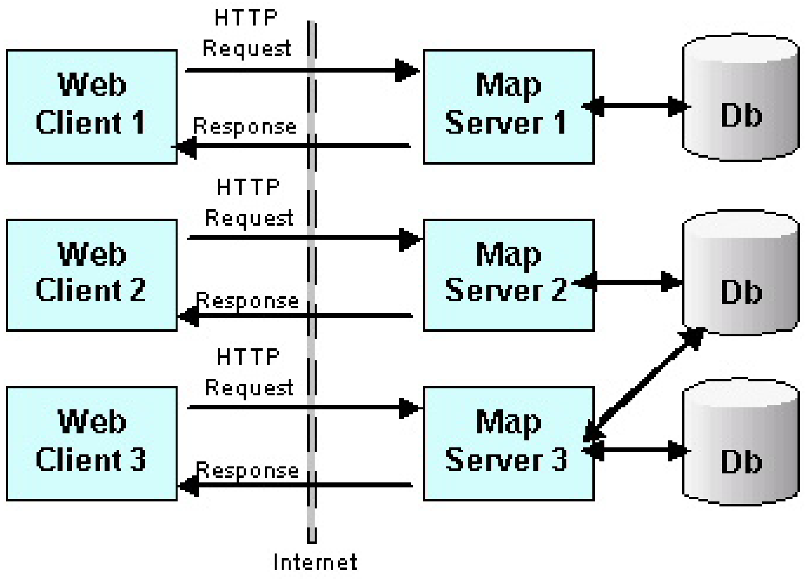


Figure 2: Client/Server Lack of Interoperability

Since data are often accessible only through one particular server, there is also very limited ability for a user to transparently access data of interest from other Map servers. In this diagram, only Web client 3 enables access to more than one database. Unfortunately, Web client 3 may not provide all the functionality that Web client 1 and Web client 2 offer. Even with Web client 3's ability to access data of interest from multiple databases, the user must still run three different applications from the different Web clients to perform a given task.

With Web mapping, as with many application types on the Web, there is a large set of servers from multiple vendors and organizations. The opportunity the Web provides for broad access is not realized if each server has a different proprietary implementation with no published interface specification. Even if an implementation is publicly documented, it may not be standard to the extent that it is in common use by multiple commercial implementations.

To address this problem, the Open Geospatial Consortium, Inc. (OGC) developed a non-proprietary Web mapping approach based on open interfaces, encodings and schemas. The OGC Specification Program and Interoperability Program provide an industry consensus process to plan, develop, review and officially adopt OpenGIS Specifications for interfaces, encodings and schemas that enable interoperable geoprocessing services, data, and applications.

Interoperability, at a technical level, refers to the ability for a system or components of a system to provide information portability and interapplication as well as cooperative process control. Interoperability comprises intercommunication at communication level protocol, hardware, software, and data compatibility layers. The aforementioned might be called *syntactic* interoperability, in the sense of parameter passing. *Semantic* interoperability, in contrast, deals with the domain knowledge necessary for informatics services to "understand" each other's intentions and capabilities.

Interoperability, in the context of the OpenGIS Specification Program, is software components operating reciprocally (working with each other) to overcome tedious batch conversion tasks, import/export obstacles, and distributed resource access barriers imposed by heterogeneous processing environments and heterogeneous data. Interoperability, with respect to geoprocessing, refers to the ability of digital systems to 1) freely exchange all kinds of spatial information and 2) cooperatively, over networks, run software capable of manipulating such information.

Vendors working together in the OGC's Web Mapping Testbed, and more recently, the OGC Web Services (OWS) Initiative, have created ways for vendors to write Web-based software that is interoperable. (Note that OGC also addresses, in some of its initiatives, distributed computing platforms other than the Web.) Their achievement enables users to immediately overlay and operate on views of digital thematic map data from different online sources offered though different vendor software. Moreover, map and imagery suppliers are beginning to make their data available over the Web through these vendors' OpenGIS-conformant servers.

The conceptual picture of how map overlay works in an interoperable way is portrayed in Figure 2 below.

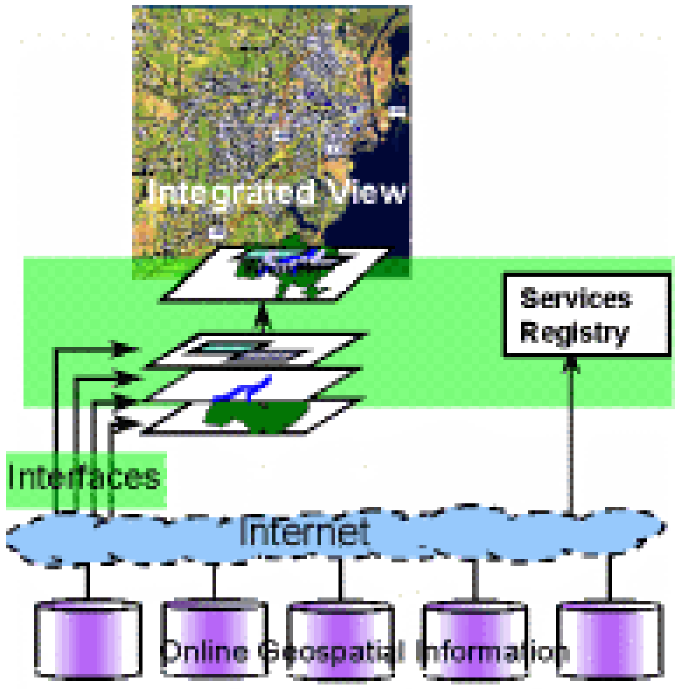


Figure 2: Interoperable Map Overlay

The interoperability that enables this automatic map overlay comes from a set of common interfaces for communicating a few basic commands/parameters. This set of interfaces is known as the OpenGIS Implementation Specification [4], and includes the Web Map Server (WMS) interface implementation specification. These specifications address basic Web computing, image access, display, manipulation and coordinate transformation capabilities. That is, they specify the request and response protocols for open Web-based client/map server interactions.

[4] *For a full listing of OpenGIS Implementation Specifications see: http://www.opengeospatial.org/specs/?page=specs.*

Overall, OGC interfaces provide a high level of abstraction that hides the "heavy lifting" in the Web Mapping environment. The heavy lifting includes finding remote data store servers, requesting data from them in specifically defined/standardized structures, attaching symbols intelligently, changing coordinate systems, and returning information ready to be displayed at the client - all in a matter of seconds.

With standards-based interoperable Web mapping, each map server implements a common interface, a messaging protocol such as the WMS interface for accepting requests and returning responses. Now, the same client has Web access to potentially all available map servers and multiple data sources, where each map server is accessed by a client through the common interface. This concept of interoperable, distributed mapping systems is portrayed below in Figure 4 below.

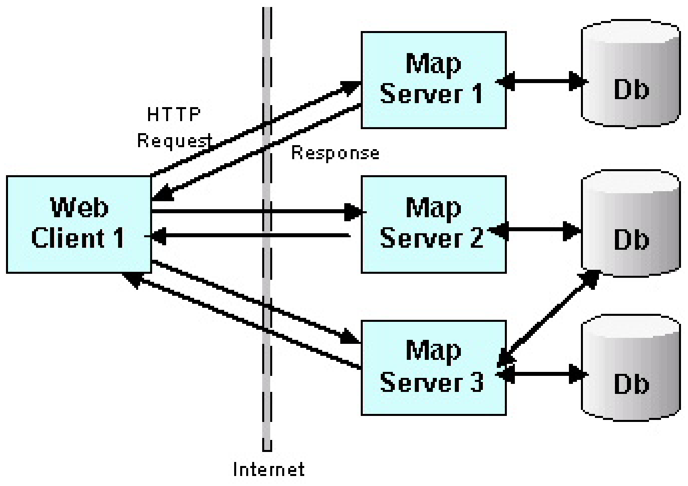


Figure 4: Client/Server Interoperability

This approach allows, among other things, the user to run a single client that accesses all the capabilities of each server. This enables a more open application environment where the best features of available Web services can be flexibly combined in innovative and previously unimagined ways to solve novel and increasingly complex problems.

The work associated with OGC's Interoperability Program of fast paced testbeds and pilot projects has led to many accomplishments. It led to a foundation of web-based interoperability involving not only map display, but also more sophisticated geoprocessing functions, as well as location based services, sensor and camera geolocation, Web catalogs of spatial data and spatial Web services, and other capabilities.

As this trend of accessing GIS data via OpenGIS standards continues, the "spatial Web" will become as open as the Web itself. Web users will easily find, view, overlay, and combine different thematic maps for a given region. One important effect is a great increase in the utility and commercial value of location-aware, Internet-connected cell phones, PDAs, laptops, and car computers.

1.2. Web Mapping Compliance: The WMS Service Interface

The OpenGIS Web Map Service Interface Implementation Specification offers a way to enable the visual overlay of complex and distributed geographic information maps simultaneously, over the Internet. In the context of WMS a "map" is a raster graphic "picture" of the data rather than the actual data itself.

The WMS Specification is a remarkable technical and commercial breakthrough. Software conforming to the WMS Specification, using ordinary Web browsers, is able to automatically overlay map images obtained from multiple dissimilar map servers, regardless of map scale, projection, earth coordinate system or digital format. Hundreds of billions of dollars worth of digital maps and earth images, which until now could not be accessed and used without special skills and software, can now become an integrated part of the broader information infrastructure. See Figure 3 for a conceptual view of the situation.

In essence, the WMS Specification enables the creation of a network of interoperable map servers from which WMS clients can overlay and build customized maps. The WMS client can be either an HTML page returned by a WMS server (cascading) or a specialized browser plug-in built with Java or ActiveX that connects to different WMS servers.

WMS clients can specify requested layers, layer styles, the geographic area of interest or bounding box, the projected or geographic coordinate reference systems (called the Spatial Reference System by OGC), image file format including width and height size, and also background transparency.

When the WMS client makes requests from multiple WMS services using the same bounding box, Spatial Reference System, and output size, the returned image files can then be overlaid to create an infused or composite map. It is important that the map requests specify transparency in order to see lower map images.

This functionality allows organizations to create WMS data networks that enable users to combine GIS data from different sources based upon their individual needs. It also allows individual WMS providers to focus on data particular to their applications and not translate "backdrop" datasets.

Figure 5, below, illustrates how WMS servers serve their information directly to a WMS client (Web browser) where the maps are "fused" and overlaid from WMS servers.

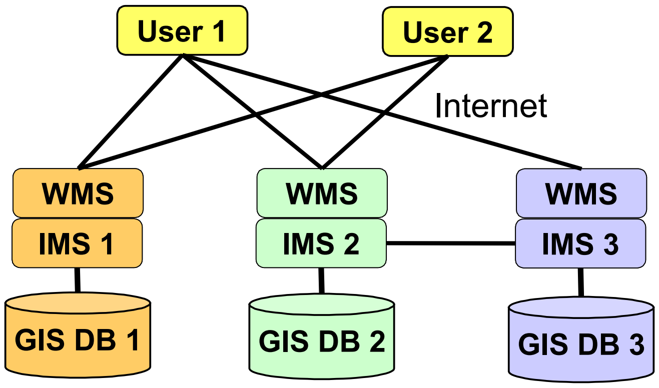


Figure 5: WMS Communication (courtesy of CSC PloenzkeAG).

There are many special cases that must be handled, making the official WMS Specification document a bit complex, but at its most basic level, the application developer only needs to know how to populate the correct values for parameters associated with each WMS request. In other words, the developer doesn't need to know the internal implementation of the application, just the formation of the request to be interpreted by the application. See Chapter 2 for some examples of WMS requests and responses.

While the WMS Specification provides a simple, interdependent framework to set up Web map server - it does have its limitations. A major WMS limitation is the users' inability to modify the map view, by zoom or pan functions, on the client without making a new request to the WMS servers. The other major limitation is the lack of rich functionality such as geometry editing or network tracing.

1.2.1. The WMS Interface Implementation Specification as an API

An API (Application Programming Protocol) is a set of software templates that gives software developers a unified way of addressing functionality on dissimilar systems. Typically, an API is a library of functions or subroutines that give application programmers access to the functionality available in a resource such as an operating system, imaging system, graphics device, etc.

With respect to the OpenGIS Specifications, an API is an interface definition that permits writing OpenGIS services for application programs without knowing details of their internal implementation.

The WMS Specification is essentially an API that enables programmers to add an interoperability interface to different geoprocessing systems from different vendors and of different types (GIS, imaging, navigation, desktop mapping, etc.).

There are three main components to any online API:

- A vocabulary for the request of information.
- A vocabulary for the response to requests.
- A protocol for the exchange of requests and responses.

The World Wide Web is arguably the first online service API. There is a client – the Web browser; a server that understands requests and how to fulfill them – the Web server; and a communication protocol – HTTP. The Web browser sends commands in a format the server understands to a Web server using the HTTP protocol (over TCP/IP). If a request is sent to a Web server using vocabulary it doesn't understand, it will respond with an error. In essence, the WMS Specification defines the vocabulary and the syntax of the commands/operations that enable Web servers and clients to communicate over the HTTP protocol.

That is the extent of the Web API. It is elegant in its simplicity, in that it enables many highly functional applications to be built on the Web. Its simple hyperlinking scheme for embedding URLs from multiple Web servers makes it an example of content integration from multiple sources, and it has been able to support orders of magnitude more users than it was intended to.

1.2.2. The WMS Interface Operations

The WMS 1.1.1 specification standardizes three operations (two required and one optional) by which maps are requested by clients, and it standardizes the way that servers describe their data holdings. In addition, the WMS Specification defines a set of query parameters and associated behaviors.

The three operations (requests) are listed below and shown in Figure 6:

1. `GetCapabilities` (required)
2. `GetMap` (required)
3. `GetFeatureInfo` (optional)

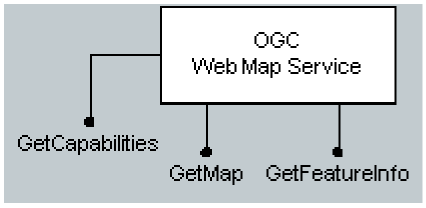


Figure 6: The OpenGIS Web Map Service Interface

Each operation is described in detail in its individual section of this chapter. Beyond these basic WMS operations, additional operations are defined by the OpenGIS Styled Layer Descriptor (SLD) Specification. See Section 1.2.3. "WMS SLD Enabled Operations" for more on this.

The WMS Specification standardizes or "defines a syntax for WWW Uniform Resource Locators (URLs) that invoke each of these operations. Also, an Extensible Markup Language (XML) encoding is defined for service-level metadata" (WMS 1.1.1) for the `GetCapabilities` operation.

In essence, a WMS server can do three things:

1. Produce a map (as a picture, as a series of graphical elements, or as a packaged set of geographic feature data),

2. Answer basic queries about the content of the map, and

3. Tell other programs what maps it can produce and which of those can be queried further.

A WMS client (e.g. a standard Web browser) can ask a WMS server to do these things just by submitting requests in the form URLs. The content of such URLs depends on which of the three tasks is requested. All URLs include the WMS Specification version number and a request type parameter. In addition,

1. To produce a map, the URL parameters indicate which area of the Earth is to be mapped, the coordinate system to be used, the type(s) of information to be shown, the desired output format, and perhaps the output size, rendering style, or other parameters.

2. To query the content of the map, the URL parameters indicate what map is being queried and which location on the map is of interest.

3. To ask a map server about its holdings, the URL parameters include the "capabilities" request type.

1.2.3. WMS SLD Enabled Operations

The WMS Specification applies to a Web Map Service that publishes its ability to produce maps rather than its ability to access specific data holdings. A basic WMS classifies its georeferenced information holdings into "Layers" and offers a finite number of predefined "Styles" in which to display those layers.

The behavior of a WMS service can be extended to allow user-defined symbolization of feature data instead of named Layers and Styles. While a WMS currently can provide the user with a choice of style options, the WMS can only tell the user the name of each style. It cannot tell the user what each portrayal will look like on the map. More importantly, the user has no way of defining unique styling rules. The capability for a human or machine client to define these rules requires an extension - a styling language that the WMS client and WMS server can both understand.

The OpenGIS Styled Layer Descriptor (SLD) Specification[5] describes this extension. In brief, an SLD-enabled WMS retrieves features from a Web Feature Service and applies explicit styling information provided by the user in order to render a map. A WMS client retrieves capabilities from a WMS server. If the WMS server supports SLD, the WMS client allows the user to create custom styles on traditional WMS layers (in SLD terminology, UserStyles for NamedLayers), which then makes an SLD-enabled `GetMap` request to retrieve a map.

SLD is robust enough to fulfill a wide range of cartographic needs and is terse enough to be useful even using only HTTP GET as a transport method. However, some of the current SLD limitations are: (1) there is no elegant way to specify a thematic or chloropleth map. For example, the user can not encode data in four colors starting with gray and ending with black without specifying the exact data ranges for each color and the exact color value for each range: (2) the ability to create styles lacks a style library service.

An SLD WMS adds the following additional operations that are not available on a basic WMS:

1. **DescribeLayer** – asks for an XML description of a map layer. The result is the URL of the Web Feature Server (WFS) containing the data and the feature type names included in the layer.

2. **GetLegendGraphic** – provides a general mechanism for acquiring legend symbols, beyond the LegendURL reference of WMS Capabilities.

3. **GetStyles** – used to retrieve user-defined styles from a WMS.

4. **PutStyles** – used to store user-defined styles into a WMS

1.2.4. Supported Distributed Computing Platform (DCP)

At present, the only distributed computing platform (DCP) explicitly supported by OGC Web Services is the World Wide Web itself, or more specifically Internet hosts implementing the HTTP. (Note that CORBA, SQL, and COM are supported by some OpenGIS Specifications, enabling interoperability across other networks and between diverse software systems running simultaneously on a single computer. See also the proposed DCP-independent specification work proposed in the Geographics Objects Feasibility Study.)

The Web, in general, requires that requests and responses be sent over the Internet using the HTTP protocol. HTTP supports two request methods: GET and POST. One or both of these methods may be defined for a particular OGC Web Service type and offered by a service instance, and the use of the Online Resource URL differs in each case. The basic WMS Specification only defines HTTP GET for invoking operations.

Basic GET encodings are directly usable by standard World Wide Web user agents, and may be bookmarked, sent in email, pasted into HTML documents, and so forth. The ease of use of the GET encoding is its primary benefit. However, it does have some disadvantages. Additional semantics must be defined to, for example, associate a list of layers with a list of styles. SLD WMS requests described in Section **Error! Reference source not found.**, below, which include an XML description of the styling, are difficult to encode directly and require that the request URL make reference to a separate SLD URL.

However, a candidate WMS Implementation Specification Part 2: XML for Requests using HTTP POST5 [6] defines optional HTTP POST encodings meant to provide additional structure in the request message and

[5] *SLD specification: https://portal.opengeospatial.org/files/?artifact_id=1188*
[6] *WMS Implementation Specification Part 2: XML for Requests using HTTP POST. See URL http://www.opengeospatial.org/docs/02-017r1.pdf. This document is a discussion paper for a new Part 2 of the WMS Implementation Specification.*

thereby to allow additional functionality. The greatest benefit is felt in the `GetMap` operation, where the comma-separated list of Layer names in HTTP GET can be replaced by a sequence of XML elements, each of which is either a named or a user-defined Layer, and directly associating Style and Filter information within each Layer. The `GetFeatureInfo` operation, which includes most of a `GetMap` request, benefits in a similar way.

An Online Resource URL intended for HTTP GET requests is in fact only a URL prefix to which additional parameters must be appended in order to construct a valid Operation request. A URL prefix is defined as an opaque string including the protocol, hostname, optional port number, path, a question mark '?', and, optionally, one or more server-specific parameters ending in an ampersand '&'. The prefix uniquely identifies the particular service instance. For HTTP GET, the URL prefix must end in either a '?' (in the absence of additional server-specific parameters) or a '&'. In practice, however, Web clients should be prepared to add a necessary trailing '?' or '&' before appending the Operation parameters defined in this specification in order to construct a valid request URL. An Online Resource URL intended for HTTP POST requests is a complete and valid URL to which Web clients transmit encoded requests in the body of the POST document.

There were many architectural decisions made that now seem obvious for selecting the Web (HTTP) as the DCP. The most important feature is that the HTTP communication protocol, which is used for sending requests (e.g. get me a map of a particular place) and responses (e.g. an HTML text or image) is a stateless, one-time transaction, meaning that one request gets exactly one response. Nothing about the requesting WMS client is known before or after the lifetime of the request.

The simplicity of the stateless HTTP protocol for requesting information over the Web can pose some limitation when developing Web client application. Programmers get around this limitation with cookies, Java applets, and other means, but these are all extensions. In order to build Web applications that allow for customized content based on the user's input, the CGI (Common Gateway Interface) is used. This is a standardized way to send initialization parameters to any program running on a Web server. These services, commonly called CGI engines (e.g., Microsoft ASP pages, Java servlet engines, or Allaire ColdFusion) make Web pages more like true user interfaces to powerful, customizable, distributed programming resources.

OGC's WMS Implementation Specification is a standard vocabulary for basic Web mapping services that is based on CGI. The CGI request is still a URL, but at some point in the URL, there is a question mark, and everything after the question mark is a list of key/value pairs.

1.2.5. Relation of WMS to other OGC Web Services

Recognizing that GIS users require functionality exceeding that possible within WMS (i.e., image files), OGC developed a number of other specifications that extend the functionality of OpenGIS WMS compliant systems.

The OGC Web Services (OWS) [7]suite includes three principal types of georeferenced information access services. Besides WMS, it also includes the Web Coverage Server (WCS)[8] and the Web Feature Server (WFS)[9]. Other standards include the Simple Feature Specification (SFS), the Geography Markup Language (GML), and others. While these standards are independent of each other, they are complementary to each other.

[7] *OpenGIS Web Services (OWS): http://www.opengeospatial.org/initiatives/?iid=78*
[8] *WCS – Web Coverage Service—Interoperability Program Report: http://www.opengeospatial.org/docs/03-065r6.pdf*

[9] *WFS 0.0.14 specification: https://portal.opengeospatial.org/files/?artifact_id=7176*

Figure 7 is an architecture diagram showing conceptually how some of the OGC Web Services are related, and naming some of the operations they define.

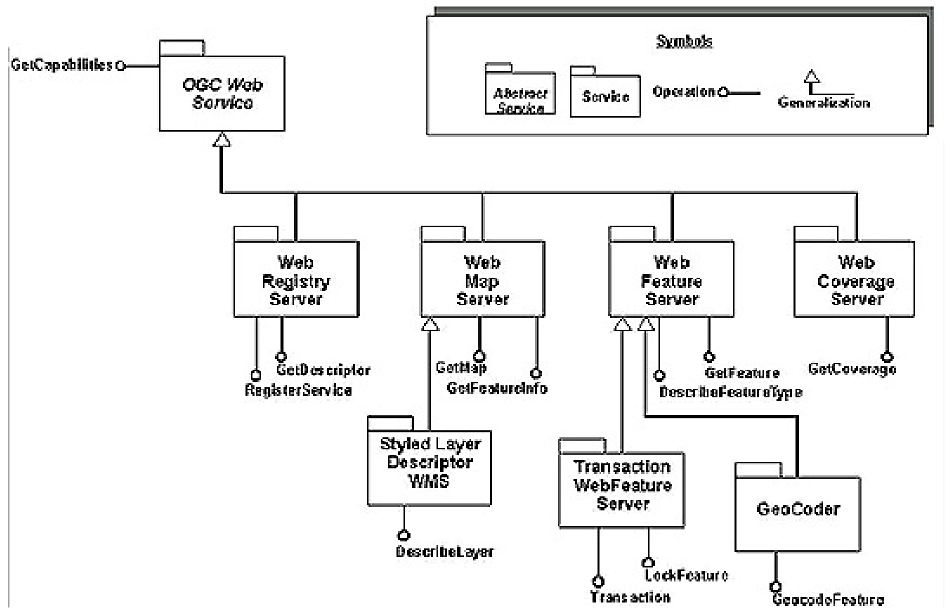


Figure 7: OGC Web Service Architecture Diagram.

The architecture of Web-based geospatial services specifies the scope, objectives and behavior of a Web services system and its functional components. The reference architecture is independent of particular technology choices and will evolve in response to implementation experiences.

A reference architecture model, in general, brings together standards at two different levels of abstraction, and under two different architectural viewpoints:

- Abstract models specify what information or requests are valid in principle (independent of individual computing environments). They define essential concepts, vocabulary, and structure of geospatial services and information transfer. These models set the stage for creating implementable specifications, and for extending existing ones to new environments.

- Implementation specifications tell software developers how to express information or requests within a particular distributed computing environment (e.g., World Wide Web, CORBA, J2EE, .NET). Implementation specifications generally include access protocols, object models, and naming conventions. Such specifications are specific to a targeted computing environment.

1.2.6. Relation of OGC Web Services to the ISO Reference Model

The ISO Reference Model[10] provides additional background on conceptual models and their role in specification design and organizes standards along a generic "stack" of interoperability layers, depicted in

[10] ISO Reference Model, http://www.iso.ch/iso/en/CatalogueDetailPage.CatalogueDetail?CSNUMBER=26002. *(ISO 19101:2002).*

Figure 8. OGC's Abstract Specification[11] Topic 0 (Overview, Section 2) explains the roles of abstract and implementation models, and the interdependence of service invocation and information transfer.

In essence, there are two types of standards:

- Service invocation standards: define the interfaces that allow different systems to work together, or the expected behavior of software systems.
- Information transfer standards: define the content of geospatial information or its encoding for transfer between different processing systems.

For distributed computing, the service and information viewpoints are crucial and intertwined. For instance, information content isn't useful without services to transmit and use it. On the other hand, invoking a service effectively requires that its underlying information be available and its meaning is clear. However, the two viewpoints are also separable: one may define how to represent information regardless of what services carry it; or how to invoke a service regardless of how it packages its information.

The "interoperability stack" presented in Figure 8 describes a layered architecture of technology and standards on which services can be implemented and deployed. The lowest levels of the stack enable connectivity of software components by enabling them to bind, send and receive messages. Higher levels in the stack enable interoperability and, via publish-find-bind mechanisms, allow software components to transparently work together in more integrated and dynamic ways.

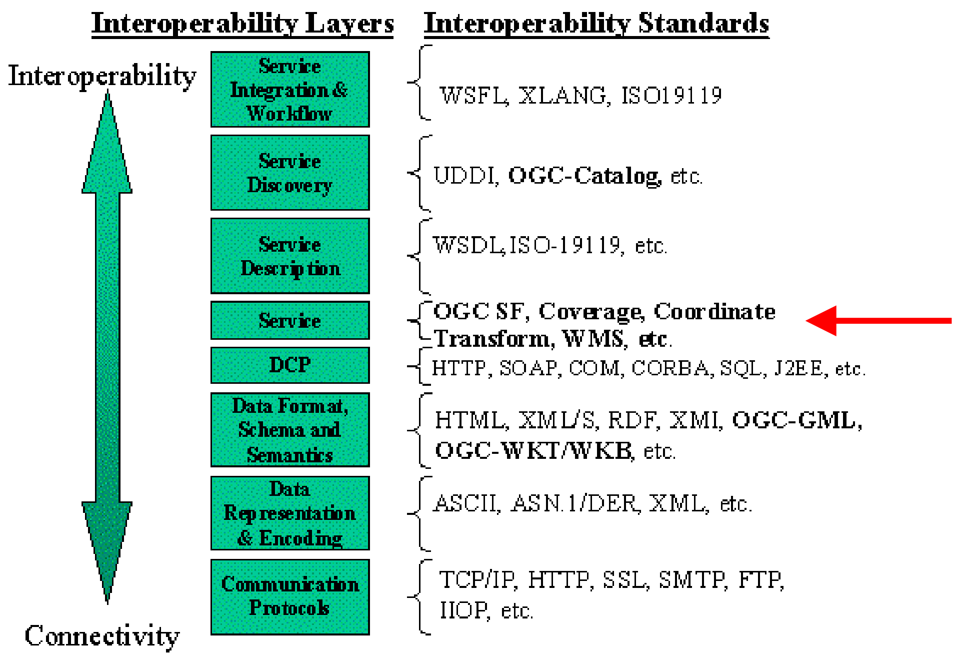

Interoperability Layers
Interoperability Standards
Interoperability
Connectivity
Service Integration & Workflow
WSFL, XLANG, ISO19119
Service Discovery
UDDI, OGC-Catalog, etc.
Service Description
WSDL, ISO-19119, etc.
Service
OGC SF, Coverage, Coordinate Transform, WMS, etc.
DCP
HTTP, SOAP, COM, CORBA, SQL, J2EE, etc.
Data Format, Schema and Semantics
HTML, XML/S, RDF, XMI, OGC-GML, OGC-WKT/WKB, etc.
Data Representation & Encoding
ASCII, ASN.1/DER, XML, etc.
Communication Protocols
TCP/IP, HTTP, SSL, SMTP, FTP, IIOP, etc.

Figure 8: Services Interoperability "Stack" (ISO Reference Model)

At the foundation of the stack are communication protocols such as TCP/IP, HTTP, SMTP, IIOP and FTP. With the exception of pure binary data, structured data will typically be encoded as XML. Formats for data encoding are described in a schema language such as DTD, XML Schema, RDF or XMI. OGC has defined a family of schemas for encoding simple features as "well-known" binary (WKB) and plain text

[11] OGC Abstract Specification Overview, http://www.opengeospatial.org/docs/99-100r1.pdf.

(WKT) as well as GML for encoding features as XML. The distributed computing platform (DCP) layer is focused primarily on the infrastructure for enabling distributed services.

A DCP is a standardized software environment enabling distributed computing by supporting cooperation between software executing on multiple computers that converse over a communications network. To the extent services are constructed from reusable software components, standard (or at least widely used) DCPs such as COM, CORBA, J2EE and SQL reside in this layer. Similarly, HTTP and SOAP are infrastructure technologies specifically enabling binding to services deployed across the Web.

For the services layer, OGC has defined implementation specifications for accessing and manipulating "simple features" via bindings for the COM, CORBA and SQL platforms. Additional implementation specifications from OGC include Gridded Coverages and Coordinate Transformations. New implementation specifications are emerging from the OGC specification process for Feature and Coverage access, Portrayal, Gazetteer, Geocoder and Geoparser services for the Web.

The Service Description layer is used to provide fundamental information required for services to discover, bind and interoperate. These include:

- Types of messages being exchanged between a service provider and a service requestor.
- Operations supported by a service provider.
- Rules for binding to a service provider
- The network address of a service provider.

The Service Discovery layer is the set of standards and technologies for publishing and finding service providers. Service descriptions tell where and how to access Web services. Service discovery enables web service descriptions to be found and utilized by service requestors. The UDDI[12] is an emerging technology for defining mechanisms for discovery of Web Services. The OGC Catalog Service implementation specification defines a service for supporting the discovery of geospatial content and services.

The top-level Integration and Workflow layer is focused on standards and technologies for enabling integration of services to support decision-making, modeling, workflow and business process integration within organizations and among Information Communities.

The OGC Web Services (OWS) reference architecture is consistent with ISO 19119 (Geographic Information - Services)[13] and ISO 10746 (Reference Model for Open Distributed Processing, RM-ODP)[14]. Even though the OWS reference architecture does not mandate particular implementation choices, it is presumed that, for the purposes of the OGC testbeds and initiatives, XML and XML Schema are used as the type definition language. Developers can assume that most OGC technology implementations will depend on the Internet and Web as the distributed computing platform, but many of the OpenGIS Specifications support other distributed computing platforms and some of OGC's initiatives focus on other platforms.

High-level objectives of OWS include:

- Support many independently-developed implementations of a given service type.
- Support many independently-provided instantiations of different of services.

[12] *UDDI: http://www.uddi.org*
[13] *ISO 19119 (Geographic Information – Service: http://www.fig.net/figtree/pub/fig2002/JS4/JS4percivall.pdf*
[14] *ISO 10746 (Reference Model for Open Distributed Processing, RM-ODP): http://community-ml.org/RM-ODP/*

- Find at runtime specific instantiations of services based on service characteristics such as service type, service content, or service quality.
- Provide access to metadata about the description of services, their location on the Internet, and the means of accessing and using these services.
- Discover what services can be used to access specific data holdings.
- Invoke services at runtime to perform useful tasks using discovered metadata.
- Enable ad-hoc chaining of services to satisfy aggregated workflow processes.
- Develop a common model for publishing, discovering, binding and chaining Web services.

The architecture presented in the OGC Basic Service Model[15] encapsulates the design requirements for enabling interoperation between instances of "Web Services" deployed using OGC Web Services specifications.

Based on these requirements, the reference architecture specifies mechanisms and rules for:

- Publishing web service descriptions.
- Publishing interfaces.
- Identifying operators (processing method).
- Publishing shared information objects.
- Organizing entities into hierarchical relationships.

1.3. Describing Your WMS Server: The GetCapabilities Request

1.3.1. Purpose of GetCapabilities: Allows a Map Server to Describe Itself

The `GetCapabilities` request returns the WMS server's service-level metadata, which is a machine-readable (and human-readable), description of the WMS service's information content and acceptable request parameters. The response to a `GetCapabilities` request is general information about the service itself and specific information about the available maps. This response is the "capabilities" document, which is an XML configuration file that is provided to requesting WMS clients. This XML file is the metadata about a WMS server indicating its data holdings and abilities. A server sends this information upon request as an XML document formatted according to well-defined Document Type Definition (DTD). The most critical part of the WMS Capabilities XML is the Layers and Styles it defines. This file is required by the WMS Specification and must conform to the OGC's WMS Document Type Definition.

[15] *OGC Basic Service Model, URL: http://www.opengeospatial.org/docs/01-022r1.pdf*

When requesting a map, a WMS client may specify the layer(s) information and associated styles, as well as the desired output format. First, however, a WMS client needs to find out what it can request from a particular WMS server (find out the WMS server's service capabilities). In order to find out what layers a WMS server supplies and what projections it supports, a WMS client makes a "Capabilities Request." Another purpose of the `GetCapabilities` request is to declare the `GetMap` services that are provided. You must be able to deliver an XML metadata file via HTTP upon receiving a request such as:

```
http://www.opengeospatial.org/wms/process.cgi?
REQUEST=GetCapabilities&VERSION=1.1.0&SERVICE=WMS
```

This request can be broken up into URL components, as shown below:

URL Component	Description
http://www.opengeospatial.org/wms/process.cgi?	*URL Prefix of server*
VERSION=1.1.1&	*Request Version*
REQUEST=GetCapabilities&	*Request Name*

The URL need not be the same as that for `GetMap`. Therefore, you could arrange for another server to provide this functionality. You must be able to provide any `GetMap` service that you declare in the XML file. This file is to be returned with mime type set to "`text/xml`".

Because each WMS server is independent and is likely to have different kinds of information for which it can produce maps, a WMS server must be able to provide a machine-readable (machine-parseable) description of its capabilities. This "Service Metadata" `Capabilities.xml` file enables WMS clients to formulate valid requests and enables the construction of searchable catalogs that can direct WMS clients to particular WMS servers.

The `GetCapabilities` operation needs to retrieve a complete listing of which interfaces a map server supports and what map layers it can serve in response to the invocation of the map interface. The `GetCapabilities` operation provides WMS clients of WMS servers with the following functionality:

- All interfaces a WMS server can support.
- Image formats it can serve (e.g., GIF, JPG, PNG).
- List of spatial reference systems available for delivery of map data from the WMS server.
- List of all exception formats for return of exception that are available from the WMS server. Inclusion of a value for this attribute is optional.
- List of all the vendor specific capabilities (or properties) that are available for modifying or controlling actions of a particular WMS server, with the current value of each capability. Inclusion of a value for this attribute is optional.
- List of one or more map layers available from a particular WMS server. Inclusion of a value for this attribute is optional.
- Whether a WMS server supports the optional FeatureInfo interface.

1.3.2. Implementing GetCapabilities: Request Parameters

Note: See also Chapter 3, which includes specific recipes on WMS server configuration utilizing the Capabilities XML and DTD documents.

`GetCapabilities` is invoked by a WMS client to get a complete listing of what interfaces a WMS server supports and what map layers it can serve in response to invocations of the `GetMap` request. The listing also contains information about whether a WMS server supports the optional FeatureInfo operation. The listing is returned as an XML document which conforms to the Capabilities DTD. See WMS 1.1.1 Annex A.1 for WMS Capabilities DTD and Annex A.2 for a sample WMS Capabilities XML.

Review WMS 1.1.1 specification, section 7.1 on the `GetCapabilities` request, especially 7.1.3 for the overview of the `GetCapabilities` request parameters. In essence, Table 1 below shows the required and optional request parameters for implementing the `GetCapabilities` functionality.

Request Parameter	***Required/ Optional***	***Description***
VERSION=version	*O*	*Request version*
SERVICE=WMS	*R*	*Service Type*
REQUEST=GetCapabilites	*R*	*Request name*
UPDATESEQUENCE=string	*O*	*Sequence number or string for cache control*

Table 1: Getcapabilities Request Parameters

The following links can be used to access tools for compiling WMS server metadata:

- Web Mapping Testbed XML Validator[16]. Check your XML via HTTP. Type URL[*] of your XML document for manual validation

- Brown University's XML Validator[17]. Brown University's validator allows file uploads.

1.3.3. GetCapabilities Response: Capabilities XML Document

The response to a `GetCapabilities` request is a Capabilities XML document (XML file as per Capabilities DTD or MIME type in case of HTTP) conforming to the Schema given in the OWS Service Metadata XML IPR, composed of four main sections depicted in Figure 9, below.

[16] *Web Mapping Testbed XML Validator http://www.digitalearth.gov/wmt/xml/validator.html*

[17] *Brown University's XML Validator: http://www.stg.brown.edu/service/xmlvalid/) to check a local or firewalled XML file.*

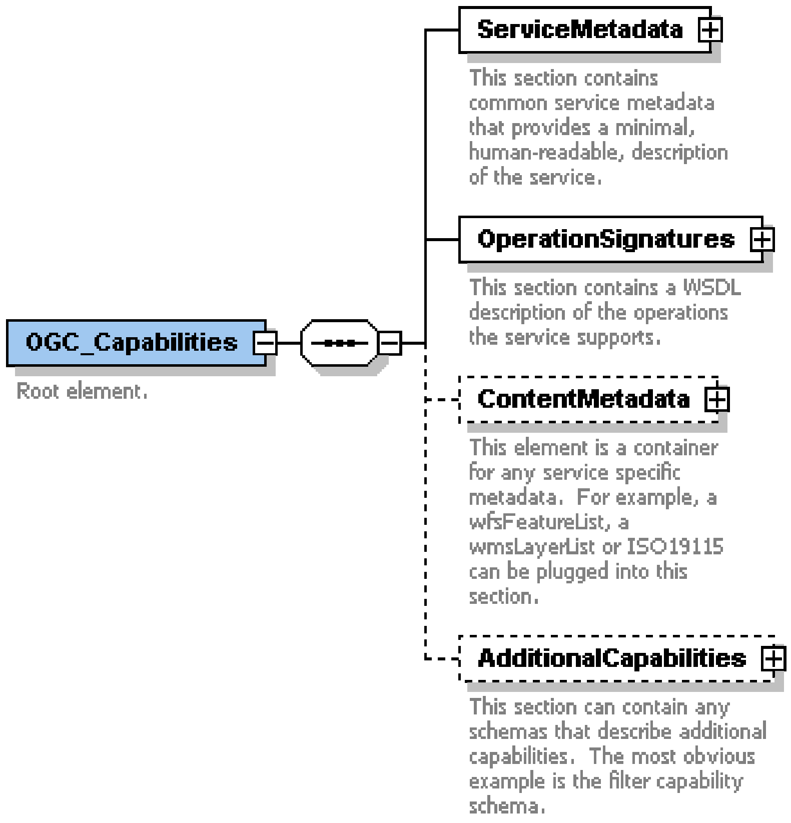

ServiceMetadata
This section contains common service metadata that provides a minimal, human-readable, description of the service.
OperationSignatures
This section contains a WSDL description of the operations the service supports.
OGC_Capabilities
Root element.
ContentMetadata
This element is a container for any service specific metadata. For example, a wfsFeatureList, a wmsLayerList or ISO19115 can be plugged into this section.
AdditionalCapabilities
This section can contain any schemas that describe additional capabilities. The most obvious example is the filter capability schema.

Figure 9: OGC_Capabilities Top-level Element

Review WMS 1.1.1 Specification section 7.1.4 on the `GetCapabilities` response and section 7.1.5 on the output formats.

The `GetCapabilities` request output (the Capabilities XML file) can be relatively large if the WMS server has a lot of map layers. This is especially true if the layers are unrelated. However, the nesting and inheritance rules can usually be used to minimize duplication of information. Also, it is possible to parameterize the layers and use the Dimension and Extent elements to reduce the size (version WMS 1.1 and up allow for this).

For instance, if the layers represent multiband data, a new layer for each band is not needed, but instead one layer with a `DIMENSION=band` declaration. Then, a `BAND=1` can be included as part of the request for that layer. This is not limited to bands, but to any other dimensions (e.g. time, elevation, etc.).

Note that a WMS server does not need to dynamically generate the capabilities response. Also, the Web server, in addition to the WMS server, can provide the capabilities response on a WMS server's behalf. In addition, the WMS Specification defines only a limited set of parameters for requesting maps. In order to be able to generate more sophisticated maps adapted to location, context or user interest, the WMS Specification must be extended by a range of parameters. These enhancements include the possibility not only to specify predefined styles for each feature type, but to specify a whole XML sub tree.

1.4. Serving a Map: The GetMap Request

1.4.1. Purpose of GetMap

The `GetMap` request returns a map image whose geospatial and dimensional parameters are well defined. The map operation of the `GetMap` request is invoked by a client to get a rectangular set of pixels. These pixels contain a picture of a map covering a geographic area or a set of graphic elements that lie in a geographic area. The `GetMap` request allows the WMS client to specify distinct layers, the spatial reference system (SRS), the geographic area, and other parameters describing the returned map format. Upon receiving the `GetMap` request, a WMS server will either satisfy the request or throw an exception in accordance with the exception instructions contained in the `GetMap` request.

The WMS server must be able to deliver a map via HTTP upon receiving a WMS client request such as the following:

```
http://www.airesip.org/wms/process.cgi?REQUEST=GetMap&
FORMAT=image/gif&WIDTH=640&HEIGHT=480&LAYERS=temperature
&SRS=EPSG:4326&BBOX=-110.,40.,-80.,30.&&VERSION=1.1.0
```

In this hypothetical example, `www.airesip.org` is the server hostname, `wms/` is its directory path, and `process.cgi` is the name of the CGI script processing the WMS client requests. This CGI script knows how to respond to the WMS request. You can choose the directory path and file names. A question mark is appended after the script name to separate it from the parameter list. The parameter list consists of parameter name=value assignments separated by an ampersand (&).

Parameters may appear in any order. Parameter names are not case-sensitive; therefore, `height=480` and `HEIGHT=480` are identical requests. However, you may choose to interpret parameter values as case-sensitive. No spaces appear anywhere in the request string.

The `GetMap` request enables the creation of a network of distributed map servers from which WMS clients can build customized maps. When two or more maps are produced with the same Bounding Box, Spatial Reference System, and output size, the results can be accurately layered to produce a composite map.

The use of image formats that support transparent backgrounds allows the lower layers to be visible. Furthermore, individual map layers can be requested from different WMS Servers.

1.4.2. Implementing GetMap: Required Parameters

When invoking a `GetMap` request, a WMS client can specify the information to be shown on the map: one or more "Layers", possibly the "Styles" of those Layers, what portion of the Earth is to be mapped (a "Bounding Box"), the projected or geographic coordinate reference system to be used (SRS), the desired output format, the output size (Width and Height), and background transparency and color.

The required fields as well as the optional fields of a `GetMap` request are presented in Table 2, below.

Request Parameter	**Required/ Optional**	**Description**
VERSION=version	*R*	*Request version*
REQUEST=GetMap	*R*	*Request name*
LAYERS=layer_list	*R*	*Comma-separated list of one or more map layers Optional is SLD parameter is present*
STYLES=style_list	*R*	*Comma-separated list of one rendering style per requested layer. Optional if SLD parameter is present*
SRS=namespace:identifier	*R*	*Spatial Reference System*
BBOX=minx,miny,maxx,maxy	*R*	*Bounding box corners (lower left, upper right) in SRS units*
WIDTH=output_width	*R*	*Width in pixels of map picture*
HEIGHT=output_height	*R*	*Height in pixels of map picture*
FORMAT=output_format	*R*	*Output format of map*
TRANSPARENT=TRUE/FALSE	*O*	*Background transparency of map* *(default-FALSE)*
BGCOLOR=color_value	*O*	*Hexadecimal red-green-blue color value for the background color (default=0xFFFFFF).*
EXCEPTIONS=exception_format	*O*	*The format in which exceptions are to be reported by the WMS (default=SE_XML)*
TIME=time	*O*	*Time value of layer desired*
ELEVATION=elevation	*O*	*Elevation of layer desired*
Other sample dimension(s)	*O*	*Value of other dimensions as appropriate*
Vendor-specific parameters	*O*	*Optional experimental parameters*
The following parameters are used only with Web Map Services that support the Styled Layer Descriptor specification [3].		
SLD=styled-layer_descriptor_URL	*O*	*URL of Styled Layer Descriptor (as defined in SLD Specification).*
WFS=web_feature_service_URL	*O*	*URL of Web Feature Service providing features to be symbolized using SLD*

Table 2: `GetMap` Request Parameters

In addition to the information in the WMS Specification, below are some issues and their workarounds, encountered when implementing some of the `GetMap` parameters.

1.4.2.1. SRS and BBOX

When developing a WMS client where it makes a `GetCapabilities` request to allow the user to select layers for mapping, a problem might be encountered when the WMS client will have to deal with WMS servers that specify multiple projections in their SRS.

The SRS parameter is a multiple list of spatial reference systems. The list for a named layer is an additive list is inherited from an ancestor layer. Note that WMS Specification version 1.1.1 allows (and prefers) multiple SRS values to be enclosed in separate `<SRS>` elements. That is, instead of:

```
<SRS>EPGS:4267
EPSG:4326 EPSG:32747
EPSG:32748
EPSG:32749</SRS>
```

it is preferred to have:

```
<SRS>EPGS:4267</SRS>
<SRS>EPSG:4326</SRS>
<SRS>EPSG:32747</SRS>
<SRS>EPSG:32748</SRS>
<SRS>EPSG:32749</SRS>
```

It is important to realize that the WMS Specification does not require (but only recommends) each named layer to have a "BoundingBox" for every SRS - "Layers may have zero or more `<BoundingBox>` elements" (WMS 1.1.1 section 7.1.4.5.7 BoundingBox). Some WMS servers are able to transform on demand to a large number of SRSes. Instead of requiring such WMS servers to compute and advertise a bounding box for every layer in every SRS, the WMS Specification lets them advertise as few as one SRS.). Only `LatLonBoundingBox` (EPSG:4326) is required on each Layer, either stated or inherited from an ancestor layer. It is recommended that if the data are stored in some other SRS then the `BoundingBox` for that native SRS also be given.

According to the above explanation of the WMS Specification on "`BoundingBox`," the following is valid. A WMS server may define each layer as a child to one parent coverage. This coverage specifies multiple projections (within the one SRS) which are then inherited by the child layers. This can be done without having anywhere in the structure a "`BoundingBox`" defined for any of the specific projections. This way, it is possible to specific map areas of layers by using the "`LatLonBoundingBox`".

To further explain and minimize the confusion that might arise when reading the WMS Specification on "`BoundingBox`" (WMS 1.1.1 section 7.1.4.5.7) where it states that WMS servers may advertise fewer BoundingBoxes than SRSes and should advertise at least the native SRS, consider the following. It may seem that with a thin WMS client the only image it can receive from that WMS server is one that is in lat/lon, since the domain of coordinates for various projections can vary widely. Trying to make a wild guess in this case, is unpractical.

The following points clarify this situation:

- If the WMS client wants maps in EPSG:4326 and the WMS server supports that SRS, there is no problem.
- If the WMS client wants a map in an SRS for which the WMS server has advertised a `BoundingBox`, there is no problem.
- If the WMS client is already displaying a map from some other WMS server in an arbitrary SRS, with a bounding box valid in that SRS, and adds a map from another WMS server that

supports that SRS but does not give a bounding box, then the WMS client will use the `BBOX` of its current map in order to overlay the two maps. The overlay result will depend on the area covered by each dataset, but the WMS server should not give out error messages if the requested `BBOX` does not overlap the data - the WMS server should simply send a blank map. The WMS client can estimate the likelihood of overlap between the two servers by inspecting LatLonBoundingBox. Hence, there is no problem.

- If the WMS client wants to request a new map in an arbitrary SRS without any advance knowledge of valid bounding box values in that SRS, then a wild guess is its only option. However, it's very unlikely that such a WMS client will ever have to make an unreasonable "wild guess" as to what coordinates to put into a `BBOX` request. Rather, its `BBOX` requests are very likely to be sensible numbers (e.g., for UTM meters, on the order of x=500,000, y=4,000,000 for northern US). This is because these requests are based upon the WMS client's current UTM viewpoint, which is based on the previous one. In addition, WMS clients usually have a starting viewpoint that gets set by either of the following: (a) some pre-loaded local data; (b) a default startup setting (*.ini file); (c) the URL that invoked the (HTML or Java applet) WMS client; or (d) some user "context" or "configuration" file.

1.4.2.2. BBOX and Projections

A problem may arise when a bounding box that produces a rectangular image in one projection system produces a differently shaped polygon in another projection system.

An appropriate interoperable solution is to require that all WMS servers draw their data into the `BBOX` and SRS of the map request. Any areas that don't have data within the specified `BBOX` get transparent pixels. WMS server developers can declare it to be invalid for a WMS server to return an error of the sort "`BBOX` is out of range". Some WMS servers generate "out of range" errors when they really shouldn't have to. This way, a WMS client can handle the different-SRS bounding box by either way of including it, or ignoring it all together - both ways will make the results valid.

Note that the WMS Specification states that a map image be stretched so that the requested bounding box fits into the requested image. (WMS 1.1.1 section 7.2.3.8). It is expected that without having a smart WMS client, a client request to the WMS server for converted coordinates followed by a subsequent request for an image would necessarily result in a distorted image.

Furthermore, WMS 1.1.1 section 7.2.3.8 contains the note: "Map distortions will be introduced if the aspect ratio WIDTH/HEIGHT is not commensurate with X, Y and the pixel aspect. WMS client developers are cautioned to minimize the possibility that users will inadvertently request or unknowingly receive distorted maps."

Handling this situation depends on what kind of WMS client is being utilized. With a WMS client that can only overlay one image on top of another, this is a non-issue. Presumably the WMS client will have specific values for its viewing window and it just passes these values to the WMS server. No image-parameter conversions are necessary. On the other hand, for a WMS client that can reproject the images it receives back from a WMS server (for example, the CubeWerx cascading map server where it acts as a WMS client), it can avoid requesting non-stretched images by adjusting its request to use square pixels. This is a matter of computing: resX = resY = sqrt(resX * resY) to get the adjusted square-pixel resolution (with equal-area pixels), and then adjust the bounding box to align with it. It is also important to remember that the WMS Specification also requires WMS servers to support non-square pixels.

1.4.3. GetMap Response

Review WMS 1.1.1 specification, section 7.2.5 on the `GetMap` response.

The output of a `GetMap` request is a single map whose type corresponds to the `FORMAT` parameter in the request. In the case of HTTP, for example, if the request included `FORMAT=JPEG`, then the returned object must be a JPEG image with a MIME type of image/jpeg. In the case of non-picture requests (i.e., graphic elements or feature data), the parameters `WIDTH`, `HEIGHT`, `TRANSPARENT` and `BGCOLOR` are not relevant and may be omitted. `WIDTH` and `HEIGHT` are mandatory when the output format is an image.

`GetMap` returns an image as specified in the format parameter of the request (GIF, JPEG, PNG). A `GetMap` request returns nothing with invalid HTTP requests or by access violations. A `GetMap` response returns an XML structure containing error information if the WMS server detected an error and the `EXCEPTIONS` parameter was set to XML. In the case of HTTP, the MIME type of the returned XML will have the following format:

```
<WMTException version= "1.1.1>
error information
</WMTException>
```

1.4.4. Exception Handling

If the `EXCEPTIONS` parameter is set to `BLANK`, the WMS server returns an object of the type specified in `FORMAT`, whose content is uniformly off. In the case of an image format such as GIF or JPEG, the returned object contains only pixels of color specified in the background color. When `TRANSPARENT=true`, pixels with the background color are transparent. In other formats, such as vector based formats, no vectors are returned.

1.4.5. Cascading WMS Servers

When two or more maps are produced with the same Bounding Box, Spatial Reference System, and output size, the results can be accurately layered to produce a composite map. The use of image formats that support transparent backgrounds allows the lower Layers to be visible. Furthermore, individual map Layers can be requested from different Servers. The WMS `GetMap` operation thus enables the creation of a network of distributed map servers from which WMS clients can build customized maps.

A particular WMS provider in a distributed WMS network need only be the steward of its own data collection. This stands in contrast to vertically-integrated web mapping sites that gather all of the data in one place that is to be made accessible by their own private interface.

A "Cascading Map Server" is a WMS server that behaves like a client of other WMS servers, and like a WMS server to other WMS clients. A cascading map server reports the capabilities of other WMS server(s) as its own and aggregates the contents of several distinct WMS servers into one service. In most cases, the cascading map server can work on different WMS servers that cannot serve particular projections and formats themselves.

See Chapter 2 to read how participants of WirelessInfo Project see the importance of WMS servers to establish cascading WMS servers. For projects such as the WirelessInfo Project, cascade servers play the role of hubs of WMS networks and offer possibility to create gateways for clients (mobile devices in the case of WirelessInfo Project), which use different sources of geographic information.

Cascading means that one WMS server can compose visualized layers from various WMS servers and play the role of a gateway to the distributed geodata sources for clients. The above-mentioned ability is demonstrated in the following picture:

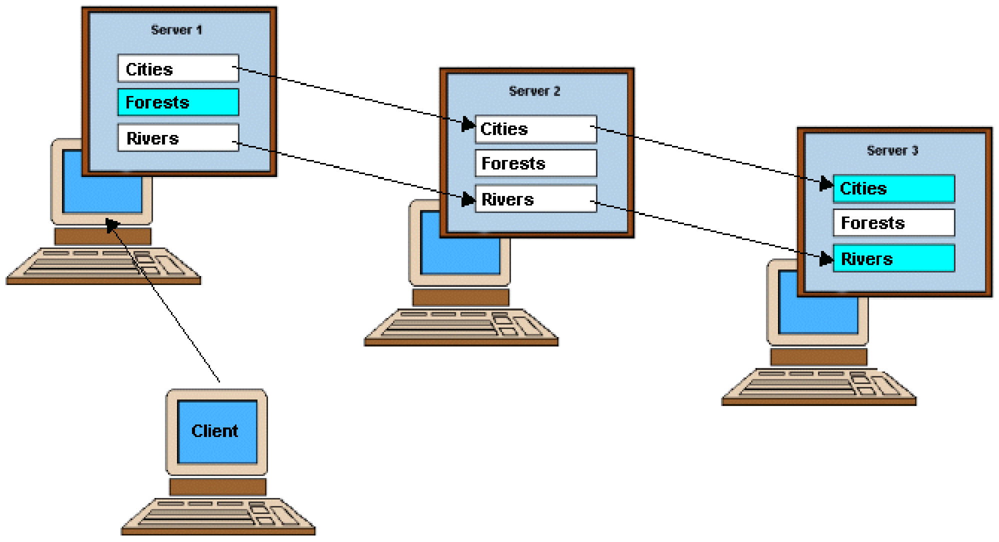


Figure 10: WMS Cascading Servers (courtesy of WirelessInfo)

Figure 10 shows Server 2 playing the role of the front-end for users. Server 2 only maintains the database named *Forests*, but it offers visualization of all accessible datasets, even the ones located on the other two servers (Server 1 and Server 2). This way, Server 2 plays the role of a gateway among the other WMS servers, which means that it does not have to maintain the other dataset locally. Being a cascading WMS server, Server 2 is able to ask the other servers in the network for the appropriate dataset when needed.

1.5. Optional Operation: The GetFeatureInfo Request

1.5.1. Purpose of GetFeatureInfo

The `GetFeatureInfo` request returns information about particular features shown on a map. If a WMS server supports this operation, its maps are said to be "queryable," and a WMS client can request information about features on a map by adding to the map URL additional parameters specifying a location (as an X, Y offset from the upper left corner) and the number of nearby features about which to return information.

The `GetFeatureInfo` operation is designed to provide WMS clients with information about a feature in a map image returned by a previous `GetMap` request. In other words, the map is identified by including all the information contained in the `GetMap` request. If a previous `GetMap` request is not repeated correctly in the `GetFeatureInfo` request, the results are undefined and will cause an exception.

`GetFeatureInfo` request embeds the bulk of `GetMap` request, adding parameters to define which layer to query. The WMS server can return a text/HTML/xml page, an Image or even a Word document in response to `GetFeatureInfo` request. The response may even contain attribute information, or the selected feature(s) in GML format.

The use case for `GetFeatureInfo` is that a user sees the response of a `GetMap` request and chooses a point on that map for which to obtain more information. The basic operation provides the ability for a client to specify which pixel is being asked about, which layer(s) should be investigated, and what format the information should be returned in. The actual semantics of how a WMS server decides what to return more information about, or what exactly to return, is left up to the WMS provider.

1.5.2. Implementing GetFeatureInfo: Required Parameters

Review WMS 1.1.1 specification, section 7.3 to learn about the `GetFeatureInfo` request, especially section 7.3.3 on the required and optional request parameters. Table 3, below, shows the required and optional parameters for `GetFeatureInfo` request.

Request Parameter	Required/ Optional	Description
VERSION=version	*R*	*Request Version.*
REQUEST=GetFeatureInfo	*R*	*Request name.*
<map_request_copy>	*R*	*Partial copy of the Map request parameters that generated the map for which information is desired.*
QUERY_LAYERS=layer_list	*R*	*Comma-separated list of one or more layers to be queried.*
INFO_FORMAT=output_format	*O*	*Return format of feature information (MIME type).*
FEATURE_COUNT=number	*O*	*Number of features about which to return information (default=1).*
X=pixel_column	*R*	*X coordinate in pixels of feature (measured from upper left corner=0).*
Y=pixel_row	*R*	*Y coordinate in pixels of feature (measured from upper left corner=0).*
EXCEPTIONS=exception_format	*O*	*The format in which exceptions are to be reported by the WMS (default-application/vnd.ogc.se_xml).*
Vendor-Specific Parameters	*O*	*Optional experimental parameters.*

Table 3: `GetFeatureInfo` Request Parameters

1.5.3. GetFeatureInfo Response

The WMS server returns a response according to the requested INFO_FORMAT (if the request is valid, or issues an exception otherwise). The actual response is at the discretion of the WMS provider, but it pertains to the feature(s) nearest the (X,Y) specified.

In general, the `GetFeatureInfo` outputs an HTML page or an image. However, the use of MIME as the return format tells the WMS server to package the result in any way it wants, but the return object must be properly accompanied with a MIME type describing the content.

1.6. Connecting drivers to OGC WMS Services

(Acknowledgment: Lars Bernard, Institute for Geoinformatics (IfGI), Muenster)

'Connecting,' in the context of WMS, means that just implementing the WMS Specification will hardly realize a network of interoperating WMS. Issues with things such as the following need to be resolved first -- which of the following are supported: SRS, spatial extents, appointment on scaling hints, and metadata (to describe the service in a registry).

Chapter 2, Example 3 under User Experience, reflects the experience of the computer science GIS group at the Harvard Division of Continuing Education (DCE) in setting up and testing distributed WMS servers. Their distributed geospatial Web environment considers the deployment of map services powered by different spatial engines, some from vendors and others from open source.

1.7. On What Technologies Does the WMS Specification Depend?

1.7.1. Extensible Markup Language (XML)

The WMS Specification is based on W3C's XML specification[18]. Understanding it requires understanding the basics of XML. Overall, you should familiarize yourself with the following:

- Creating an XML document instance
- Rules for XML well-formalness
- Distinction between well-formed and valid documents
- Internal and external subsets
- Elements and attributes
- Entities (parameter, general, internal and external, parsed and unparsed)
- Notations
- Parsing/Validating a DTD and XML document

XML is best thought of as a language or encoding for data description. More correctly, XML is a language for expressing data description languages. XML however, is not a programming language. There are no mechanisms in XML to express behavior or to perform computations. That is left for other languages such as Java and C++. XML provides a means of describing (marking up) data using user defined tags. Each segment of an XML document is bounded by starting-tags and end-tags.

Almost every data model that can be expressed in an XML document can be represented as a tree, where the root is the top element and the branches are the other elements nested in it. Elements have relationships with each other that will be familiar to anyone who has worked with object-oriented

[18] *W3C XML Specification: URL: http://www.w3.org/TR/2000/REC-xml*

programming. In terms of the tree, as the three branches out, the leaves and twigs become *children* of the branches. Elements that are on the same level in the documents are called *siblings*.

Each element has an element type name and zero or more attributes, and each attribute consists of a name and a value. Each element consists of two tags: a start tag and an end tag. An element can have attributes, which are written inside the start tag. Each attribute has a name and a value.

There can be multiple attributes of one element, and the content of the element and the values of the attributes are to a very large degree interchangeable, as follows:

```
<Feature>
.... more XML descriptions ...
....
</Feature>
```

The following sections describe technologies and tools that are parts of the XML family that are relevant to Web mapping:

1.7.2 DTD & XML Schema

1.7.3 XSL (XSLT, XPath, etc)

1.7.4 XPointer, Llink

1.7.5 XMLNS (Namespaces)

1.7.6 DOM and SAX

1.7.2. Validate XML - DTD (Document Type Definition) & XML Schema

You should familiarize yourself with the following through online tutorials[19] .

- Read and use a DTD
- Write a DTD:
- Internal and external subsets
- Elements and attributes
- Exceptions
- Entities (parameter, general, internal, and in attributes)
- Data content types (PCDATA, RCDATA, and CDATA)
- Notations
- Markup minimization

[19] XML DTD Tutorial: http://www.xmlfiles.com/dtd/, XML Schema Tutorial Part 1 – Structures: http://www.w3.org/TR/xmlschema-1/, XML Schema Tutorial Part 2 – Datatypes: http://www.w3.org/TR/xmlschema-2/

- Marked sections
- Understand the differences between XML DTDs and the XML Schemas

The valid tag names and the XML markup in general are determined or defined by the DTD. Which tags can appear enclosed within an opening and closing tag pair is also determined by the DTD. The names of the elements and the attributes are also constrained by the DTD in name, and in some cases, in terms of the values that the attributes can assume; and, what data types they can contain, and the structure into which they can be combined are together called a document type and are defined in DTD. The DTD is either contained in a `<!DOCTYPE>` tag, contained in an external file and referenced from a `<!DOCTYPE>` tag, or both.

Example snippet from Capabilities DTD:

```
<!ELEMENT GetCapabilities (Format+, DCPType+)>
<!ELEMENT GetMap (Format+, DCPType+)>
<!ELEMENT GetFeatureInfo (Format+, DCPType+)>
```

XML is typically read by an XML parser. All XML parsers check to ensure the data is well formed so that data corruption (e.g., a missing closing tag) cannot pass undetected. Many XML parsers are also validating, meaning that they check that the document conforms to the associated DTD. Using XML it is comparatively easy to generate and validate complex hierarchical data structures. Such structures are common in geographic information applications.

DTD drawbacks and limitations:

- Another syntax to learn (not supported by XML tools).
- No object-oriented extensibility (inheritance).
- Limited constraints for validity-check: databases, value-ranges.

1.7.3. Transform and Format XML:XSL (Transforming the Web)

You should familiarize yourself with the following through online tutorials[20].

- Use XSLT engines to transform XML to other data formats, including HTML, other user-designed XML, and other tagged or labeled data formats.
- Draw the logical tree represented by sequential XML data and understand how to navigate that tree using Xpath.
- Use expressions, location paths, and patterns to select parts of XML documents and control their transformation.
- Use XSLT to create complex Web pages.
- Create multi-file XSLT stylesheets.
- Do basic calculations in XSLT for selection of content to output and to create output data.

[20] *XSLT http://www.xml.com/pub/a/2000/08/holman/index.html.*

The original focus of XML was to provide a means of describing data separate from its presentation, especially in the context of the Web. The biggest change to the computing industry that XML brings is that it enables the programmer (or even system designer or Web site designer) to declare the rules for how information should be processed and handled. The XML application itself describes the rules for how information should be structured and how the markup should be constructed. The representation of the information is shaped by the use of a markup vocabulary language determined by the designer, which also declares the data types of the markup vocabulary. This situation enables tools to constrain the creation of an instance of information and enable users to validate a properly created instance of information against a set of constraints.

XSL is a language for expressing stylesheets. It consists of:

- XSL Transformations (XSLT): a language (rules declarations) for transforming XML documents into other data formats.

- The XML Path Language (XPath): defines the structure, content, and semantics of XML documents. It identifying parts of XML documents.

Note that since many tools (e.g. MS IE 5.0) were developed before the XSLT label had stuck, XSL is still often used when only XSLT is intended. For discussion presented here, XSLT is of concern.

XSL is a fairly simple language. It provides a powerful syntax for expressing pattern matching and replacement. It is declarative. It is easy to read what the XSLT says to do. The user does not get to see how it is accomplished. Using XSLT's companion specifications (XPath and XQL), the developer can specify some very powerful queries on an XML document. Furthermore, XSLT provides the ability to call functions in another programming language, such as VBScript or Java, through the use of Extension Functions. This means that XSL can be used to do the querying and selection, and then call out to Java or another language to perform needed computation or string manipulation. For simple tasks, XSLT provides built in string handling and arithmetic capabilities.

XSLT is similar to other forms of content transformation in that it deals with the documents as trees of abstract nodes. XSLT, or rather Xpath, enables you to identify structures and to make as many passes as required over them, modifying the structures in the source information (or rather, the markup of the information). The information being transformed can be traversed in any order needed and as many times as required to produce the desired result. The algorithms are declared in the XSLT code and are handled by the XSLT processor. It only works with the markup as abstract nodes, not with the source data or with the semantics on the markup.

The generic XML processor has no idea what is meant by the XML, and the XML markup does not usually include formatting information. The information in an XML document might not be in the form desired to present it, but this information has to be described somewhere else.

The XSLT processor handles the mechanics of the operations. High-level functions such as sorting and counting are available when required as functions in the language. The XSLT processor handles low-level functions such as memory-management, node manipulation, how nodes are node traversed and created, and garbage collection. In other words, the XSLT programmer does not have to think about the mechanics of the operations. This also applies when considering the mechanics of the presentation, which can be left to the browser and the stylesheet processor.

XSLT does not actually transform the original document. It works with a copy of the source tree, which is transformed into the result tree. In other words, the XSLT transformation sheet is an instruction for how to turn the source document into the result document. It only affects the parts that are to be transformed.

An XSLT transformation sheet consists of a set of templates (the instructions to the processor) that are matched to the source document (by addressing them by using Xpath). When a match is found, the matching part of the source document will be transformed to the resulting document.

Because an XSLT stylesheet is an XML document, it is no harder to produce than creating any other XML document. Any text editor will suffice, although it is easier to use an XML editor (to help you with formatting).

1.7.4. Navigate in XML: XPointer, Xlink

XML Pointer Language (XPointer)[21]*:* the language to be used as the basis for a fragment identifier for for any URI reference that locates a resource. XML Pointer Language:

- Is a generalization of Xpath
- Can refer to any part of the document (XPath refers only to logical parts such as Elements
- Allows for String-Matching

XML Linking Language (XLink)[22]: a language that allows elements to be inserted into XML documents in order to create and describe links between resources. XML Linking Language:

- Is a generalization of HTML-Links
- Provides Multidirectional Links
- Enables links be created outside of locator and resource documents

1.7.5. Managing XML-Extensibility: Namespaces

Since there is no central authority that controls the birth of new XML applications, there is a risk that two or more applications will use the same element name to mean different things. If this situation happens, the XSLT processor cannot tell the difference between the names. Moreover, applications use the element type name to determine how to process the element. In a distributed environment like the Web, names must be globally unique, otherwise one name might accidentally be used for different purpose. To avoid naming conflicts, XML supports a simple namespace mechanism. In an XSLT stylesheet, every element and attribute belongs to an XML namespace.

Namespaces are defined in the W3C recommendation *Namespaces in XML*[23]. In a document, they are given by using the colon-delimited prefixes. The prefix is significant when comparing element names within a document. Therefore, *xsl:template* and *template* are different. However, the prefix can vary between documents. The significant part is the association of a prefix string with a URI. That is the function of the xmlns: attribute in the style sheets. For instance, the namespace declaration:

```
"xmlns: xsl=http://www.w3.org/TR/WD-xsl"
```

associates the namespace prefix xsl with the URI that follows it: `"http://www.w3.org/TR/WD-xsl"`.

Because the prefix is arbitrary, instead, it could be:

[21] *Xpointer: http://www.w3.org/TR/xptr/*
[22] *XLink: http://www.w3.org/TR/xptr/*
[23] *W3C recommendation Namespaces in XML: http://www.w3.org/TR/WD-xml-names.*

```
"xmlns:anyname=http://www.w3.org/TR/WD-xsl"
```

The names in an XML document can be associated with a URI that is globally unique. The name that is used inside the document is called a *local name*, and the name plus the associated URI is called a *qualified name*. That URIs are unique, by virtue of their association with the DNS, means that the qualified name and the XML elements will both be globally unique. However, the URIs are actually only used to identify the elements names. There is no requirement that there should be a description, schema, DTD, or aything else behind it.

1.7.6. Parse XML: DOM and SAX

XML documents do not have to be structured in the order that they should be displayed or processed. The order and rendering can be manipulated by programs or scripts via the Document Object Model (DOM) of the W3C or via SAX.

Document Object Model (DOM)[24] is an object-oriented, platform- and language-neutral interface that allows programs and scripts to dynamically access and update the content, structure and style of documents.

- Tree of objects is created completely in memory
- No incremental processing of documents
- The Simple API for XML Parsing (SAX) is an event-based interface for (Java) XML parsers.
- Memory-efficient stream-based parsing, (almost) no document-size limit
- No W3C-Standard, very popular for Java-parsers

Most XSLT processors accept SAX events generated by some XML processors as input, and some accept DOM objects that might be generated by other XML parsers or created from scratch. The SAX events or DOM objects are used as the basis for the node trees that follow the Xpath/XSLT data model.

The XSLT processing can be done in the browser, but also in the server. It has turned out to be done more efficiently in the server, especially for formats that are to be displayed in devices that are less capable in terms of display and processor.

Conceptually, the XSL processor begins at the root node in the source tree and processes it by finding the template in the stylesheet that describes how that element should be displayed. Each node is then processed in turn until there are no more nodes left to be processed. This situation can get more complicated when you have each template specifying which nodes to process - some nodes might be processed more than once and some might not be processed at all.

MSXML, for example, provides both an XML parser and an XSL processor in one Windows DLL. Internet Explorer (version 5 or higher) is an application that utilizes MSXML to handle the processing of XML documents that have been associated with a stylesheet by means of processing instructions in the XML.

The XML processor reads the XML input. The XSLT processor performs the actual XSL transformations. It will have to use an XML processor to read the source XML and the XSLT. An XML processor (sometimes called a parser) reads a source XML file and identifies the syntactic units (elements, attributes, and text content).

[24] DOM parser: http://www.w3.org/DOM/

An XSLT processor takes a stylesheet and applies it to the tree representation of a source XML document (produced by an XML parser) and generates a tree representation of an output XML document. The product of the XSLT processing can be composed not only of XML but also HTML, or text.

1.8. Implementing WMS Compliance

1.8.1. Writing a WMS-Compliant Translator for a Map Server

A translator, or a wrapper script, can be build that takes WMS requests and translates them into map requests for a non-WMS compliant map server.

Look into Chapter 2, in the XSL/XSLT stylesheet example section to see a XSL script that takes in a WMS `GetMap` request and transforms it into another XML dialect understood by the map server.

1.8.2. Making a Map Server WMS-Compliant

Making a map server WMS (the server instance) compliant is mostly about implementing the Capabilities XML file. See Chapter 3 for recipes on how to do this. The lat/lon **deegree** and UMN MapServer recipes are good examples of this.

Usually a map server needs to be compiled so that the various WMS parameters are set (described earlier and/or referenced to the WMS 1.1.1 specificaiton). Knowing that the server can produce a valid XML `GetCapabilities` response, the `GetMap` request can be utilized. Simply adding "`VERSION=1.1.0&REQUEST=GetMap`" to your server's URL should generate a map with the default map size.

For example, in the case of the UMN Mapserver, a mapfile - regular MapServer mapfile - was made in which some parameters and some metadata entries are mandatory (some parameters are the map level and others at the layer level). Most of the metadata is required in order to produce a valid `GetCapabilities` output for the WMS client.

1.8.3. Making a Web Client WMS Compliant

Making a Web client WMS compliant is mostly about implementing the `GetMap` request and its parameters. See Chapter 3, for the WMS client setup with UMN MapServer, which is a good example of this.

Harvard University user experience in Chapter 2 discusses the use of the Proj4 Cartographic Projection Library, necessary for WMS Clients.

1.9. WMS Web Applications Development Technologies

In developing Web applications, in order to build customized content based on the user's input and to enhance the simplicity of using HTTP to request information, there are many technologies that can be used. These are described below.

1.9.1. CGI and Servlets

1.9.1.1. CGI

The Common Gateway Interface (CGI) is a standard for interfacing external applications with information servers, such as HTTP or Web servers. A plain HTML document that the Web client retrieves is static, which means it exists in a constant state: a text/html file that doesn't change. A CGI program, on the other hand, is executed in real-time, so that it can output dynamic information.

OGC's WMS Implementation Specification is a standard vocabulary for basic Web mapping services that is based on CGI. The CGI request is still a URL, but at some point in the URL, there is a question mark, and everything after the question mark is a list of key/value pairs. A CGI program can be written in any language (C/C++, PERL, VB, etc.) that allows it to be executed on the system.

One of the problems of CGI is the fact that each incoming HTTP request gets a new process, creating a burden on the server.

With MapServer, it is the "mapserv" CGI program that knows how to handle WMS requests. So setting up a WMS server with MapServer involves installing the mapserv CGI program and setting up a mapfile (files which defines map object) with appropriate metadata in it. This is covered in Chapter 3, under Recipe 3.

1.9.1.2. Servlets

Servlets are generic extensions to Java-enabled servers. Their most common use is to extend Web servers, providing a very secure, portable, and easy-to-use replacement for CGI. A servlet is a dynamically loaded module that services requests from a Web server. It runs entirely inside the Java Virtual Machine (JVM). Because the servlet is running on the server side, it does not depend on Web browser compatibility.

1.9.2. ASP versus JSP

Technologies such as Active Server Pages (ASP)[25] and JavaServer Pages (JSP), make Web pages more like true user interfaces to powerful, customizable, distributed programming resources.

Both the ASP and JSP technologies follow the same model of separating programming logic from page design through the use of components (scripting blocks) that are called from the page itself. HTTP requests and responses are available as well-established objects you can get to easily from within the script blocks. Both also provide developers an easier and faster alternative to creating Web applications using CGI scripts.

The biggest difference between JSP and ASP technologies lies in the approach to the software design. ASP is based on ISAPI whereas JSP is implemented as a part of J2EE. JSP technology is designed to be both platform and server independent, created with input from a broader community of tool, server, and database vendors. In contrast, ASP is a Microsoft technology that relies primarily on Microsoft technologies.

ASP consists of a single DLL (asp.dll) which generates dynamic content when an ASP page with server-side script combined with HTML is parsed through it. Similarly, the JSP-enabled engine on the Web server will process the JSP page, which may include technology-specific tags, declarations, and possibly Scriptlets in JAVA, along with HTML or XML tags.

[25] *ASP Tutorial: http://www.w3schools.com/asp/default.asp*

JSP and ASP have some basic concepts in common:

1. They both make use of simple sever-side scripting to provide access to Web server information and functionality.

2. They both have similar styles of delimiting this scripting from a page's content. In fact, Microsoft has recently come up with ASP+, which is much more similar to JSP than ASP.

Yet while ASP primarily supports two scripting languages, JScript and VBScript, JSP actually supports real Java code, not a new scripting language. The difference is that the Java code inside a JSP page is more script-like because it doesn't require Java class and package definitions. JScript, VBScript, and Java (in JSP) are all object oriented to some degree as they are all provided with a set of pre-established objects by the Web server that they use to generate a dynamic Web page.

1.9.2.1. ASP.NET

ASP.NET brings with it a whole new programming model incorporating Web forms, server-side controls, data binding, and Web services. Web forms and server-side controls work by tailoring the markup language they spit out to match the client browser attached. Data binding formalizes exchanging data between controls at runtime (such as edit boxes and combo boxes) and data variables within the Web site program. Finally, Web services formalize the process of getting multiple computers talking to each other automatically using XML and HTTP (SOAP). ASP.NET is ripe for creating Web services in which a machine's software can reveal itself to the rest of the world as a SOAP server.

1.9.2.2. JSP, JSTL and XMLC

Three technologies or approaches to developing Java-based Web applications that utilize Web services are JavaServer Pages (JSP), JSP with the use of tags from the JSP Standard Tag Library (JSTL), and the eXtensible Markup Language Compiler (XMLC).

See the SIRC paper, *“Implementing OGC Web Map Service Client Applications Using JSP, JSTL and XMLC,”* under User Experiences on the OGC Cookbook website.

Figure 11 below shows a process diagram where a XML request is being transformed so that it is understood by the map server, and returned according to the desired outcome. First, the user (Web client) sends the XML request, which is first being handled by 'Main.JSP' controller, which is an HTML page with inline Java code that manipulates dynamic content and specific tags of the XML request. Second, the XML request is processed by the XML request handler that works with the SAX parser to scan the XML request for the necessary information (specific tags needed by the map server to complete the request). The parsed XML request is then sent by the broker to the map server that produces the response. Before the actual XML response gets sent back to the Web client, it is processed (again) now by the XML response handler that performs the XSLT transformation. This is essential for producing the desired content outcome (i.e., map with specific user styles).

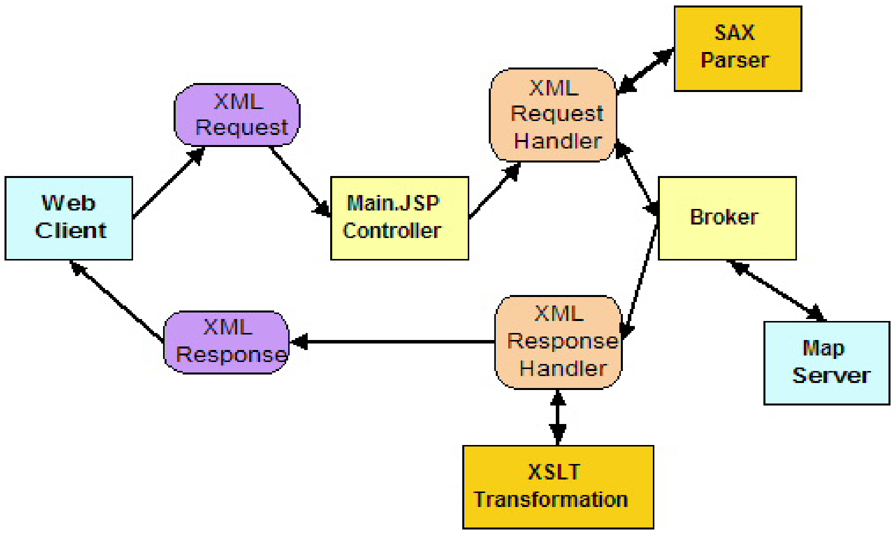


Figure 11: XML Request Transformation Process Diagram

See Chapter 2 for an example of a XSL/XSLT stylesheet for transforming one XML dialect into one that is understood by the specific map server.

1.10. Performance Criteria

[Acknowledgement: This section is from the SIRC paper, "Implementing OGC Web Map Service Client Applications Using JSP, JSTL and XMLC." See the full paepr under User Experiences on the OGC Cookbook website.]

The following set of criteria can be used as a baseline for evaluating the performance of the many approaches and technologies outlined above for implementing WMS client applications:

- **Ease of parsing XML/GML**: Parsing XML/GML documents is an important task to be achieved by the Web clients, since WMS (and more significantly WFS), will utilize XML/GML to transfer information. The ease with which Web clients that process XML/GML can be implemented will be influential in determining which approach is adopted for Web client development. For example, JSTL's XML tags enable XML documents to be parsed within a Web page to simply display XML-based data, but manipulation of those data still needs to be performed in back-end Java programs to prevent page complexity. When using traditional JSP and XMLC, all XML documents parsing tasks are performed in back-end Java programs.

- **Multiple-server interaction**: In a distributed system, a Web client may wish to allow information from different servers to be retrieved and then merged into a single cohesive response for display to the user. The issue to be addressed here is the extent to which the implementation of such processing within a Web client is supported by the implementation approaches.

- **Map handling**: Retrieving and organizing image maps is another key issue that needs to be addressed in a WMS client application. The maps could be retrieved from one map server or multiple map servers, could present a common area or different extents on the earth, or may be overlapped for display to the user. In addition, changing the order of overlapping maps may result in different effects (the order of the layers displayed in the map is determined by the order of the layer names in the list that appeared in the request). Requesting such an image map and placing it into a Web page are very simple tasks using any of the approaches described. Zooming in/out or scrolling through a map is realized by changing the Bounding Box values and submitting a new request, which are done in the back-end (Java) program. The map can then be inserted into the page using the query URL as the image resource.

- **Interface layout**: WMS client applications use the Web browser as the user interface. How easily a Web client can generate dynamic Web pages using the different implementation approaches is an important issue and needs to be considered. For example, compared with JSP's mix of Java and HTML, JSTL makes the Web page cleaner by using standard tags instead of Java code to control dynamical content. However, some functions such as method calling with arguments, which can easily be achieved using Java codes, are not supported in JSTL expression language. XMLC separates all data control from the page; the disadvantages of JSP and JSTL have mostly disappeared in XMLC.

- **Execution speed**: Speed is a factor that always has been used to measure the efficiency of an application or a program. The speed issues to be addressed with WMS clients include the speed of compiling, the speed of request handling, and the speed of pages loading. For example, a slight delay is encountered when a user requests a JSP page for the first time, because JSP and JSTL have to compile the JSP page before processing the request. In contrast, XMLC parses and interprets the page prior to run-time.

- **Ease of revision**: It is normal to modify an established page and application by adding or cutting some components and functions during the development. The issue here is to critique how easily the client can make changes or upgrades based on the previous work with the different implementation approaches.

1.11. Additional Information

1.11.1. OpenGIS Web Map Context Implementation Specification

This OpenGIS Specification[26] describes a standardized approach to enable the capture and maintenance of the context - or state information - of a Web Map Server (WMS) request so that this information can be reused easily in a future user session.

1.11.2. OGC Compliance Testing

Compliance Testing determines that a product implementation of a particular OpenGIS Implementation Specification fulfills all mandatory elements as specified and that these elements are operable. The compliance testing for the WMS Specification is currently in development.

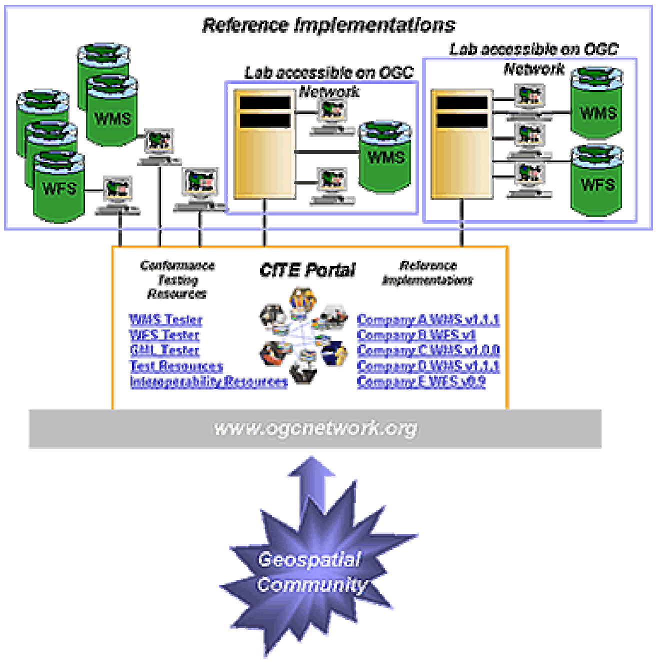
The primary purpose of the testing program[27], of which this conformance testing program is the first phase, is to permit vendors and users to take full advantage of the valuable standards that OGC has created. The Compliance Testing Program provides a process whereby compliance and interoperability can be tested and certification can be supported. When compliance has been confirmed, participants who agree to the terms of the trademark license which accompanies this program document may affix the

[26] *The OpenGIS Web Map Context Documents Implementation Specification is available to the public at https://portal.opengeospatial.org/files/?artifact_id=3841.*
[27] OGC Compliance Testing: URL http://www.opengeospatial.org/resources/?page=testing Testing Program Documentation: URL http://www.opengeospatial.org/resources/?page=testing&view=testdocs

"OpenGIS" or "OPENGIS" mark to their products, thus indicating to their customers that complies with OpenGIS Implementation Specifications has been achieved, and providing incentives for potential customers to preferentially purchase such products.

1.11.3. Conformance and Interoperability Test and Evaluation (CITE) Initiative

CITE[28] is focused specifically on the development of tools and processes for validating the conformance of Standards-based Commercial Off the Shelf (SCOTS) products to OpenGIS specifications. The CITE-1 Initiative is developing a conformance test engine to test capabilities for WMS (and also WFS and GML). CITE is also developing open source "reference implementations" of WMS (and WFS).



[28] *Conformance and Interoperability Test and Evaluation (CITE) Initiative:* http://www.opengeospatial.org/initiatives/?iid=65