Open GIS Consortium Inc.

OpenGIS Web Map Server Cookbook

April 28, 2003

Editor: Kris Kolodziej

OGC Document Number: 03-050

Version: 1.0.0

Stage: Draft

Language: English

Copyright Notice

Revision History

Version	Date	Author/Editor	Comments
1.0.0	*April 14, 2003*	*Kris Kolodziej*	

Document Contact Information

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

Name	Organization	Contact Information
Greg Buehler	*OGC*	*email@opengis.org +1 (812) 334-0601*

Future Work

Table of Contents

Preface ... **1**
Submitting Organizations (The Contributors) ... 1
Organization of the Cookbook ... 2
Acknowledgments ... 2
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 Conformance Testing ... 38
1.11.3. Conformance and Interoperability Test and Evaluation (CITE) Initiative ... 38
2. WMS Examples: Software Architectures, User Experiences, Requests, and More ... **39**
2.1. Introduction to this Section ... 39
2.2. Software Architectures ... 41
2.2.1. The Web Client ... 41
2.2.2. The Map Server ... 43
2.2.3. The Database ... 43
2.2.4. Software Architecture Example 1: University of Minnesota ... 43
2.2.5. Software Architecture: Example 2: WirelessInfo ... 45
2.2.6. Software Architecture Example 3: GeoMedia ... 46

2.2.7. Software Architecture Example 4: ArcIMS 47
2.3. User Experiences Examples 50
2.3.1. User Experience Example 1: Wupperverband 50
2.3.2. User Exerience Example 2: Wireless Info Project 57
2.3.3. User Experience Example 3: Harvard University WMS (Distributed) Servers 61
2.3.4. Experience with ESRI ArcIMS WMS 65
2.3.5. Experience with Intergraph GeoMedia WebMap Professional 65
2.4. WMS Request Examples 66
2.4.1. Request Example: Get_Capabilities 66
2.4.2. Request Example: Get_Map 71
2.5. XSL/XSLT Stylesheet Examples 72
2.5.1. XSL/XSLT Stylesheet Example 72
2.5.2. DTD & XML Example 1: Understanding and Using Styles 77
2.5.3. DTD & XML Example 2: Basic Symbols 78
2.5.4. DTD & XML Example 3: Scaleable Symbols 80
2.5.5. DTD & XML Example 4: Complex Types 81
2.5.6. DTD & XML Example 5: Create and Use Legends 84
***3. WMS Recipes ....** Error! Bookmark not defined.*
*3.1. Recipe 1: The ArcIMS OGC WMS Connector: .... **Error! Bookmark not defined.***
3.1.1. Step 1: Preliminary Requirements 88
3.1.2. Step 2: Installation of the ESRI OGC WMS Connector 89
3.1.3. Step 3: Configure the WMS Connector 90
3.1.4. Step 4: Setup a MapService and Client for your WMS 92
3.1.5. Step 5: Test your WMS Service 92
3.2. Recipe 2: deegree Web Map Server 93
3.2.1. Step 1: Preliminary Requirements 93
3.2.2. Step 2: Installation of deegree WMS 93
3.3. Recipe 3: University of Minnesota (UMN) MapServer 99
3.3.1. Recipe 3A: UMN Mapserver HOWTO for Setting Up a WMS Server with Mapserver 99
3.4. Recipe 3B: UMN MapServer HOW TO for Setting Up a WMS Client with Mapserver 105
3.4.1. Step 1: Compilation / Installation 105
3.4.2. Step 2: Check your MapServer executable 105
3.4.3. Limitations/To Do 108
3.4.4. Additional Information: Using ArcExplorer 4 as a Client for WMS-compliant UMN Mapserver 108
3.5. Recipe 4: Intergraph GeoMedia WebMap WMS Adapter Kit 108
3.5.1. Step 1: Preliminary Requirements 109
3.5.2. Step 2: Creating your WMS using the GeoMedia WebMap WMS Adapter Kit 111
3.6. Recipe 5: How to build a WMS from Free Parts 116
3.7. Recipe 6: GeoServNet (GSN) 3D Client 120
3.7.1. Steps for Installing the OGC 3D Viewer 121
3.7.2. 2D Viewer Operation 121
Appendix 1. deegree WMS Configuration Files (Recipe 2) 136
Appendix 1.A. Configuration.dtd 136
Appendix 1.B. Configuration.xml 137
Appendix 1.C. Gcdescriptor.dtd 139
Appendix 1.D. Gcdescriptor.xml 140
Appendix 1.E. jld.dtd 143
Appendix 1.F. deegree.xml layer descriptor 148
Appendix 1.G. XSLT-script for formatting the results of a FeatureInfo Request 157
Appendix 2. International Interfaces (Recipe 5): Basic-wms2.py file 158

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.

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 Greg Buehler at email@opengis.org.

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 [Editor]	*M.I.T.*	*Chapter 1 Chapter 2: Preface, System Architecture Example, XSL/XSLT Example*	*+1-646-244-2731*	*Kwk@mit.edu*
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*	*+1-613-565-5056*	*mckenna@dmsolutions.ca*

[2] WMS 1.1.1 Adopted OpenGIS Specification http://www.opengis.org/techno/specs/01-068r3.pdf

		Recipe (Client & Server)		
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 Levison	*Harvard University*	*Chapter 2: WMS User Experience*		lovison@seismology.harvard.edu
Allan Doyle	*International Interfaces*	*Chapter 3: Making a WMS of out Free Parts*	*+1-781-433-2695*	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*		

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

Date	*Release*	*Author*	*Section modified*	*Description*
March, 2003	*1.0.0*	*Kris Kolodziej*		*Original document*

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.

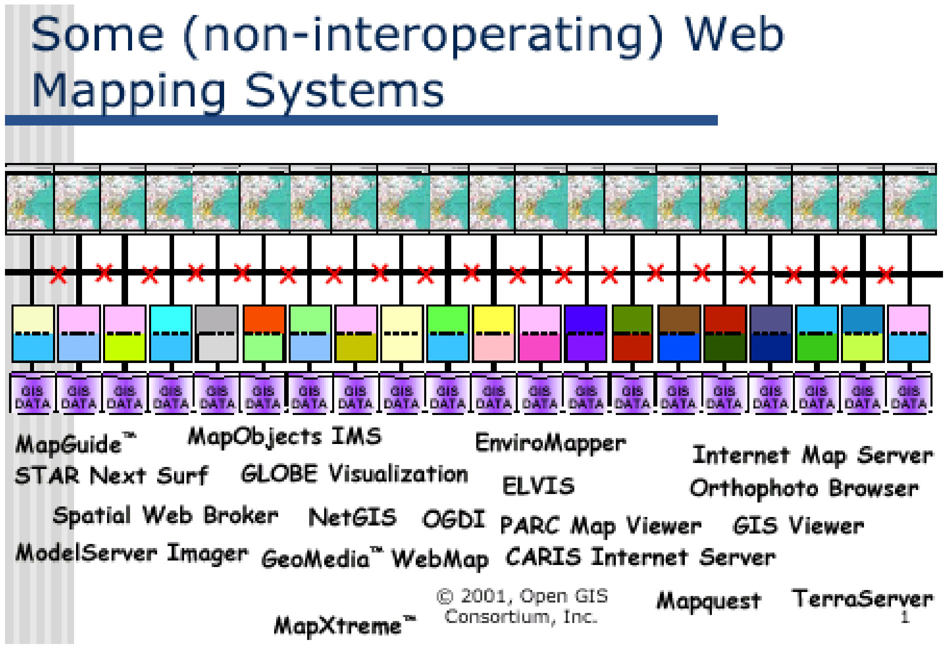


Figure 1: Current Status of Non-Interoperable Web Mapping Systems

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.

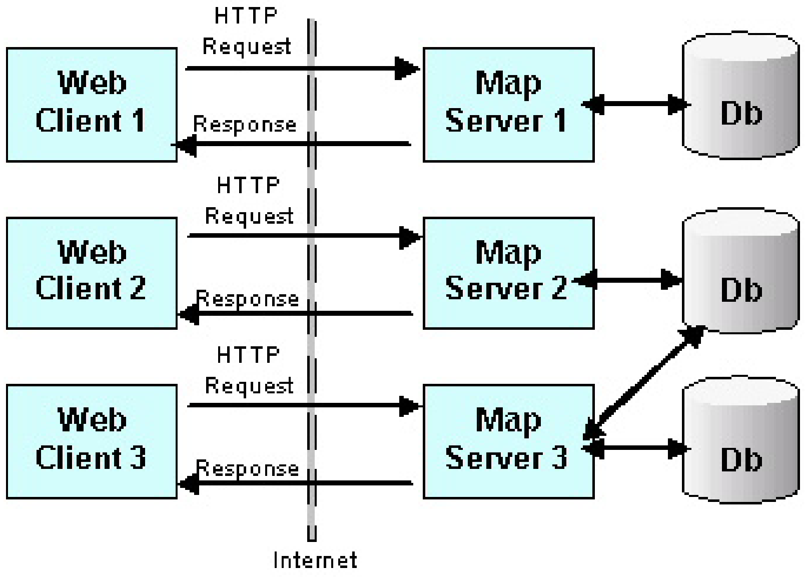


Figure 2: Client/Server Lack of Interoperability

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 GIS 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.

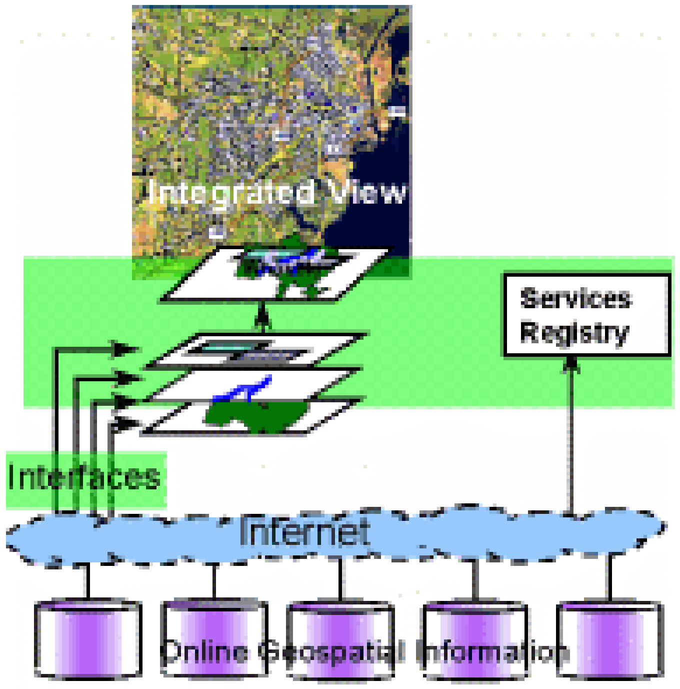


Figure 2: Interoperable Map Overlay

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://opengis.org/techno/implementation.htm.*

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.

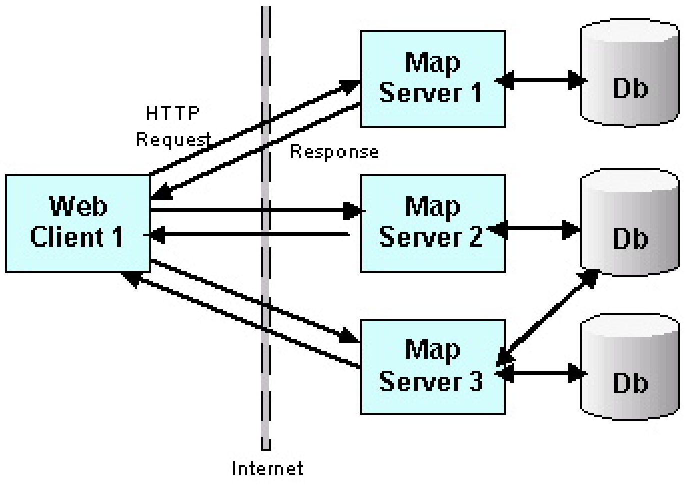


Figure 4: Client/Server Interoperability

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.

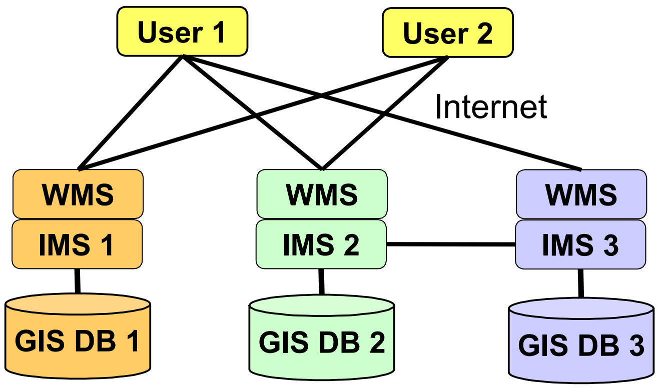


Figure 5: WMS Communication (courtesy of CSC PloenzkeAG).

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)

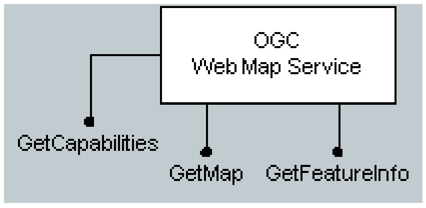


Figure 6: The OpenGIS Web Map Service Interface

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 2.5.2. , 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: http://www.opengis.org/techno/RFC14.pdf*

[6] *WMS Implementation Specification Part 2: XML for Requests using HTTP POST. See URL http://www.opengis.org/techno/discussions/02-017rl.pdf. This document is advanced as a draft candidate 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://ip.opengis.org/ows/*
[8] *WCS – Web Coverage Service—Interoperability Program Report: http://www.opengis.org/techno/discussions/02-024.pdf*

[9] *WFS 0.0.14 specification: http://www.opengis.org/techno/RFC13.pdf*

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.

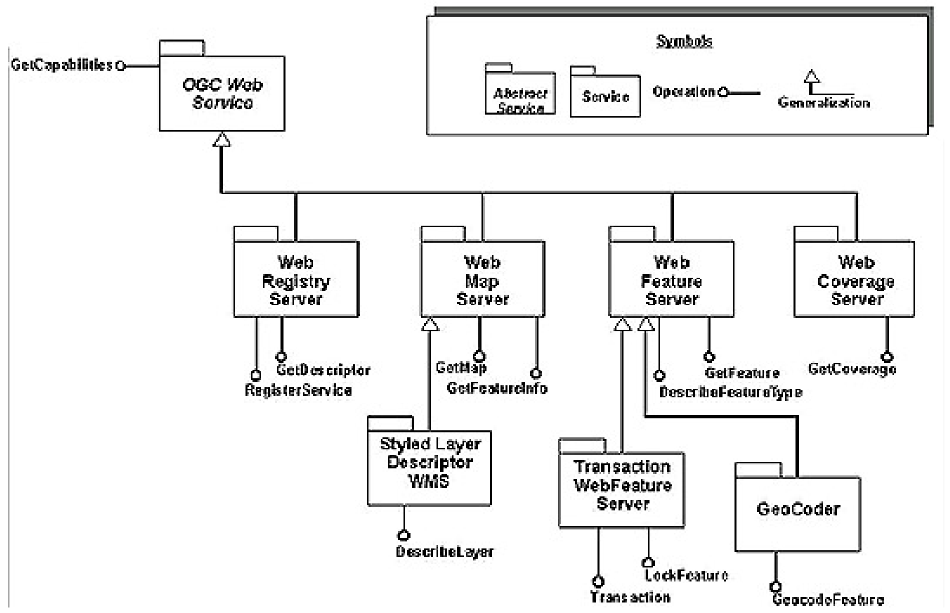


Figure 7: OGC Web Service Architecture Diagram.

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.

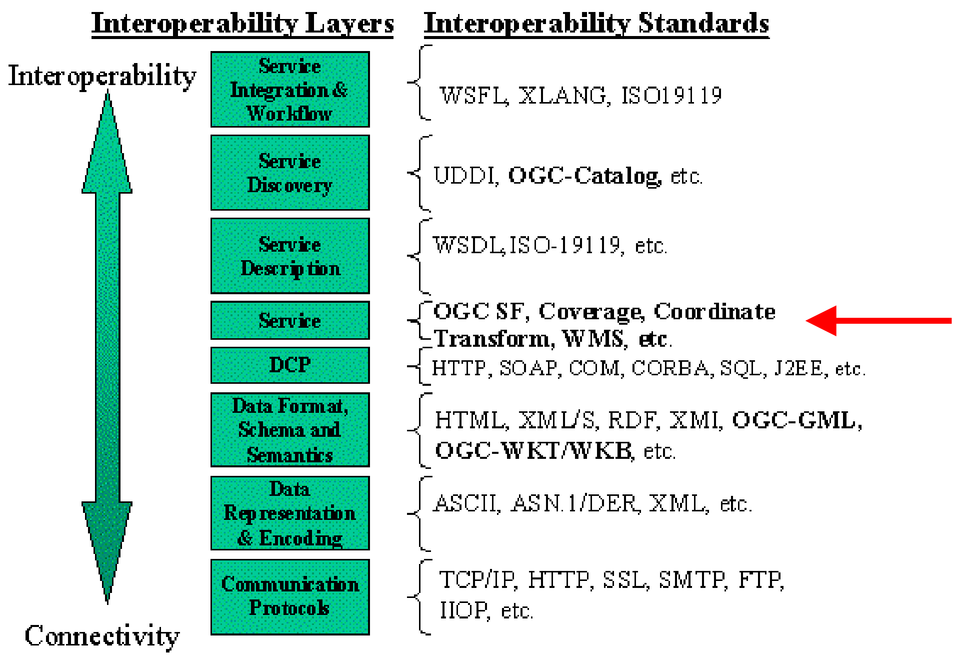

Interoperability Layers
Interoperability Standards
Interoperability
Connectivity
Service Integration & Workflow
WSFL, XLANG, ISO19119
Service Discovery
UDDI, OGC-Catalog, etc.
Service Description
WSDL, ISO-19119, etc.
Service
OGC SF, Coverage, Coordinate Transform, WMS, etc.
DCP
HTTP, SOAP, COM, CORBA, 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.opengis.org/public/abstract/99-100rl.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.intl-interfaces.com/servicemodel/gsm/gsm-2001-08-15.html*

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.opengis.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.opengis.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.*

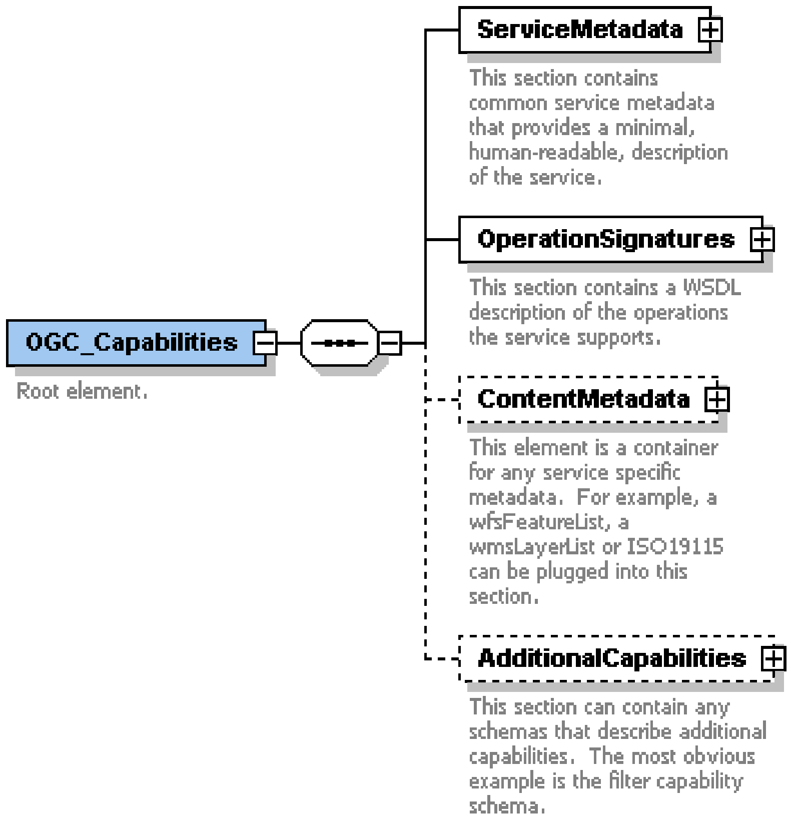

ServiceMetadata
This section contains common service metadata that provides a minimal, human-readable, description of the service.
OperationSignatures
This section contains a WSDL description of the operations the service supports.
OGC_Capabilities
Root element.
ContentMetadata
This element is a container 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:

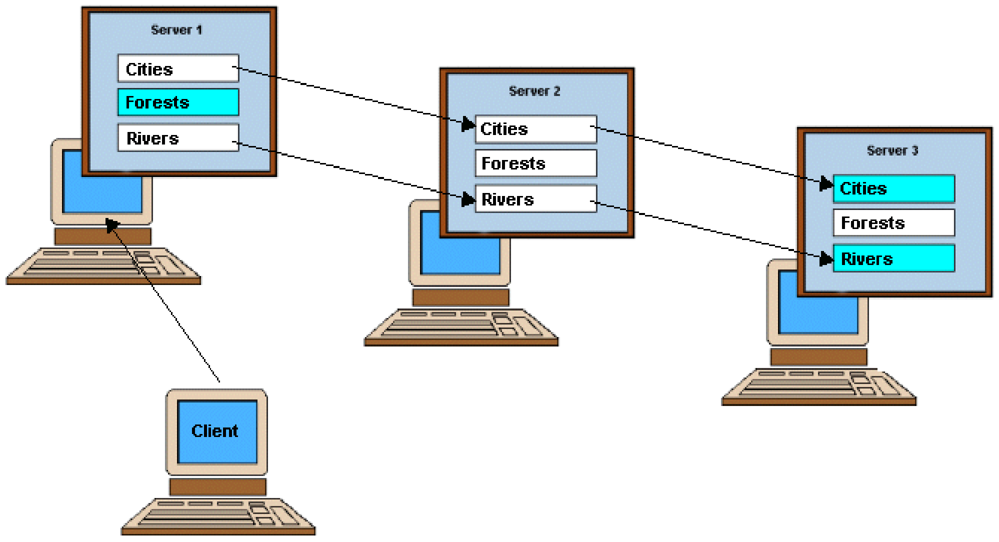


Figure 10: WMS Cascading Servers (courtesy of WirelessInfo)

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

'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 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*.

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

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
- Marked sections

[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/

- 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.

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

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

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:

```
"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

[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.*

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).

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.

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

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).

JSP is based on Java servlet technology and allows the insertion of Java code directly into static HTML. A JSP page is basically an HTML page with inline Java code that manipulates dynamic content and specific tags in addition to the regular HTML tags. The embedded Java code is expressed using script elements. A JSP page is processed by a JSP container. When the Web browser makes a request for a JSP page, the JSP container in the Web server first compiles the JSP page into a servlet, which is a Java program. Then the JSP container compiles the servlet with more business logic Java code (i.e. JavaBeans), and finally the JSP container executes the compiled servlet class to produce a generated Web page to the Web browser. Therefore, a JSP page is actually another way to write a servlet without having to be a Java programming expert.

JSTL is based on JSP, but uses standard functional HTML-like tags instead of Java code to manipulate dynamic content in a JSP page. JSTL includes tags for many common tasks such as looping over data, performing conditional operations, importing and processing data from other Web pages, simple XML manipulating, database accessing, and text formatting. JSTL also supports an Expression Language (EL) that is inspired by both ECMAScript (JavaScript) and the XPath expression languages. This makes JSP pages have easier access to the data required.

XMLC is a very different method from JSP and JSTL. XMLC provides an object-oriented mechanism for generating dynamic Web pages from static template HTML pages, which entirely separates the Web page appearance design from the development of dynamic content. Web page authors can design attractive static HTML pages, adding "id" attributes and mock-up content into the elements that will be manipulated later. XMLC then compiles the template HTML pages and converts them to Java classes, where the HTML pages are represented using the DOM. The generated classes can be accessed, and the attributes, content, and nested tags of the elements with "id" attributes can be replaced or removed by Java programs using standard DOM manipulation APIs to create dynamic web pages.

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).

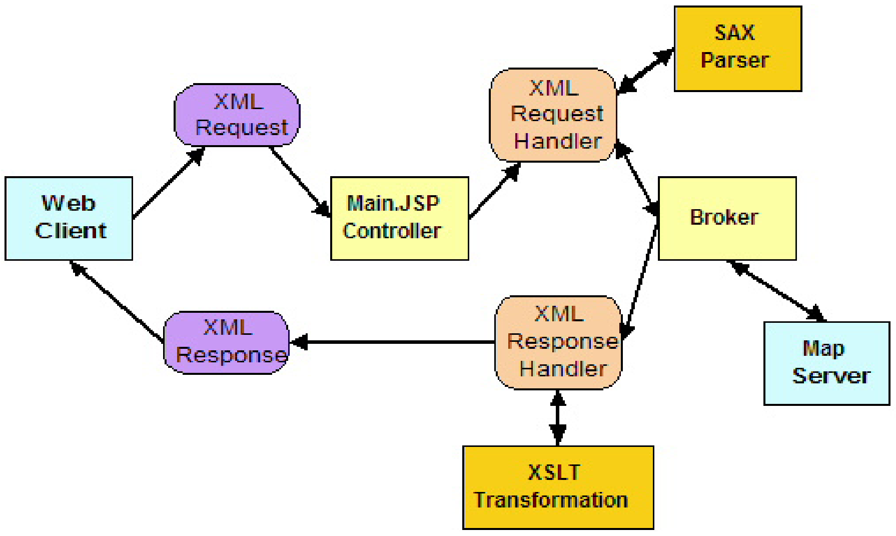


Figure 11: XML Request Transformation Process Diagram

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

In order to help evaluate the performance of the many approaches and technologies outlined above for implementing WMS client applications, a set of criteria is presented below:

- **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.

[26] The OpenGIS Web Map Context Documents Implementation Specification was approved in April, 2003 and will be available to the public at http://www.opengis.org/techno/implementation.htm by June, 2003.

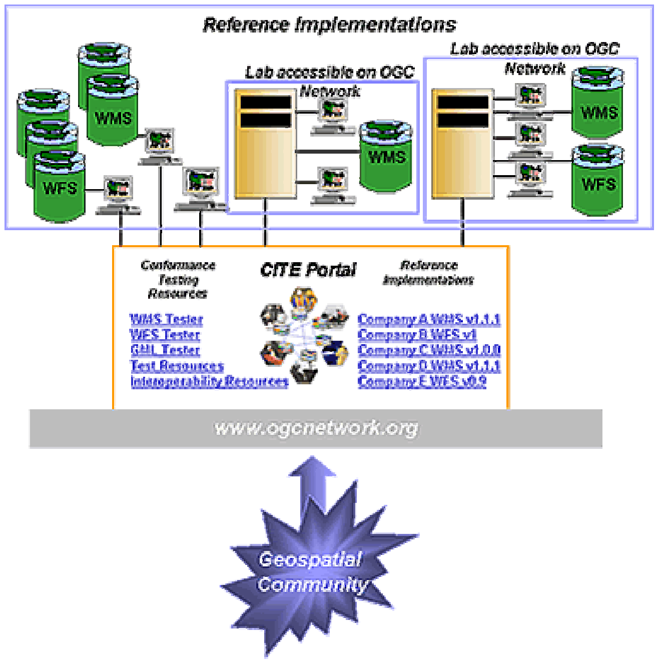
1.11.2. OGC Conformance Testing

Conformance 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 conformance 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 Conformance Testing Program provides a process whereby conformance and interoperability can be tested and certification can be supported. When conformance has been confirmed, participants who agree to the terms of the trademark license which accompanies this program document may affix the "OpenGIS" or "OPENGIS" mark to their products, thus indicating to their customers that conformance 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).



[27] OGC Conformance Testing: URL http://www.opengis.org/testing/about.html Testing Program Documentation: URL http://www.opengis.org/testing/documents.html#ctpdoc

[28] *Conformance and Interoperability Test and Evaluation (CITE) Initiative:* http://ip.opengis.org/cite/

2. WMS Examples: Software Architectures, User Experiences, Requests, and More

2.1. Introduction to this Section

The WMS Specification specifies interfaces and schemas that enable developers to enable users to dynamically integrate geographic information from several different WMS services, no matter what underlying technology is used. It enables the creation of a network of WMS map servers from which WMS clients can build customized maps.

The **Software Architectures** section shows implementation examples of WMS Web mapping systems and how relevant system components are deployed and interact.

The **User Experience** section builds further understanding of how the WMS interface is used by different users around the world. In addition, user experiences provide their motivation and an explanation of the practical use of the WMS interface.

The **WMS Request Example** sections feature some practical WMS request and response examples.

The **XSL/XSLT Stylesheet Example** section shows an example of a XSL script that transforms a request (its XML dialect) into one understood by the map server.

The **DTD/XML Example** section includes examples on how to define rules that will control such things as map rendering, styles, symbols, and legends.

The table below provides a guide to these sections.

SOFTWARE ARCHITECTURES			
Example	**Description**	**Contributor**	**Pg. No.**
1	Open Source (UMN MapServer, PostGIS)	Editor	43
2	WirelessInfo system architecture	The WirelessInfo Project	45
3	GeoMedia software architecture	Intergraph	46
4	ArcIMS Software architecture	ESRI	47
USER EXPERIENCE			

Example	Description	Contributor	Pg. No.
1a	Publishing your data as a WMS Service using ESRI ArcIMS 4	CSC Ploenzke	51
1b	Integrating published data in your ArcIMS WMS Client	CSC Ploenzke	51
2	WMS for wireless devices	The WirelessInfo Project	57
3	Distributed WMS servers	Harvard University	61
WMS REQUEST EXAMPLES			
Example	**Description**	**Contributor**	**Pg. No.**
1	`GetCapabilities` Request	Ocean ESIP, JPL	66
2	`GetMap` Request	Ocean ESIP, JPL	71
XSL/XSLT STYLESHEET EXAMPLES			
Example	**Description**	**Contributor**	**Pg. No.**
1	Transforming a WMS `GetMap` Request	Editor	72
DTD & XML EXAMPLES			
Example	**Description**	**Contributor**	**Pg. No.**
1	Understanding & using styles	lat/lon	77
2	Basic symbols	lat/lon	78
3	Scaleable symbols	lat/lon	80
4	Complex types	lat/lon	81
5	Create and use legends	lat/lon	84

Table 4: Architectures, Experience and Examples in this Cookbook

2.2. Software Architectures

A single-tier application runs only at the client and does not need persistent storage. A two-tier application consists of a client that provides a user interface and a server that provides persistent storage. Two-tier applications tend to be personal productivity applications, such as word processors and spreadsheet programs, which locally access user files.

A multi-tier architecture makes use of three or more tiers, each of which may make use of several applications or services. Map server applications usually consist of three tiers, described as the presentation tier, the business logic tier, and the data storage tier. Note that these tiers describe logical groupings of the functionality of the various application components (and do not necessarily correspond to their physical location). Figure 12, below, depicts the building blocks of a 3-tier system architecture.

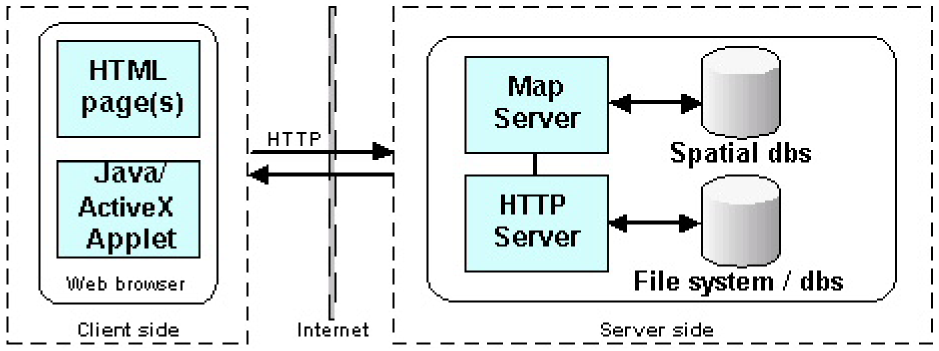


Figure 12: A Mulit-tier System Architecture

Many commercial and open source solutions already available allow building up a 3-tier system architecture. Several of them are described in this chapter.

2.2.1. The Web Client

The Web client is a program for processing the user requests and visualizing spatial data. It is a series of HTML pages dynamically generated and running inside the Web browser that can communicate directly with a map server via the HTTP protocol. More specifically, a WMS compliant Web client is an application/program that communicates with the WMS servers using the three functions: `GetCapabilities`, `GetMap`, and `GetFeatureInfo`.

In its most basic form, a "thin" Web client is an HTML page running inside a Web browser. The Web client uses the Web browser to handle interactions with the user. The Web client, also by way of the Web browser, sends WMS interface requests (i.e., `GetMap`, `GetCapabilities` and `GetFeatureInfo`) to map servers as Web-standard HTTP URLs. The Web client receives (via the Web browser) the response (an image map in GIF, JPEG, PNG, XML or some other MIMI-encoded format) from the map server. Some further interaction may be achieved by computing the HTML pages on demand by using CGI scripts or ASP or Java servlets on the server site.

A thick Web client is a more intelligent program that has either a Java application or a Web browser plug-in. The advantages of "thin" versus "thick" clients is their low cost, and the ability of almost every web browser to visualize such web pages. However, the interaction with the user is very restricted. For implementing more demanding applications, this technique is not sufficient.

In a typical WMS client/server interaction, as portrayed in the configuration in Figure 13, the Web client requests `GetCapabilities` from the a WMS compliant map server in order to determine what the map server can do and what maps the map server can provide. The Web client then requests `GetMap` with the map server's capabilities information in order to get a map image. Finally, the Web client can request `GetFeatureInfo` by specifying a point on the map to receive more geographic feature information about a particular map location. All these Web client requests to the map server marked as #1 in the figure are encoded as an HTTP URL.

In response to a `GetCapabilities` request, the WMS server produces an XML document containing the Web map server's service metadata, describing all the operations it supports, and providing information about the available maps. The Web client application has to parse the XML capabilities document to retrieve the information necessary to request a map. See Section 1.7.6. on the DOM and SAX parsers. With the capabilities information, the Web client can request a map image from the map server using the `GetMap` operation. See Section 1.3.2. for the list of essential parameters for `GetMap`).

Next, the map server processes the request (#2), perhaps accessing data stored in a database or in a set of files. Lastly, the map server returns to the Web client a map image (#3) encoded in the form of a MIME-type picture such as GIF or JPEG.

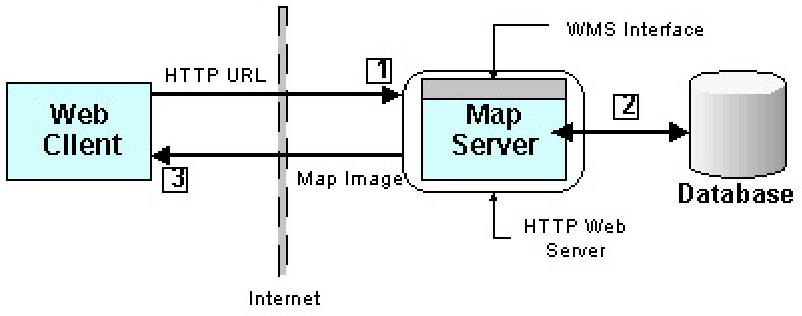


Figure 13: Typical WMS Client/Server Interaction

After getting a map, the Web client can request feature data of a specific point on the map using the `GetFeatureInfo` function. The response from the WMS server will be one of the three output formats: a GML file, a plain text file, or an HTML file. The Web client application parses the GML and displays the feature information in the Web browser. The resulting map should be presented as an HTTP image map so that the user can click on a specific point of the map to get more feature information using the `GetFeatureInfo` function.

In the second task (#2), the client continues to interact with a single WMS, but multiple `GetMap` requests will be sent if more than one Layer is selected by the user - one request per Layer. Thus, if the user selects three Layers (for example, buildings, roads, and parcels), three separate maps will be requested. Because the maps are requested for the same extent, they can be overlapped in a user specified order to make a new map. Each single-Layer map must be transparent so that one map will not be hidden by those on top of it when they overlap.

In a scenario where the client connects with multiple Web map servers, and each map server has different capabilities, the Web client application must send `GetCapabilities` requests to each of the WMS servers and parse each capabilities XML document individually. When the user selects Layers belonging to different map servers, each map server will return a map, and these maps may represent different extents of the earth. Only maps of the same extent can be transparently overlapped by the Web client application.

When the user clicks a specific point on the combined map, the feature data about that point could include data from multiple maps that are from different map servers. The Web client application must request `GetFeatureInfo` to each of those map servers, parse each GML document, and display all feature data in the web browser.

2.2.2. The Map Server

The map server handles the requests of the Web clients. A map server is a software component that delivers symbolized graphics across a common interface to a Web client. Map servers accept requests from Web clients as Web-standard HTTP URL strings and return results encoded as GIF, JPEG, PNG, XML, etc. In general, a map server is closely linked with an Internet server that handles the HTTP requests of the Web browser. The map server is independent of the application.

A WMS server is a map server that provides the three WMS operations: GetCapability, `GetMap`, and `GetFeatureInfo`. WMS compliant map servers interact with Web clients using the HTTP (CGI-BIN) profile of the WMS interface. A WMS server accepts requests from WMS client and viewer client in the form of HTTP URL strings, and returns results encoded as XML, GIF, GML, and so on.

A map server typically provides geographic information for a specified spatial extent and this information is typically divided into thematic layers. In the case of WMS compatible map server, the extent covered by the WMS map server and the thematic layers available are described by the WMS server's metadata file (the capabilities XML document).

2.2.3. The Database

The database is a system for storing data and processing requests of the map server. The database management system is independent of the application but the database (i.e. the data model and the data) must be designed according to the requirements of the applications. The spatial database stores geographic data features that can be accessed by the WMS interface and utilized by the WMS server to generate maps and/or GML documents.

A request of a Web client usually triggers one or more queries to be performed by a database system or by a comparable system. The query results of the database system are processed first by the map server and then transmitted to the Web client application.

2.2.4. Software Architecture Example 1: University of Minnesota

Note: See Chapter 3, Recipe 3, to get step-by-step instructions on how to set up a WMS server and a WMS client with UMN Mapserver.

Note: Look into Chapter 3, Recipe 5, to see how to build a WMS from free parts.

As shown in Figure 14 below, the user interacts locally with the Web client and submits HTTP GET/POST requests to the WMS client. The Web server in the WMS client accepts user requests and parses (see Chapter 1, section 1.6 on parsing tools and technologies) them before forwarding them to the application server. The application server then processes these requests in an application, and returns dynamically generated HTML pages to the Web browser.

In real practice, `GetCapabilities` and `GetFeatureInfo` are requested from, for example, a Java program, and a `GetMap` request is embedded in the HTML pages in the form of `<img src=GetMap request url\>` to fetch an image map from the map server and display it directly in the Web browser.

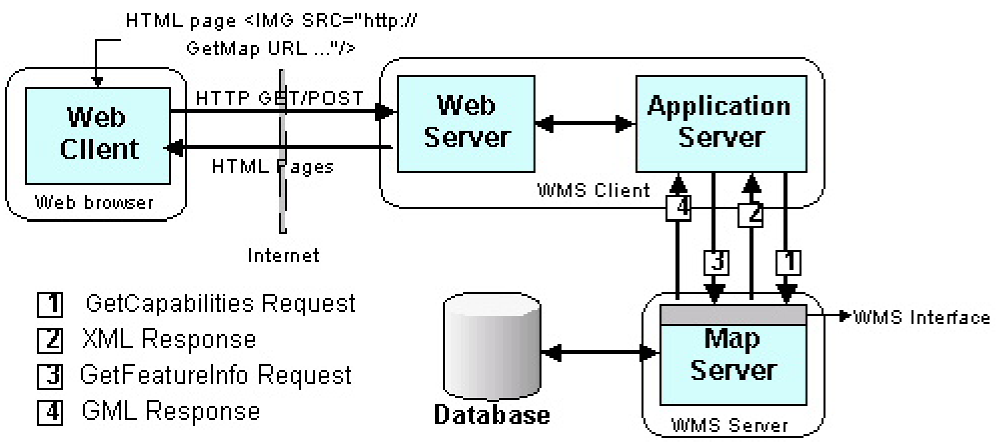


Figure 14: Local WMS System Architecture

Based on the architecture described previously, the WMS Web mapping system can be built using open source software. Figure 15 below shows such a system - the WMS client uses Jakarta Tomcat[29] (as the Web server) and JSP/servlet container, UMN MapServer (as the WMS compatible map server), PostgreSQL/PostGIS (as the GIS database). Any Web browser supporting HTML 4.0, such as Internet Explorer or Netscape Navigator, can be used as a Web client.

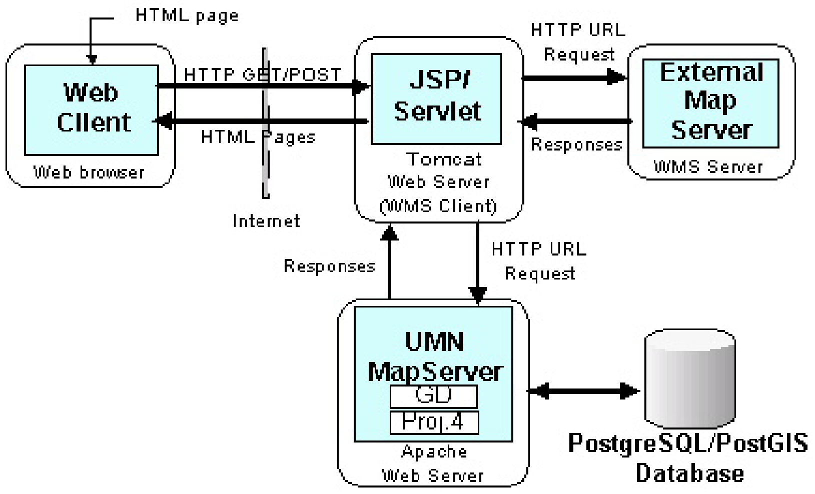


Figure 15: Practical Open Source Web Mapping System

In this example, the WMS server was set up using the UMN MapServer 3.6[30], which supports WMS Specification version 1.1.0. The UMN MapServer consists of only one executable file named "mapserv", which is a CGI program running in the Apache HTTP Web server[31] that knows how to handle WMS

[29] *Jackarta Tomcat: http://jackarta.apache.org/tomcat*
[30] *University of Minnesota (UMN) MapServer: http://mapserver.gis.umn*
[31] *Apache HTTP Web Server: http://www.apache.org*

requests. In practice, when multiple map servers are needed, several Apache Web servers can simultaneously run UMN MapServers on different ports. UMN MapServer must be compiled together with the PROJ.4[32], which is a cartographic projections library, for OGC WMS compliance (See Harvard University's User Experience contribution in this chapter, to learn more).

UMN MapServer also uses GD library[33] to render GIFs or PNGs. Each UMN MapServer needs a mapfile (a control file with the suffix .map), which is a configuration file defining display and query parameters and the data source to be used. A mapfile must include information about where to get the spatial data, how to draw the map, and what layer metadata and spatial information must be included in the WMS capabilities document. Look into Chapter 3 under the UMN MapServer recipe contribution for more.

UMN MapServer uses PostGIS[34] as a vector database to store geo-feature data. PostGIS is an extension to the PostgreSQL[35] object-relational database system that allows GIS objects to be stored in the database. PostGIS supports the "simple features" defined by the OGC. These simple features are `Point`, `LineString`, `Polygon`, `MultiPoint`, `MultiLineString`, `MultiPolygon`, and `GeomCollection`. Vector format data is a coordinate-based data structure that represents the location and shape of feature and boundaries precisely, and can be used directly by UMN MapServer to draw maps or generate GML files.

The vector datasets to be stored in the PostGIS can be converted from ESRI shapefiles. PostGIS provides a tool called "shp2pgsql" to create an SQL file from a shapefile. The SQL file holds SQL commands such as "create table" and "insert data", which can be executed directly by PostGIS. One shapefile corresponds to one PostGIS table, containing the points, lines, or polygons to display as map layers.

2.2.5. Software Architecture: Example 2 – WirelessInfo

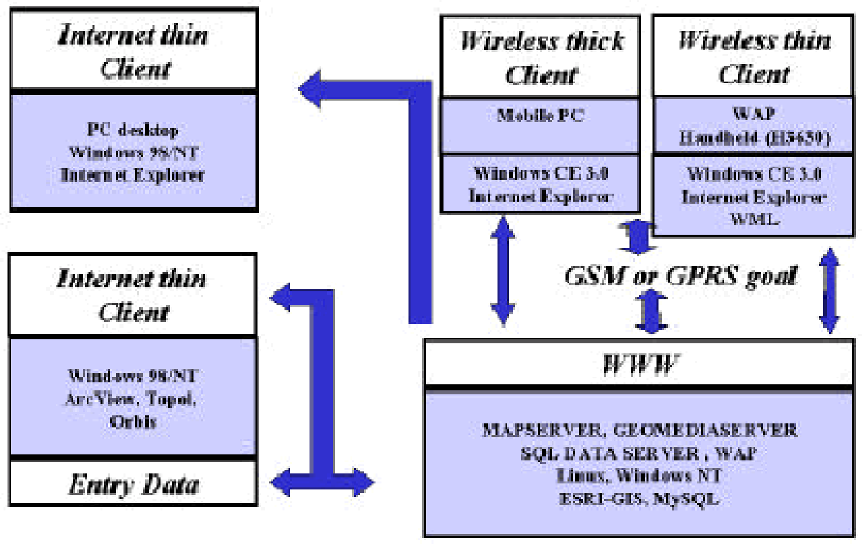


Figure 16: WirelessInfo System Architecture (WirelessInfo)

[32] *PROJ.4 Compilation Projection Library: http://www.remotesensing.org/proj*
[33] *GD Library for rendering GIF's and PNG's: http://www.boutelll.com/gd*
[34] *PostGIS http://postgis.refractions.net*
[35] *PostgreSQL http://www.postgresql.org*

2.2.6. Software Architecture Example 3: GeoMedia

2.2.6.1. Intergraph GeoMedia Web Map Architecture and the WMS Adapter

By implementing a Geographic Data Objects (GDO) Data Server for WMS (and WFS), any data store supporting the OGC WMS (or WFS) interfaces can be accessed and utilized by GeoMedia products in exactly the same way as any other data stores.

In GeoMedia or GeoMedia Professional, the user specifies a warehouse connection using the Connection Wizard. However, instead of specifying a directory path or database name, a URL is specified. Once the connection is made, the GeoMedia or GeoMedia Professional user can treat this warehouse like any other GeoMedia warehouse. Data from a WMS can be used as a background while data from a WFS is sufficiently rich for analysis. This includes the ability to use all analysis, display, and reporting functions.

Figure 17, below, depicts how the OGC interfaces are incorporated into the existing GeoMedia architecture.

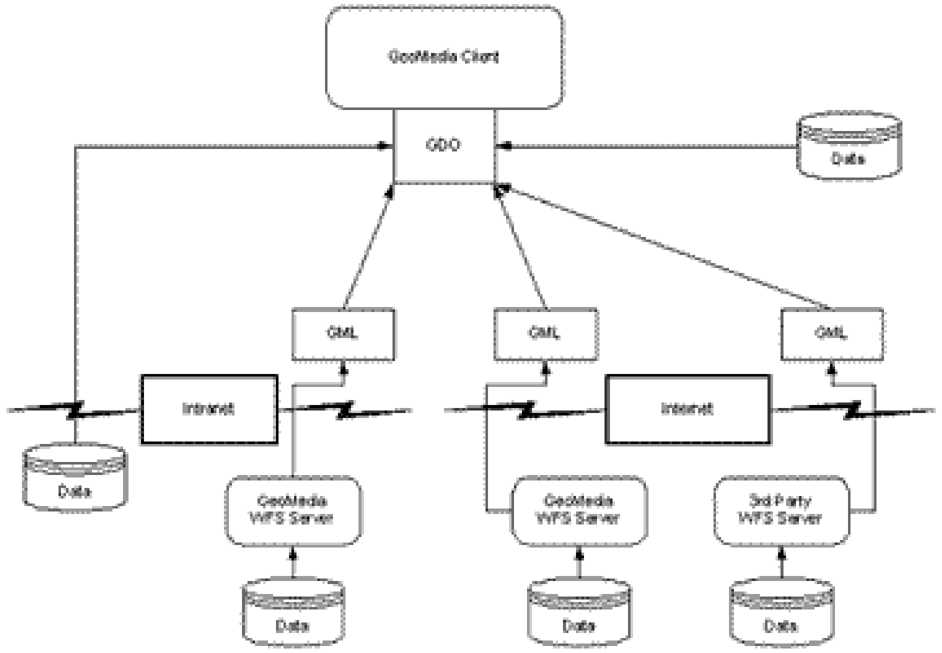


Figure 17: Intergraph GeoMedia System Architecture (courtesy of Intergraph)

GeoMedia WebMap and GeoMedia WebMap Professional can connect to WMS and WFS sources. In this manner, they can integrate data coming from traditional sources with data coming from OGC compliant web servers.

2.2.6.2. Why Implement OGC Interfaces in GeoMedia?

The standard spatial data access API used in GeoMedia is referred to as Geographic Data Objects (GDO). GDO is an open, public interface that providers can use to expose data to GeoMedia clients. Data connectivity is tied to the existence of GDO data servers (data providers). This has proven to be extremely useful in resolving the cases where there was no data sharing due to disparate GIS systems in use in different departments of an enterprise.

The user has a problem to solve and wants to solve that problem as quickly and effectively as possible. Intergraph's GeoMedia product suite capitalizes on this philosophy. GeoMedia supports live data access from many different vector and raster formats, integrating and fusing spatial data on-the-fly, and supporting analysis across and between disparate data sources.

With the incorporation of the OGC interfaces into GeoMedia, Intergraph has extended this philosophy to incorporate emerging industry standards for spatial data interchange and access across the WWW.

Software Architecture Example 4: ArcIMS

2.2.6.3. ESRI ArcIMS Architecture & the WMS Connector

ArcIMS was designed with a multi-tier architecture. A multi-tier architecture makes use of three or more tiers, each of which may make use of several applications or services. Figure 18 below shows the ArcIMS architecture[36], which consists of three tiers: (1) presentation tier; (2) the business logic tier; and (3) the data storage tier. These tiers describe logical groupings of the functionality of the various application components and do not necessarily correspond to their physical location.

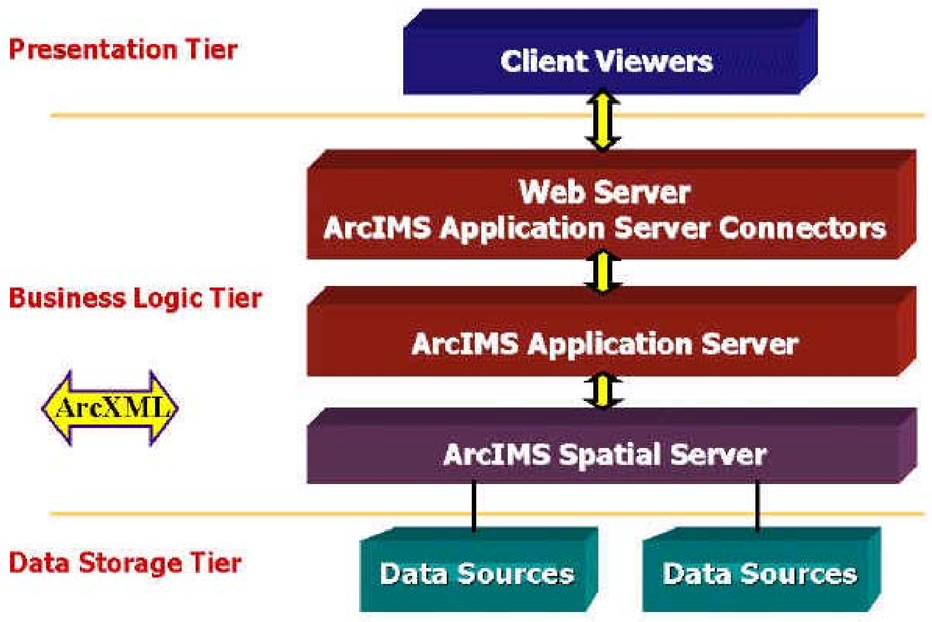


Figure 18: ArcIMS Multi-tier Software Architecture (courtesy of ESRI)

The presentation tier includes the ArcIMS client viewers for accessing, viewing, and analyzing geographic data. The components in the business logic contain the components needed to run MapServices and process requests and responses. The components include the Application Server Connectors, the ArcIMS Application Server, and the ArcIMS Spatial Server. The framework also requires the Web server, JavaVM, and the servlet engine. The data tier includes all data sources available for use with ArcIMS.

[36] *The ArcIMS Architecture: An ESRI White Paper* http://supportt.esri.com/

2.2.6.4. ArcIMS System Components

Figure 19 below shows that the business logic tier of ArcIMS consists of the ArcIMS Application Server Connectors, Application Server, Spatial Server, and Manager as well as a Web server. The server components are used to process requests, create and run MapServices, and manage the site.

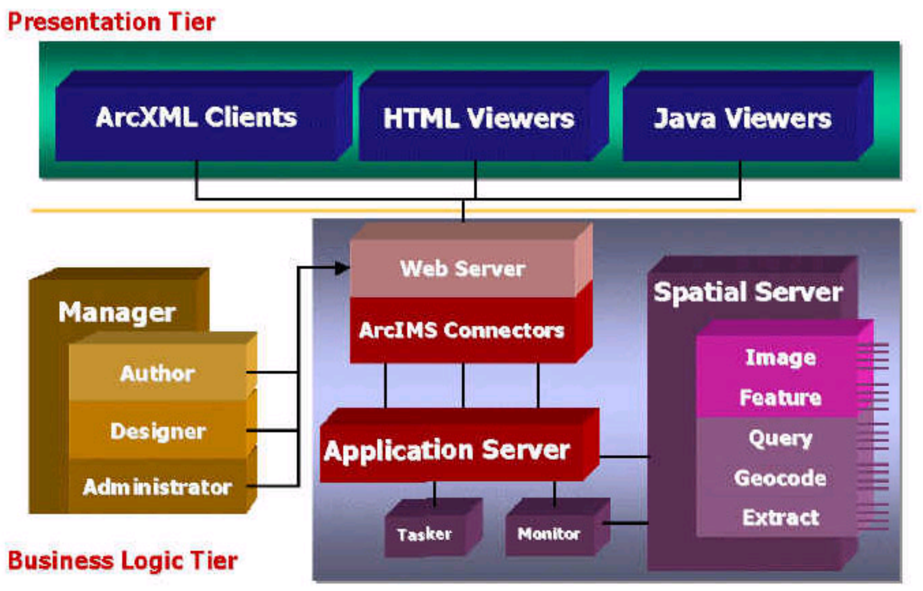


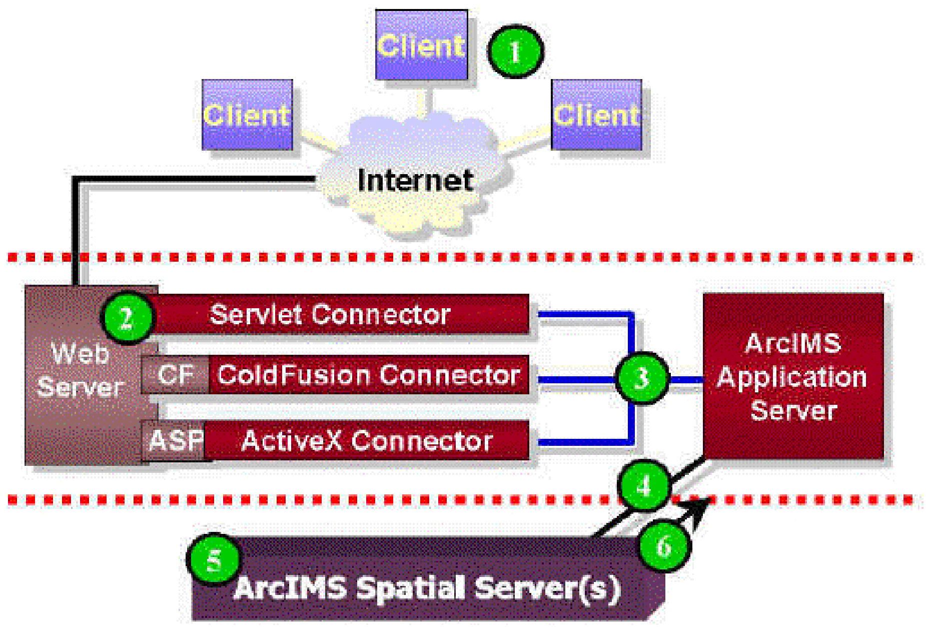
Figure 19: ArcIMS System Components (courtesy of ESRI)

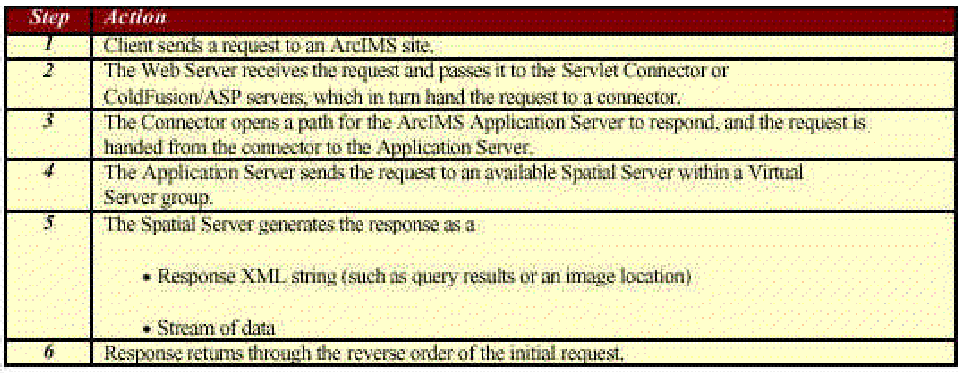
The ArcIMS Application Server runs as a background process and handles the load distribution of incoming requests. It also catalogs which MapServices are running on which ArcIMS.

The ArcIMS Spatial Server provides the functional capabilities for accessing and bundling maps and data into the appropriate format before sending the data back to a Web browser. The Spatial Server is a container for holding components that support different functionality as shown in Figure 20. Each of these components (i.e., Image server and Feature server) makes up a server type inside the ArcIMS Spatial Server.

2.2.6.5. ArcIMS Requests/Responses and the WMS Connector

When an ArcIMS request is made, it is first handled by the Web server, passed through one of the connectors, and then forwarded to the ArcIMS Application Server. The Application Server, in turn, dispatches the request to an ArcIMS Spatial Server for processing. The request/response cycle is shown in Figure 20, below.





Step	*Action*
1	Client sends a request to an ArcIMS site.
2	The Web Server receives the request and passes it to the Servlet Connector or ColdFusion/ASP servers, which in turn hand the request to a connector.
3	The Connector opens a path for the ArcIMS Application Server to respond, and the request is handed from the connector to the Application Server.
4	The Application Server sends the request to an available Spatial Server within a Virtual Server group.
5	The Spatial Server generates the response as a • Response XML string (such as query results or an image location) • Stream of data
6	Response returns through the reverse order of the initial request.

Figure 20: Steps Taken by ArcIMS to Process a Request and Send a Response (courtesy of ESRI)

Since the Application Server can only process requests written in ArcXML[37], connectors are needed to either pass the ArcXML straight through or translate a WMS request (or third party syntax such as ColdFusion, ASP, or JSP) prior to forwarding the ArcXML request to the Application Server. Thus, the connectors provide a communication pipeline between a Web server or third party application server and the ArcIMS Application Server. The connectors communicate with the application server through TCP.

ArcIMS has four connectors, but we concentrate here on the WMS Connector, which processes WMS requests. This connector allows any OGC WMS-compatible browser or client the ability to access an ArcIMS MapService.

[37] *The ArcXML Programmer's Reference* http://support.esri.com/

2.3. User Experiences Examples

2.3.1. User Experience Example 1: Wupperverband

The following user experience is about publishing data through a WMS service by using ArcIMS 4 with its inherent WMS Connector, and about integrating published data in an ArcIMS Web Client.

In the late 1990s, the Wupperverband – a German water body authority, operator of dams and sewage plants – started building a GIS and recording geodata. Soon it turned out that there was a need for external geodata from many other authorities and companies. Wupperverband had to deal with more than 20 communities, state and county offices and another more than 20 utilities and sewage companies within their field of responsibility – a region of over 800 square kilometers in the densely populated south of the German Ruhrgebiet. The partners used GIS products from ESRI, SICAD, Smallworld and many others, resulting in data exchange problems.

The implementation of the GIS system at the Wupperverband has been managed by CSC Ploenzke since the beginning of the year 2000. By the end of that year, Wupperverband and CSC Ploenzke had developed the idea of solving the import/export, conversion and update problems of the external data by using the new OGC WMS standard. For the last two years, a pilot project has been running in Germany trying to integrate the commercially available Internet Map Servers – ESRI ArcIMS[38] and SICAD IMS from SICAD Geomatics[39] using the OGC WMS Specification.

The vision was to communicate online with the partner authorities and companies to simply bypass all these problems. The first partner in this effort was rapidly found: the community of the town of Wuppertal.

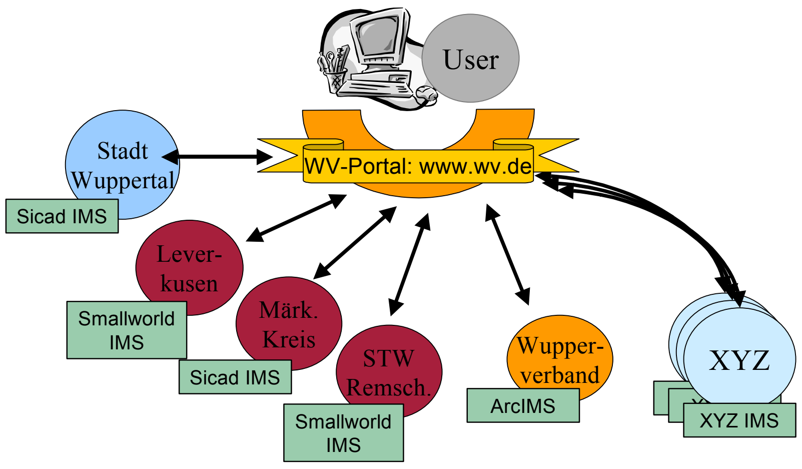


Figure 21: Wupperportal - the Vision of a WMS network in the Wupper region

They had used a SICAD based system for years and already had recorded lots of geodata. After some analysis of the content and technical aspects, a first implementation was realized in spring 2001 with a simple HTML approach. As a result, Germany's first practical WMS project went online. The financial effort for the realization was minimal, because the WMS extensions of ESRI and SICAD were provided free of charge.

[38] *ESRI ArcIMS Product Page: http://www.esri.com/software/arcims/index.html*
[39] *SICAD IMS Product Page (SICAD Internet Suite) http://www.sicad.com/pages/products/technology/sicad internet suite/index.html*

The service was improved in several small steps over a period of several months. In autumn 2002, the services were migrated from ArcIMS version 3.1 to 4.0 and from SICAD Internet Suite version 4.x to 5.1. Now, the WMS extension is a built-in component of ArcIMS 4. The service was improved to become more maintainable from hard coded HTML to generic Java Servlet technology. The capability to communicate with more than one external WMS service simultaneously is being developed now. The integration of other WMS services (e.g. rasterdata from the state office in Düsseldorf, based on AED's ALK/GIAP product) were tested successively and other authorities and companies showed interest in joining the network (e.g. the Wuppertaler Stadtwerke, using a Smallworld based Utilities GIS).

The project was presented for the potential partners and the local and technical press during a workshop event on November 12, 2002. Discussion focused on integration of these local services into the wider concept of the Geodata Infrastructure initiative of North Rhine Westphalia[40]. The Vision of the online geodata network WUPPER (Wupperportal) portrayed in Figure 21 above got closer and the project has begun to get connected to statewide and global networks.

2.3.1.1. Publishing your Data as a WMS Service

If you want to provide your GIS data to others, independent from their GIS software product, in a read-only mode, and in a visual form that you predefine, then a WMS service is what you need. By using a WMS service, your customers/users can use your data as you provide it without the need for any data conversions. This will get rid of the need to export your data to common GIS formats like DXF or Shapefile.

One advantage of using a WMS service is that your customers will not have to worry about the data being outdated. This is because your customers will work on the most up-to-date data through the WMS service, as opposed to working with local copies or replicated geo-databases that are often outdated data.

If you are a data provider, you can use WMS services as an advertisement for your data products. Users can see them and – if you implemented it – can get attribute information, but they cannot steal your data, copy it or manipulate it for internal or commercial purposes.

2.3.1.2. Integrating Published Data in your ArcIMS Web Client

Without WMS, in general, as a user, if you need to look at external geodata for your work, you have to ask the different data providers to export this data in a compatible format. Then you have to import the data into your system and configure the visualization. This is time consuming and very challenging. In most cases, you will be faced with a loss of information during the conversion process and the received data will soon be out of date. For these reasons, you often will determine that this is inefficient and not bother with it when you can avoid it.

Having a WMS compliant client, you can access different data from different data providers dynamically with no conversion effort. You can combine GIS data from all over the world and maybe find useful sources you never thought of before.

If you want to integrate published data from other data-vendors via a WMS service into your ArcIMS installation, you are not required to install and configure the ArcIMS OGC WMS Connector. Instead, you'll have to configure your Web client.

Version 4.0 of the ArcIMS Installation CD contains a bunch of sample clients, which are free for use and which are thoughts and ideas regarding what you can do and what you might want to do with your ArcIMS. Besides samples for the different connectors, the ArcIMS CD also contains a couple of samples

[40] *GDI NRW: http://gdi.ecc-gmbh.de/*

for the standard ServletConnector consisting of nothing more than HTML- and JavaScript-files. These samples use the standard ServletConnector and have the big advantage of requiring no additional software on the client's computer to run without the web browser (e.g. Java Runtime Environment, ArcExplorer). Also, there is no need for a big web application to run on the server (e.g. an Active Server Pages ASP-Application).

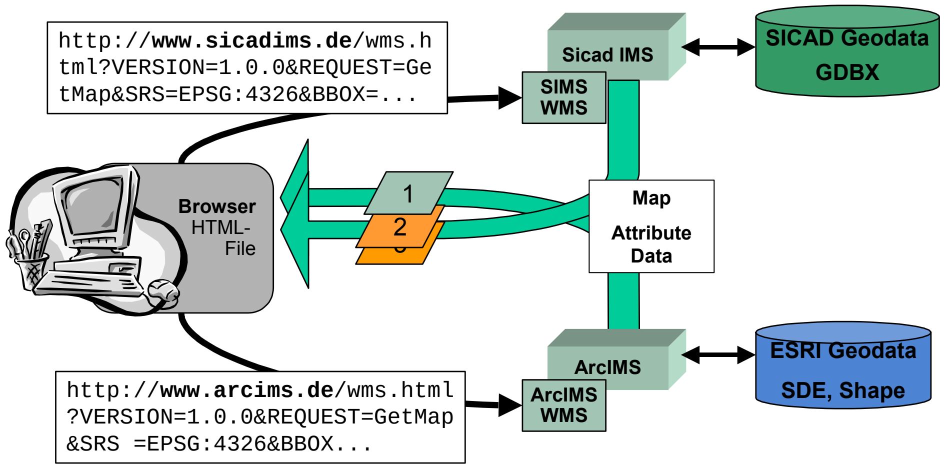

http://www.sicadims.de/wms.html?VERSION=1.0.0&REQUEST=GetMap&SRS=EPSG:4326&BBOX=...
Sicad IMS
SIMS WMS
SICAD Geodata GDBX
Browser HTML-File
1
2
3
Map
Attribute Data
ArcIMS
ArcIMS WMS
ESRI Geodata SDE, Shape
http://www.arcims.de/wms.html?VERSION=1.0.0&REQUEST=GetMap&SRS =EPSG:4326&BBOX...

Figure 22: WMS Architecture and Data Flow in the Wuppertal project

The disadvantage is that there exists only one client demonstrating the functionality of the ArcIMS WMS connector, which was not developed to do anything except to act as the WMS connector. Also, the other samples were developed to use the proprietary interface language of ArcIMS - ArcXML (sending and receiving ArcXML requests as the communication protocol). Because of these disadvantages, CSC Ploenzke developed its own enhanced client application, described below.

2.3.1.3. CSC's Ploenzke enhanced Client Application

The need for Web enhanced client applications comes from the lack of a Web client being able to use both:

- ArcXML as a protocol for ArcIMS MapServices revealing all the functionality ArcIMS offers (which is quite a bit more than that of the WMS interface) and
- WMS syntax to connect to every WMS service available on the Internet.

Such a Web client would incorporate the better of two worlds: the vast functionality of ArcIMS and the incredible possibilities offered by open standards allowing you to use data from everybody publishing it – no matter what software or geodata structure is used in the background.

The CSC Ploenzke evolving Web client application is an enhancement of one of the standard HTML viewers provided with ArcIMS 4.0. (This is the template of HTML- and JavaScript-files, which is generated if you are using the ArcIMS Designer application to create your MapService Website.)

After the installation of the ArcIMS CD, you will be able to use the HTML viewer "out of the box."

The CSC Ploenzke client application uses a simple Java applet that will run on most browsers (version 4 or higher) without having to install a plug-in. This applet is used for interaction with two additional Java Servlets running on the already used Servlet Engine. The Java Servlets allow the user to connect to any WMS service and to integrate their data into the client interface.

Specifically, the user will be able to:

- See a list of available WMS layers as part of the layer tree (through the `GetCapabilities` request)
- Obtain metadata for each of the provided WMS layers
- Control which WMS layer(s) are or are not to be viewed
- Draw the WMS map above or below the map generated by ArcIMS (through the `GetMap` request)
- View feature information of a certain WMS layer (through the `GetFeatureInfo` request

As a result, the user will be able to use all of the functions provided with the standard HTML-client and use all possible functions available through the WMS interface. Both of these functions are smoothly integrated to the CSC Ploenzke interface.

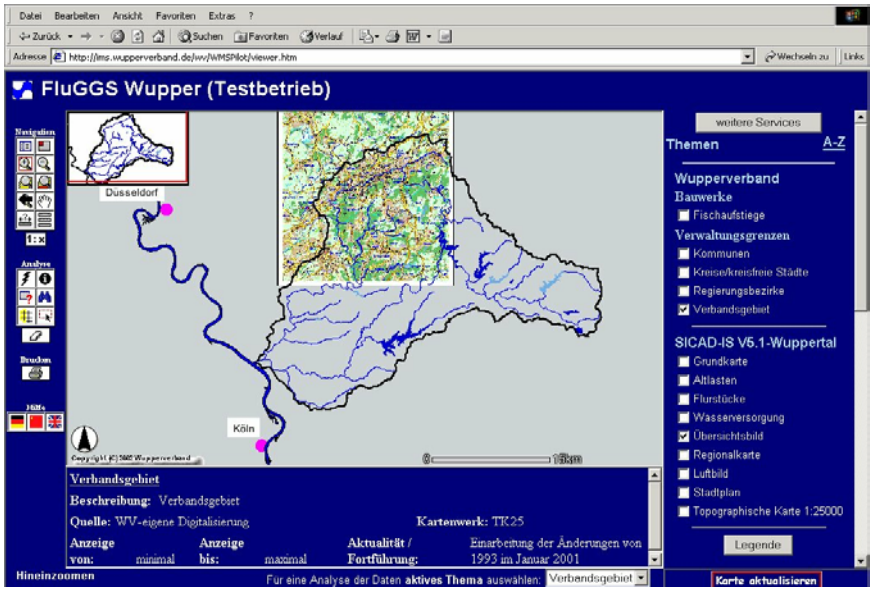
The option of offering customized Web clients to different groups was taken into account, for example:

- Using the WMS functionality with one MapService and not using it with another MapService or
- Using localized versions in different languages.

You will be able to manipulate the client appropriately within minutes.

With the current CSC Ploenzke Web application, you can only work on one ArcIMS service and connect to one WMS service at a time. You can change the WMS service you are connected to within one session, but you cannot connect to two WMS services at the same time. The ability to connect to multiple WMS services as well as the ability to connect to multiple ArcIMS services at the same time is being implemented by CSC Ploenke.

Note: If you are interested in using the viewer described here, please contact Roland Stahl (see Submitting Contact Points).


Datei Bearbeiten Ansicht Favoriten Extras ?
Zurück Suchen Favoriten Verlauf
Adresse http://ims.wupperverband.de/wv/WMSPilot/viewer.htm
Wechseln zu Links
FluGGS Wupper (Testbetrieb)
Navigation
Analyse
Drucken
Hilfe
Düsseldorf
Köln
weitere Services
Themen A-Z
Legende
Hineinzoomen
Karte aktualisieren

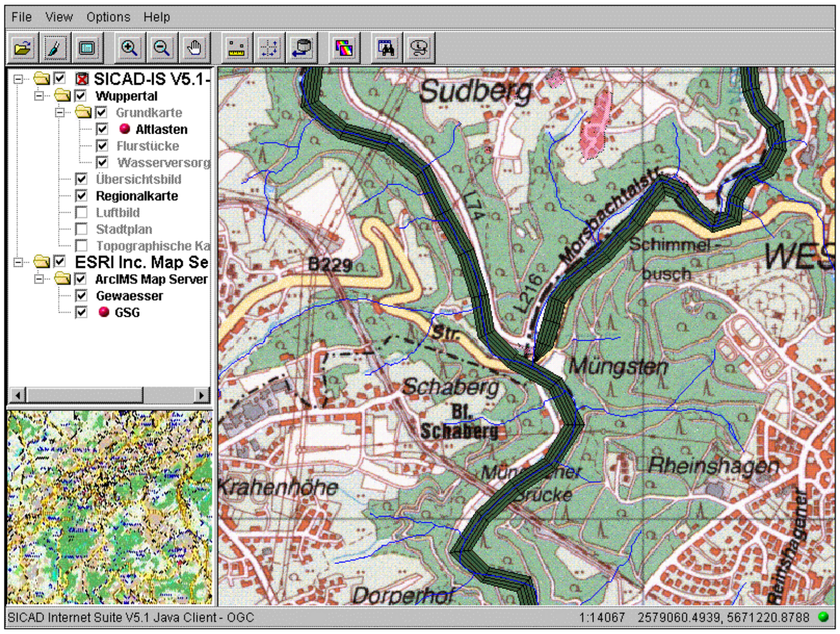
Wupperverband
Bauwerke
☐ Fischaufstiege
Verwaltungsgrenzen
☐ Kommunen
☐ Kreise/kreisfreie Städte
☐ Regierungsbezirke
☑ Verbandsgebiet

SICAD-IS V5.1-Wuppertal
☐ Grundkarte
☐ Altlasten
☐ Flurstücke
☐ Wasserversorgung
☑ Übersichtsbild
☐ Regionalkarte
☐ Luftbild
☐ Stadtplan
☐ Topographische Karte 1:25000

Verbandsgebiet
Beschreibung: Verbandsgebiet
Quelle: WV-eigene Digitalisierung
Kartenwerk: TK25
Anzeige von: minimal
Anzeige bis: maximal
Aktualität / Fortführung: Einarbeitung der Änderungen von 1993 im Januar 2001

Für eine Analyse der Daten **aktives Thema** auswählen: Verbandsgebiet

Figure 23: WMS Service example screenshot of combined geodata from SICAD and ESRI Internet Map Servers through the WMS interface using the enhanced ESRI HTML



☑ SICAD-IS V5.1-
☑ Wuppertal
☑ Grundkarte
☑ Altlasten
☑ Flurstücke
☑ Wasserversorg
☑ Übersichtsbild
☑ Regionalkarte
☐ Luftbild
☐ Stadtplan
☐ Topographische Ka
☑ ESRI Inc. Map Se
☑ ArcIMS Map Server
☑ Gewaesser
☑ GSG

SICAD Internet Suite V5.1 Java Client - OGC 1:14067 2579060.4939, 5671220.8788

Figure 24: WMS Service example screenshot of combined geodata from SICAD and ESRI Internet Map Servers

2.3.1.4. ArcIMS Tuning Recommendations

The `GetCapabilities` document is generated by default by the ArcIMS WMS connector as long as you do not have an xml-file stored under your capabilities directory using the name of the requested MapService as the filename. For example, if you have the MapService "SanFrancisco" running and you have a XML-file named `SanFrancisco.xml` stored at

```
c:/ArcIMS/capabilities/
```

The WMS connector will provide this file as a result of a `GetCapabilities` request. If the WMS connector does not find such a file, it will create one from scratch.

It is up to the user to decide if he/she wants to provide a manipulated, but more detailed and exact, `GetCapabilities` document with the disadvantage of having to manually manipulate that file every time the underlying MapService is changed.

If you want to create your own `GetCapabilities` document, it is recommended just to save the file in the capabilities directory provided by ArcIMS and to edit it manually using a text editor (e.g., Notepad). Here is a list of parameters (xml-tags) you might want to check/change in the `GetCapabilities` document to better offer your MapService to the public.

Depending on the end-users of your ArcIMS MapService you will probably want to offer the file in another encoding, in addition to the UTF-8 encoding:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
```

You probably want to offer your `Capabilities` file from your server and to make it available independent of the digitalearth server. Download the DTD-file to your file system, store it in the capabilities directory and write down the appropriate path to the XML-file, replacing the "`http://www.digitalearth.gov...`":

```
<!DOCTYPE WMT_MS_Capabilities SYSTEM
"http://www.digitalearth.gov/wmt/xml/capabilities_1_1_0.dtd">
```

You can change the entries of the following tags to better describe the MapService you are offering:

```
<Service>
<Name>OGC:WMS</Name>
<Title>ArcIMS WMS 1.1.0 Map Server</Title>
<Abstract> </Abstract>
```

Make sure that the value of the `xlink:href` attribute in the `<OnlineResource>` tag points to the exact MapService. If the link ends with a question mark (`com.esri.wms.Esrimap?`), it will point to the MapService you have defined as defaultService in the file "`WMSEsrimap_prop`". If you use more than one MapService, make sure the link ends with the specific name of your MapService:

```
com.esri.wms.Esrimap?ServiceName=<MapServiceName>
```

You should make sure that the only file format which is listed here is the format your MapService is using. The format can be found and changed using ArcIMS Administrator.

```
<Capability><Request><GetMap>
<Format>image/png</Format>
```

If you check the layers offered by your MapService, you will see that none of them is offering to be queryable, meaning no `GetFeatureInfo` capability on your layers. If you are trying to make such a request by hand (typing a get Feature request as a URL into your browser), you will see that you can successfully do such a request. If you want to let people know that they can make `GetFeatureInfo` requests, add the attribute queryable to each layer-tag that is describing data:

```
<Layer queryable="1">
```

You will probably have specified a minimum or maximum extent for layers in your MapService to provide the valuable information at which scale it should be available and drawn onto a map. This information gets lost when the `GetCapabilities` document is generated automatically by default. To pass this information to the public, add the following tag to each layer you want:

```
<ScaleHint min="xxx" max="xxx" />
```

To create appropriate min and max values, you can at first use the rough rule of thumb to convert the minimum and maximum values you specified in the MapService's axl-file (the map configuration file). Open the file in an (xml- or text) editor and find the minimum and maximum values for each layer. Use this formula to get the values to write in the `ScaleHint-tag: axl_minimum / 3846.15 = scalehint_min`

For example, if your minimum value in the axl-file is 1:20.000, you will divide it by 3846.15 resulting approximately in 5.2, which will be the value for min in the ScaleHint. The same will be required for the maximum value.

This solution is not meant to be geodetically precise, but it does the job in most cases.

```
<Capability><Layer><Layer>
<SRS> and <LatLonBoundingBox>
```

There are some remarks to add regarding coordinate information of layers available through the WMS interface:

- Although ArcIMS provides the possibility of Map projections on the fly, it is not supported with the WMS interface, that is if you request a layer, the only transformation you will get is that which you have defined in the axl-file of your MapService (if you have not defined one, the default one will be taken). If you want to provide your data in more than one coordinate space, you will have to create a MapService for each coordinate system you want to use. In addition, you will have to create a `GetCapabilities` document for each of these MapServices with the different coordinate systems.

- As long as your MapService is in EPSG:4326, the values in tag `LatLonBoundingBox` are correct. If you change your MapService to another coordinate system, the values are updated but are not correct. The WMS Specification states that the values provided with the tag `LatLonBoundingBox` indicate the edges of the enclosing rectangle in latitude/longitude decimal degrees as in SRS EPSG:4326 [WGS1984 lat/lon]. That is why you should change the values back to lat/lon degrees.

- It is recommended that you add the tag BoundingBox to every layer which is to be used for a specific SRS:

```
<BoundingBox SRS="EPSG:your_epsg_number" minx="xxx"
miny="xxx" maxx="xxx" maxy="xxx" />
```

2.3.1.5. Troubleshooting

Check the WMS log-file "WMSEsrimap.log" located under:

```
c:\ArcIMS\workingdir
```

If you have problems getting the `GetCapabilities` document, try using the tool "wget"[41].

2.3.2. *User Exerience Example 2: Wireless Info Project*

The following user experience is about using the WMS interface for Web mapping on mobile devices.

The WirelessInfo project focuses on the development of a planning support system for the evaluation of landscape changes. The whole project is oriented at the connection of GIS systems (including GPS technologies, remote sensing, photogrammetry, 3D modeling) with wireless communication based on utilization of 2G and 2.5G networks (GSM, GPRS, HSDCS). The main objective is the exploitation of wireless communication for decreasing the data transport (server - terrain, terrain - server) time between data collection and data processing.

The central aim of the work effort is to find the most effective way to obtain data for Web mapping using wireless devices. In many applications with mobile data access, it is necessary to have parallel data access. The data sources are usually distributed among many organizations (many data servers) which collect this data. Up till now, this situation was solved by replication of the same data in different institutions. But here the task of modifying a map or creating new features in the most convenient and exact way is in question. Another important point was to obtain the most effective dual mode communication with the University of Minnesota (UMN) MapServer. A mobile UMN MapServer public client was implemented to address this.

The project solutions were based on INSPIRE[42] recommendations and utilization of OGC standards. This ensures base conditions for interoperability of GIS services. Specifically, the solution is based on WMS server cascades. This type of solution offers a more operative data access anywhere at anytime.

2.3.2.1. Solution Outline & Testing Overview

The WirelessInfo solution is considered optimal for several reasons:

- Mobile device gateway is already developed and running inside of a project
- Implementation of OGC standards
- Open solution with AGILE community of the developers

A variety of GIS software, wireless devices, and GPS receivers were analyzed and compared, with due consideration for the fact that geographical data standards are different in the Czech Republic from those in other countries. To provide tools for 3D analysis and remote sensing analysis, a direct link between UMN MapServer and another Open Source solution, GRASS, was established.

At this stage, the participants[43] of the WirelessInfo project are testing and establishing the proposed structure on an Intranet environment. They are simulating a network of WMS enabled data sources with cascading WMS server, such as the UMN Mapserver. WMS cascade servers play a role of hubs of WMS networks and offer the possibility to create a gateway for mobile devices that use different sources of geographic information.

For mobile systems, two approaches need to be considered. The first is shown in Figure 25:

[41] *wget tool: http://wget.sunsite.dk/*
[42] *Towards a European Spatial Data Infrastructure: http://195.228.254.144/Eloadasok/Stream1/Wednesday_11hr/Alessandro_Annoni_11_str1/GSDI6_26_Annoni.pdf.*
[43] *Participants in the WirelessInfo project included: UV, SPACECAD, INGR, HSRS, and Lesprojekt.*

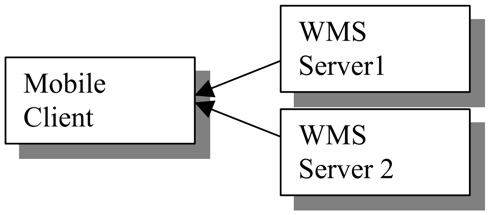


Figure 25: DHTML/CSS mobile client

Advantage:

- Servers work in parallel, so there is faster response from the server side.

Disadvantages:

- The CSS explorer is necessary on the terminal side. Many current mobile terminals do not have this functionality. There is a higher processing load on client side.
- It is necessary to obtain information from all servers in the some coordinate system.
- A large amount of data is transferred, which could be very important using GPRS technology.

The second approach is shown in Figure 26:

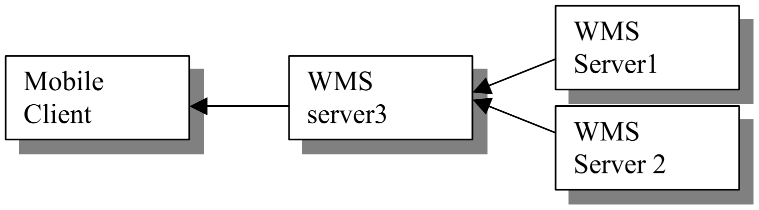


Figure 26: Cascading Server using Mapserver

Advantages:

- Low processing load on the client side (HTML is enough).
- Solution supports Java clients.
- It is possible to use data from servers that use different coordinate systems, Coordinate Transformation Web Services are used by server3.

Disadvantages:

- It is necessary to have one more server
- The data are downloaded after downloading of all data on the server.

After analysis, it was decided to use the first model only in the case where PC's were used as field computers. In this case a thick client could be used and part of the data could be stored on a field server and WMS services could run on the field server. Such systems work partly independent of the GPRS cellular network, and there usually is a large amount of data that is already downloaded onto the field server. For handheld type computers, there is a recommendation to use the second solution model, which is independent of a particular technology.

Figure 27 below shows a sketch of the structure of the proposed WirelessInfo network.

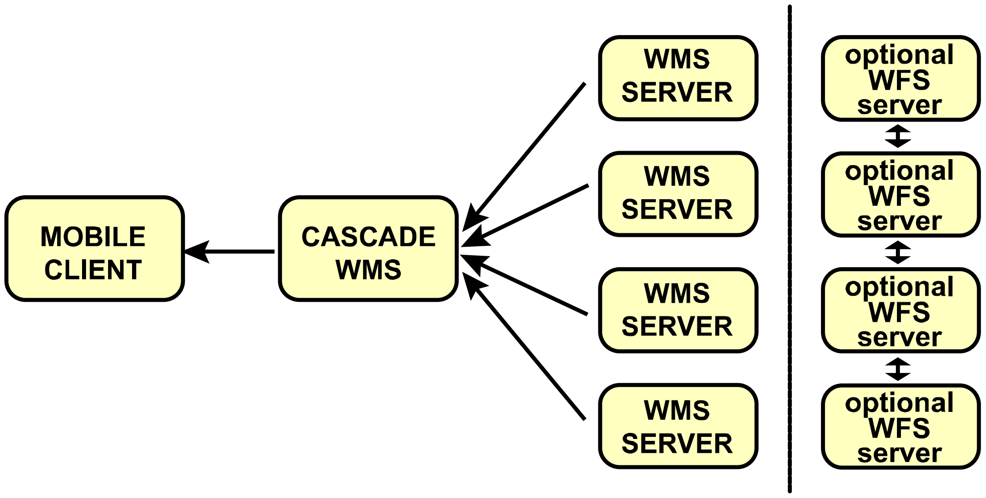


Figure 27: Proposed Structure of the WirelessInfo Network

In terms of network nodes, WirelesInfo is testing all WMS solutions that can be used by current data providers. A key challenge that resulted during testing is spatial projection system handling. This is, of course, a necessary condition for geodata interoperability. The problem is the absence of some national projections in the EPSG list[44]. In the case of the Czech Republic, this is the S-JTSK system, which is widely used by state administrators. During testing, data was transformed into WGS84 coordinates. Efficiency of the transformation needs to be resolved.

2.3.2.2. Pilot testing in Forest Management Institute

The current firewall system in FMI Brandys n. L. is based on the VPN-1 Enterprise FireWall, which covers license to unlimited numbers of users and also contains a management module for engineering additional distributed modules of firewall. There are good conditions for using the protective modules in external subjects and fixation of two functions:

- Effective protection of external subject network by the high-tech
- protection system with opportunity costs.
- Creation of protected communication data line among the external
- subjects and the central evidence for data transfer by any TCP/IP protocol.

[44] *See OpenGIS Coordinate Transformation Services Implementation Specification: Revision 1.00 OpenGIS Project Document 01-009.*

2.3.2.2.1. Testing tools:

- Phone Card 2.0 with enabled data service
- SW Driver for Phone Card 2.0
- Notebook PCMCIA with OS Windows 2000SW VPN Secure Client FW
- Service Setup for card phone 2.0

2.3.2.2.2. Set-up:

- Installation of SW Driver for Phone Card 2.
- Device connection to PCMCIA slot.
- System configurations to FW for authorisation access to Firewall from the Internet.
- Connectivity verification to data service provider (Eurotel).
- Installation of FW Secure Client to notebook with W2000.
- System configuration for connection to Firewall and authorisation to system Firewall.
- Verification and connection to UHULnet.
- Wireless connection of distant client to FMI network through VPN.

2.3.2.2.3. Hardware and software equipment required:

- Module Nokia Card Phone 2.0
- Corresponding software for Module Nokia Card Phone 2.0 handling
- Activation of data services on the corresponding SIM card for certain providers of the mobile data data network is necessary.
- Firewall software for secure remote client

2.3.2.2.4. Self-connection approach:

- Installation of Module Nokia Card Phone 2.0
- Plug-in of Module Nokia Card Phone 2.0 during installation
- Reset of the system
- Setting of baud rate in SW Nokia Card Phone 2.0 system
- Installation of SW Firewall Secure Remote client
- Reset of system

- Configuration of connection to data network mobile provider
- Configuration of SW Firewall Secure Remote client
- Self connection to network

Using GPRS data service, it was discovered that GPRS Eurotel data service does not support the IPSec protocol. Testing of this service should debug different forms of access to data, methods and forms of data storage for different server types (UMN MapServer and Intergraph GeoMedia WebMap). The goal is to create an optimized communication environment and to navigate a user so that the user maximally exploits the prepared data. In addition, the goal was to have the data stay at the data collector.

2.3.3. User Experience Example 3: Harvard University WMS (Distributed) Servers

The following reflects the experience of the computer science GIS group at the Harvard Division of Continuing Education (DCE) in setting up and testing distributed Web mapping servers.

Harvard's distributed geospatial Web environment considers the deployment of map services powered by different commercial map servers (ESRI ArcIMS, Intergraph GeoMedia WebMap) and open source (i.e., PostGIS/PostgreSQL, FreeBSD). Further installations and testing will be documented at Harvard's interoperability page[45].

The GIS group built a WMS server according to the OGC WMS 1.1.1 specification. The server is installed in a FreeBSD operating system for testing purposes. The GIS group plans to develop interoperable applications that will be tested in diverse environments for a multiplicity of users, both within the university and outside.

2.3.3.1. Steps for Installing OGC WMS Server on FreeBSD

This section describes the installation of an OGC WMS server (WMS 1.1.0. compliant) in a PC with a FreeBSD operating system. Many of the libraries that OGC WMS needs to have installed are available on FreeBSD under the ports system for FreeBSD[46].

This section shows how to install OGC WMS with many of the optional libraries and features. This procedure can also probably be used for other BSD based systems that make use of the 'ports' system for installing 3rd party packages. The ports system is a package management system for handling the retrieving, compiling, and installation of software that is not included with the base FreeBSD system. It is very similar to Redhat's RPM package format. For more information refer to FreeBSD system ports page[47].

These instructions will not be helpful to Linux users as the package management tools are different for Linux users (or other UNIX flavors for that matter). The following libraries are extra libraries with which WMS can link with Proj.4 (Cartographic Projection Library, necessary for WMS Clients)[48]:

- libwww (necessary for WMS Clients)
- libjpeg

[45] *Harvard's interoperability page: http://www.gis.harvard.edu/interoperability*
[46] *FreeBSD: http://www.freebsd.org*
[47] *FreeBSD System Ports: http://www.freebsd.org/ports*
[48] *Proj.4 (cartographic projection Library): http://www.remotesensing.org/proj/*

- libpng
- libgeotiff
- libgif
- freetype
- gdal
- ogr
- postgis (Libraries for interfacing with Postgresql library)

For additional reference, you can also look at the information on compiling an OGC WMS for UNIX from *UMN Mapserver HOWTO4*[49].

NOTE: These instructions assume you have root privileges on the FreeBSD operating system. If you do not have root privileges, then these instructions don't directly apply to your environment.

Step 1: Install FreeBSD onto your system. Installation instructions for FreeBSD can be found in the FreeBSD handbook[50].

Step 2: Install the ports system during your FreeBSD installation. Although it is not required, we recommend that you update your ports collection tree on your FreeBSD system. This will update all the available 3rd party packages that FreeBSD knows of and put it on the system.

Step 3: Install the gdal port. It is located under /usr/ports/graphics/gdal. Here is a sample session of installing the gdal port:

```
> cd /usr/ports/graphics/gdal
> make
[Lots of output]
> make install
[Lots of output]
> make clean
[Lots of output]
```

Step 4: gdal will install a large set of the libraries that you will need. In particular, it will install the following:

- libpng
- libtiff
- libgeotiff
- libjpeg
- libgif

[49] *Compiling OGC WMS for Unix (from UMN Map Server): http://mapserver.gis.umn.eedu/doc36/unix-install-howto.html*
[50] *FreeBSD handbook: http://www.freebsd.org/handbook*

In addition, gdal will install the ogr libraries for you automatically. This process will take some time depending on the speed of your system. Expect a couple of hours for building everything on a Pentium 2400Mhz-based system.

Step 5: Install libwww. This is under /usr/ports/www/libwww. Here is a sample session of what you would type in at the prompt to get it installed:

```
> cd /usr/ports/www/libwww
> make
[Lots of output]
> make install
[Lots of output]
> make clean
[Lots of output]
```

Step 6: Install the freetype library. This is under /usr/ports/print/freetype. Here is a sample installation; run:

```
> cd /usr/ports/print/freetype
[Lots of output]
> make
[Lots of output]
> make install
[Lots of output]
> make clean
[Lots of output]
```

Step 7: This will install the gd2 library. This is under /usr/ports/graphics/gd2.

Here is a sample installation; run:

```
> cd /usr/ports/graphics/gd2
[Lots of output]
> make
[Lots of output]
> make install
[Lots of output]
> make clean
[Lots of output]
```

Step 8: This will install a database backend that WMS servers can connect with. The database used will be PostgreSQL.

Here is a sample installation; run:

```
> cd /usr/ports/databases/postgresql7
[Lots of output]
> make
[Lots of output]
> make install
[Lots of output]
> make clean
[Lots of output]
```

Step 9: Download, install, and configure PostGIS and Proj.4 libraries. At the time of this writing, the latest releases are Proj.4 4.4.6 libraries[51] and PostGIS release 0.7.4[52].

Follow the URLs given above and download these to your FreeBSD system. For this set, it is recommended to place the downloaded archives in the directory `/usr/local/src` and open them with the tar command in `/usr/local/src.`

Once there, you can cd to the subdirectory that is created for each of them and follow the instructions in the INSTALL or README files to install the libraries onto your system. The defaults for installation should be fine. An example for installing PostGIS is given below:

```
> cd /usr/local/src/postgis
> ./configure
[Lots of output]
> make
[Lots of output]
> make install
[Lots of output]
```

Step 10: You will also need a Web server to serve HTTP requests. Install the Apache Web server (or use another if preferred)[53]. Here is a sample run for installing Apache. Apache is in the ports collection under `/usr/local/www/apache13`.

An example run follows:

```
> cd /usr/ports/www/apache13
[Lots of output]
> make
[Lots of output]
> make install
[Lots of output]
> make clean
[Lots of output]
```

Step 11: Download the UMN MapServer source archive into `/usr/local/src` (or in another preferred area). At the time of this writing the latest release of UMN Mapserver is 3.6.4[54]. Unarchive the tar file and cd into that directory.

Configuration of WMS server: you want to notify the UMN MapServer of all the available libraries since WMS has problems auto-detecting them properly if you just run the configure script.

Below is a sample for enabling all of the available libraries you installed:

```
> cd /usr/local/src/mapserver
> ./configure --with-jpeg=/usr/local --with-gd=/usr/local \
--with-freetype=/usr/local --with-wmsclient --with-proj=/usr/local
\
--with-postgis=/usr/local
--with-gdal=/usr/local
--with-ogr=/usr/local
```

[51] *Proj. 4 4.4.6 libraries: ftp://ftp.remotesensing.org/pub/proj/proj-4.4.6.tar.gz*
[52] *http://postgis.refractions.net/postgis-0.7.4 tar.gz*
[53] *Apache Web server installation: http://httpd.apache.org/docs/*
[54] *UMN MapServer (version 3.6.4) download http://mapserver.gis.umn.edu/dist/mapserver-3.6.4.tar.gz*

```
[Lots of output]
> make
[Lots of output]
```

Step 12: The previous step created the UMN MapServer executable called mapserver. Copy this into the cgi-bin directory of Apache. By default, this is located in `/usr/local/www/cgi-bin> cp mapserver /usr/local/www/cgi-bin`

Step 13: From this point, you can follow the MapServer demo tutorial53[55] to get started working with the UMN MapServer and customizing it to your needs.

2.3.4. Experience with ESRI ArcIMS WMS

The WMS server directly is powered by ArcIMS 3.1 spatial engine (a new ArcIMS 4.0.1 is installed on a UNIX Solaris OS). ArcIMS 3.1 spatial engine allowed us to develop a collaborative Web-based GIS tool for teaching and training called Map Events Tool. The major difference between ArcIMS version 3.1 and version 4.0.1 is that the latest version has the WMS interface built in for easing the installation of the OGC WMS connector. If you have ArcIMS 3.1, you need to download the WMS connector from ESRI's interoperability page[56].

The ESRI ArcIMS OGC WMS connector works properly with ArcIMS 3.1, Apache3, Tomcat 3.2.1, or installed through ArcIMS4.0.1 on Solaris operating system.

2.3.5. Experience with Intergraph GeoMedia WebMap Professional

Note: For the steps on installing and customizing the GeoMedia WebMap WMS adaptor see Intergraph's contribution in Chapter 3, Recipe 4.

2.3.5.1. Recommendations

For New Users: WebMap is tightly integrated with IIS. Although it is possible to get GeoMedia WebMap Professional to work with other servers it is recommended that new users stick with using IIS as the default Web server.

Security: Make sure to update and patch your Windows system and install the latest patches for IIS to ensure that your machine is not vulnerable to any Web based viruses that can affect IIS[57]. It is also recommended that you use the IIS Lockdown Tool to make sure that IIS's default security settings are tightened up.

Cache: Create a cache directory for GeoMedia WebMap Professional from the IIS Administration Tool. Create a new virtual directory under IIS, alias: cache

Directory: E:\inetpub\wwwroot\cache. Change this to suit where your setup is Properties: Read, Run Script.

[55] *UMN Mapserver tutorial: http://mapserver.gis.umn.edu/doc36/demo_readme.html*
[56] *ESRI's InteroperabilityWebsite: http://www.esri.com/software/opengis/interopdownload.html*
[57] *Window's update: http://windowsupdate.microsoft.com, Windows IIS patches: http://www.microsoft.com/technet/treeview/default.asp?url=/technet/prodtechnol/iis/Dedfauly.asp*

Ensure that WebMap is aware of the newly created cache directory by going to WebMap Administrator -> Tools -> Edit System Settings.

2.3.5.2. Acknowledgements

Alain Hoang, a Harvard DCE graduate student, and comments from students from GIS course CSCIE-40 at the Harvard DCE.

2.4. WMS Request Examples

2.4.1. Request Example: Get_Capabilities

The purpose of the `GetCapabilities` request is to declare the `GetMap` services that you provide. You must be able to deliver an XML metadata file via http upon receiving a request such as:

```
http://www.airesip.org/wms/process.cgi?REQUEST=GetCapabilities&VERSION=1
.1.1&SERVICE=WMS
```

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".

REQUEST: The parameter REQUEST=`GetCapabilities` indicates that this request is for metadata (rather than a map, i.e. `GetMap`). A sample XML metadata file is shown below:

```
<?xml version='1.0' encoding="UTF-8" standalone="no" ?>
<!DOCTYPE WMT_MS_Capabilities SYSTEM
"http://www.digitalearth.gov/wmt/xml/capabilities_1_1_0.dtd">
<WMT_MS_Capabilities version="1.1.0" updateSequence="0">
<Service>
<Name>Air ESIP</Name>
<Title>Air ESIP Map Server</Title>
<OnlineResource>http://www.airesip.org</OnlineResource>
</Service>
<Capability>
<Request>
<GetMap>
<Format><image/GIF /><image/JPEG /></Format>
<DCPType><HTTP><Get
onlineResource="http://www.airesip.org/cgi-bin/de.pl"
/></HTTP></DCPType>
</GetMap>
<GetCapabilities>
<Format><WMS_XML /></Format>
<DCPType><HTTP><Get onlineResource="http://www.airesip.org/cgi-
bin/de.xml" /></HTTP></DCPType>
</GetCapabilities>
</Request>
<Layer>
<SRS>EPSG:4326</SRS>
```

```
<Title>SeeAll Radiometer</Title>
<LatLonBoundingBox minx="-180" miny="-82" maxx="180" maxy="82" resx="1"
resy="1" / >
<Name>temperature</Name>
</Layer>
</Capability>
</WMT_MS_Capabilities>
```

Some of the key parts of this file are discussed below. The first three lines are header lines:

```
<?xml version='1.0' encoding="UTF-8" standalone="no" ?>
<!DOCTYPE WMT_MS_Capabilities SYSTEM
"http://www.digitalearth.gov/wmt/xml/capabilities_1_1_0.dtd">
<WMT_MS_Capabilities version="1.1.0" updateSequence="0">
```

The first line identifies the code as XML. The second line gives the URL for the DTD file. The third line provides version information, including an "updateSequence" number that is normally incremented every time a change has been made to the `Capabilities` file. This enables cataloging systems to determine quickly if an update to a server's holdings has occurred.

Next are `<Service>` and `<Capability>` tags. The `<Service>` must include at a minimum: `<Name>`, `<Title>`, and `<OnlineResource>` (a web page providing further description of the site).

```
<Service>
<Name>Air ESIP</Name>
<Title>Air ESIP Map Server</Title>
<OnlineResource>http://www.airesip.org</OnlineResource>
</Service>
```

The `<Capability>` tag (note that this tag name is singular) includes `<Request>` and `<Layer>` tags.

```
Capability>
<Request>
...
</Request>
<Layer>
...
</Layer>
</Capability>
```

`<Request>` consists of a `<Map>` and a `<Capabilities>` tag (note plural). The `<Map>` section describes the formats and request types available from the server for a `GetMap` request. The `<Capabilities>` tag describes what you get from a `GetCapabilities` request (which is the XML file itself).

```
<Request>
<Map>
<Format><image/GIF /><image/JPEG /></Format>
<DCPType><HTTP><Get onlineResource="http://www.airesip.org/cgi-
bin/de.pl" /></HTTP></DCPType>
</Map>
<Capabilities>
<Format><WMS_XML /></Format>
<DCPType><HTTP><Get onlineResource="http://www.airesip.org/cgi-
bin/de.xml" /></HTTP></DCPType>
</Capabilities>
</Request>
```

Each `<Layer>` tag (there can be multiples present in the file) describes a layer that the server can provide. The `<Name>` tag gives the actual name that appears in the `LAYER=` parameter of a `GetMap` request. If a SRS other than latitude-longitude is used, the `<LatLonBoundingBox>` boundary can be approximated, and a `<Bounding Box>` tag must be added to give the exact bounding box in that coordinate system.

```
<Layer>
<SRS>EPSG:4326</SRS>
<Title>SeeAll Radiometer</Title>
<LatLonBoundingBox minx="-180" miny="-82" maxx="180" maxy="82" resx="1"
resy="1" />
<Name>temperature</Name>
</Layer>
```

The Layer tag may include several parameters:

- `opaque=1` denotes that the map is mostly space-filling (generally the case with raster data) and should probably appear at the bottom of a stack of images.
- `queryable=1` denotes that `GetFeatureInfo` can be invoked on it.
- `fixedWidth=xxx` denotes that the map width is fixed to the specified value
- `fixedHeight=xxx` denotes that the map height is fixed to the specified value
- `nosubsets= 1` denotes that only the entire bounding box is accessible

2.4.1.1. Multiple Layers

In most cases, your server will be providing more than a single layer (variable). Each layer is listed within its own `<Layer>` tag in the `Capabilities` file. The `<Layer>` tags can be nested when common properties apply to multiple layers. In the nested case, it is permissible for a nested layer to override the parent description.

It is a requirement to enclose multiple layers in a single outer `<Layer>` element. Properties common to all layers may be listed there, for convenience. In particular, any SRSs common to all layers should be listed there.

2.4.1.2. Vendor Specific Parameters (Optional)

You may choose to allow additional parameters in a `GetMap` request that are specific to your server (e.g. `SPECTRAL_BAND`). This feature of your server is declared in the `Capabilities` XML file, inside the `<VendorSpecificCapabilities>` tags:

```
<VendorSpecificCapabilities>
<SPECTRAL_BAND>
<Title>Spectral Band</Title>
<Abstract>Spectral Band, expressed as a number between 1 and 7.
</Abstract>
</SPECTRAL_BAND>
</VendorSpecificCapabilities>
```

Web clients are never required to include such a vendor specific parameter value in their requests. Therefore, your server must have default value set for each of these parameters.

2.4.1.3. Exceptions

You can choose how to return error messages back to the client. The three options are:

- in an XML file (`EXCEPTIONS=application/vnd.ogc.se_xml`)
- as a message "painted" onto the returned image (`EXCEPTIONS=application/vnd.ogc.se_inimage`)
- as a last resort by returning a blank image (`EXCEPTIONS=application/vnd.ogc.se_blank`).

If you choose the "painted" (or "inimage") option, add the following three lines to your `Capabilities` file, within the `Capability` tag group of the `Capabilities` file.

```
<Exception>
<Format>application/vnd.ogc.se_inimage</Format>
</Exception>
```

An example XML response is given below. Its MIME type must be set to '`application/vnd.ogc.se_xml`', and placed within the `<Capability>` tag.

```
<WMTException version="1.1.0">
Text of error message <WMTException>
<Format><INIMAGE /></Format>
</Exception>
```

2.4.1.4. Time-Dependent Data

If you offer the same dataset at multiple time instances, you will want users to include a "`time`" parameter in their call. This parameter can be specified in many possible ways; some common examples are:

TIME=1999-12-23

TIME=1999-12-23T20:15

TIME=1999-12

TIME=1999-01-01,1999-04-01,1999-07-01,1999-10-01

TIME=1995-04-22T12:00/2000-06-21T12:00/P1D

The first example is for data on a particular day (December 23, 1999). The second form is more specific, as it references a particular time (20:15 on December 23, 1999). The third form is less specific, as it references a particular month. The fourth form is a request for four specific days of data.

The fifth form is a request for daily data (P1D refers to a period of one day) during the specified time interval. See http://www.digitalearth.gov/wmt/time.html for further time options.

To use Time for your datasets, you must declare Time to be a "Dimension" in the Capabilities file and then specify the time "extent" of the dataset. The following pair of lines are an example of what would be placed within a layer tag group in the Capabilities file:

```
<Dimension name="time" units="ISO8601" />

<Extent name="time" default="2000-12-24">1992-11-02/2000-12-
24/P5D</Extent>
```

In this example, the data are provided every 5 days between 11/02/92 and 12/24/00. A default date is provided for those who do not specify a date.

2.4.1.5. Vertical Dimensions

If you offer the same dataset at multiple elevations, you will want users to include an "elevation" parameter in their `GetMap` call. This parameter can be specified in many possible ways; the most common examples are:

```
ELEVATION=1000

ELEVATION=1000, 2000, 3000

ELEVATION=1000/5000/P1000
```

The first example is at an elevation of 1000 units. The second form is a request for three specific elevations. The third form is a request for every 1000 units between 1000 and 5000.

To use Elevation for your datasets, you must declare Elevation to be a "`Dimension`" in the Capabilities file and then specify the elevation "extent" of the dataset. The following pair of lines are an example of what would be placed within a layer tag group in the Capabilities file:

```
<Dimension name="elevation" units="EPSG:5030" />

<Extent name="elevation" default="0">0,2000,4000,6000,8000</Extent>
```

In this example, the data are provided every 2000 m between 0 and 8000. EPSG:5030 represents meters above the WGS84 ellipsoid. A default elevation is provided for those clients who do not specify it.

2.4.1.6. Additional Dimensions

In addition to 3-D space and time, your dataset may have other inherent "dimensions" such as spectral bands. In such cases you can create and declare dimensions of your own. For example, spectral band might be requested using calls such as:

```
BAND=0.8

BAND=0.6,0.8,1.8

BAND=0.8/1.8/P0.1
```

The first example is a request for a spectral band of 0.8 units. The second form is a request for three specific bands. The third form is a request for every 0.1 units between 0.8 and 1.8.

To use this (or any other dimension) in for your datasets, you must declare each to be a "Dimension" in the Capabilities file and then specify the "extent" of the dimension. The following pair of lines are an example of what would be placed within a layer tag group in the Capabilities file:

```
<Dimension name="band" units="micron" />
```

```
      <Extent name="band" default="0.8">0.6/2.4/P0.1</Extent>
```

In this example, the data are provided every 0.1 micron between 0.6 and 2.4 microns. Its default value is 0.8.

2.4.2. Request Example: Get_Map

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

```
http://www.airesip.org/wms/process.cgi?REQUEST=GetMap&FORMAT=image/gif&WIDTH=6
40&HEIGHT=480&

LAYERS=temperature&SRS=EPSG:4326&BBOX=-110.,40.,-
80.,30.&SERVICE=WMS&VERSION=1.1.1
```

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 client requests. 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. A description of each parameter is given below:

2.4.2.1. Request

The parameter REQUEST=`GetMap` indicates that this request is for a map (rather than for metadata or some other request).

2.4.2.2. Format

You choose which output formats to support. Permissible image formats are: image/gif, image/jpeg, image/tiff, image/geotiff, image/png, image/ppm, and image/wbmp. Note that TIFF and GeoTIFF support floating point data types as well as 16-bit integers, although not all TIFF/GeoTIFF readers support these features. You also can choose to support additional formats not listed here (see the OpenGIS specifications for details). In WMS Version 1.0, the "`image/`" parts of the format values were not included.

2.4.2.3. Width and Height

You must be able to provide images in any requested size. These values will be requested as positive integers.

2.4.2.4. Layers

You choose which layers (variables) to provide and how they are named. You must be able to receive a request for multiple layers (e.g. `LAYERS=temperature,coastline,cities`). In this example, the order is temperature, coastline, followed by cities.

2.4.2.5. SRS

Users choose the map projection or SRS in which to deliver their data products. The SRS is commonly expressed in terms of its EPSG numeric code. The standard lat-lon grid has a code of 4326, and would be sent as `SRS=EPSG:4326`[58]. Alternatively, the SRS can be expressed as a projection with user-defined parameters and units, known as an AUTO projection[59].

2.4.2.6. BBOX

The bounding box request is sent by the client as integers or floating point numbers (in units of the specified SRS). These values correspond to the outer edges of the boundary pixels (not the center of the boundary pixels). The aspect ratio inherent in this request need not match the aspect ratio of the width/height request, so the user must be able to "stretch" the image to satisfy both requirements.

2.4.2.7. Transparent

Users must be able to provide a transparent background image if the user requests it (using `TRANSPARENT=TRUE`) and the format is one that supports transparency. GIF and PNG support transparency, but JPEG does not. Note that GIF89 (rather than GIF87) is required to implement this feature for GIFs. Transparency enables other maps to be overlaid. If this parameter is not present in client requests, its value must be set to false.

2.4.2.8. BGColor

The user must be able to set the background (or missing value) color to the specified hex string (e.g. `BGCOLOR=0xFF0000`) if a user requests it. If this parameter is not present in client requests, its value must be set to white (0xFFFFFF).

2.4.2.9. Version

Currently, there are only two versions of WMS available, 1.1 (which can be listed as 1.1.1) and 1.0 (1.0.0). Your server will need to either support all official versions or support a "client-server negotiation" to meet a client request. All versions are listed at http://www.digitalearth.gov/wmt/xml. (In WMS 1.0, "VERSION" was called "WMTVER").

2.5. XSL/XSLT Stylesheet Examples

2.5.1. XSL/XSLT Stylesheet Example

This stylesheet is an example of how to use XSL functions for transforming one XML dialect to another XML dialect, one that is understood by the map server. This particular example is of the WMS `GetMap` request being transformed into ArcXML GetImage request. The intention is to show the structure of the XSLT stylesheet and the embedded functionality that can be applied to any XML documents transformations for producing the desired outcome.

[58] *Other EPSG Codes are listed in: http://www.petroconsultants.com/products/geodetic.html*
[59] *AUTO projections are described in http://www.digitalearth.gov/wmt/auto.html*

Note: XML tags are in black text, comments are in gray text, bold gray text indicates the major headings of the WMS `GetMap` request that are being transformed.

The original XML request is (simplified for purpose of this example):

```
<ARCXML version='1.1'>
 <REQUEST>
 <GET_IMAGE show="layers">
  <PROPERTIES>
   <FEATURECOORDSYS>
    <FILTERCOORDSYS>
   </FILTERCOORDSYS>
<ENVELOPE>
 minx=your_attribute, miny=your_attribute, maxx=your_attribute,
 maxy=your_attribute
</ENVELOPE>
<IMAGESIZE>
 width="500" height="500"
</IMAGESIZE>
 <LAYERLIST>
   <LAYERDEF id="scalebar" visible="true"
   <LAYERDEF id="label_name" visible="true" />
   <LAYERDEF id="STATE" visible="true" />
   <LAYERDEF id="COUNTY" visible="true" />
   <LAYERDEF id="CITY" visible="true" />
   <LAYERDEF id="ZIPCODE" visible="true" />
   <LAYERDEF id="TRACT" visible="true" />
   <LAYERDEF id="BLOCKGROUP" visible="true" />
   <LAYERDEF id="UserStyle" visible="true" />
 </LAYERLIST>
<OUTPUT>
 format='image/png', 'image/jpeg', 'image/gif'
  </OUTPUT>
   <BACKGROUND>
    color="255, 255, 255", transparent=”true”, transcolor=”255,255,255”
   </BACKGROUND>
</PROPERTIES>
<LAYER type="acetate" name="label_name" id="acetate">
 <OBJECT units="pixel">
  <TEXT>
   cords=”1 10”, label=”label_name”
   <TEXTMARKERSYMBOL/>
  </TEXT>
  </OBJECT>
 </LAYER>
<LAYER>
 <DATASET/>
 <VALUEMAPRENDERER lookupfield="WIDOWED">
 <VALUEMAPRENDERER>
  <EXACT value="132; 9; 25; 28; 79; 67; 205">
```

```
<SIMPLEPOLYGONSYMBOL fillcolor="88,100,27" filltype="solid"
filltransparency="1" " />
<!-- OR you can place a mark, “star,” on the selected feature layer
polygons -->
  <SIMPLEMARKERSYMBOL color="27,0,227" width="15" type="star" />
     </EXACT>
    </VALUEMAPRENDERER>
   </LAYER>
 </GET_IMAGE> </REQUEST></ARCXML>
```

The following XSLT stylesheet is for transforming the above mentioned XML dialect into a WMS `GetMap` compliant XML dialect:

```
<?xml version="1.0"?>
<xsl:transform xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
version="1.0"
        xmlns:xlink="http://www.w3.org/1999/xlink"
        xmlns:xalan="http://xml.apache.org/xslt">
       <!-- ADD your other reference or pointers to other stylesheets
here, if
       applicable -->
<xsl:template match="/"> <!-- root element -->
<aRCXML version='1.1'>
 <REQUEST>
 <GET_IMAGE show="layers"> <!-- ArcXML's request -->
  <PROPERTIES>
<!------- WMS BoundingBox srsName --------->
<!-- FEATURECOORDSYS: projection coordinate system that layers in your
MapService are projected to. -->
   <FEATURECOORDSYS>
    <!—reference your coordinate reference database here if applicable -
->
   </FEATURECOORDSYS>
   <!-- FILTERCOORDSYS: The current coordinate system of the requesting
    client. For example, it corresponds to the EPSG code -->
   <FILTERCOORDSYS>
   <xsl:attribute name="id">
   <xsl:value-of select="substring-after(/BoundingBox/, '#')" /> <!--
this XSL
    select statement grabs the “BoundingBox” coordinates from the WMS
GetMap
    request and inserts them into the “Filtercoordsys” tag of the ArcXML
    GetImage request -->
   </xsl:attribute>
   </FILTERCOORDSYS>
<!------- WMS BoundingBox gml:coord --------->
<!-- the following portion uses XSL select function to grab “your
attribute” values from the minx, miny, maxx, maxy from the WMS GetMap
request GML coordinates and inserts them into the “Envelope” tag of the
```

```
ArcXML GetImage request -->
<ENVELOPE>
 <xsl:attribute name="minx">
  <xsl:value-of select="/your_attribute"/><xsl:text> </xsl:text>
 </xsl:attribute>
 <xsl:attribute name="miny">
  <xsl:value-of select="/your_attribute "/><xsl:text> </xsl:text>
 </xsl:attribute>
 <xsl:attribute name="maxx">
  <xsl:value-of select="/your_attribute "/><xsl:text> </xsl:text>
 </xsl:attribute>
 <xsl:attribute name="maxy">
  <xsl:value-of select="/your_attribute /><xsl:text> </xsl:text>
 </xsl:attribute>
</ENVELOPE>
<!--------- WMS WIDTH & HEIGHT ----------->
    <IMAGESIZE>
 <!—example: <IMAGESIZE width="500" height="500"/ -->
 <xsl:attribute name="width">
  <xsl:value-of select="/Output/Size/WIDTH"/> <!-- this XSL select
  statement grabs the “Width” value from the WMS GetMap request located
  within ‘/Output/Size’ portion of the GetMap request -->
 </xsl:attribute>
 <xsl:attribute name="height">
  <xsl:value-of select="/Output/Size/HEIGHT"/>
 </xsl:attribute>
 </IMAGESIZE>
<!----------- WMS NamedLayer and Name ---------------->
<StyledLayerDescriptor version="0.7.2">
    <NamedLayer>
     <Name>layer_name</Name>
     <NamedStyle>
      <Name>style_name</Name>
     </NamedStyle>
    </NamedLayer>
   </StyledLayerDescriptor>
 <LAYERLIST>
    <xsl:for-each select="/OWS/WMS/GetMap/LAYERS/LAYER">
    <!-- don't(?) iterate through the nodes in REVERSE order -->
    <xsl:sort select="position()" data-type="number" order="descending"
/>
    <!-- XSL sort selection function orders the Layers for appropriate
overlay
    display results… -->
    <LAYERDEF visible="true" id="{@NAME}" />
      </xsl:for-each>
   <LAYERDEF id="scalebar" visible="true" /> <!-- first layer specified
here, is
    the bottom layer of the output image map -->
   <LAYERDEF id="label_name" visible="true" />
   <LAYERDEF id="STATE" visible="true" />
   <LAYERDEF id="COUNTY" visible="true" />
   <LAYERDEF id="CITY" visible="true" />
   <LAYERDEF id="ZIPCODE" visible="true" />
   <LAYERDEF id="TRACT" visible="true" />
   <LAYERDEF id="BLOCKGROUP" visible="true" />
   <LAYERDEF id="UserStyle" visible="true" />
  </LAYERLIST>
<!-------- WMS FORMAT -------------->
  <OUTPUT>
 <xsl:if test="string(/output/FORMAT)='image/png'">
  <xsl:attribute name="type">PNG</xsl:attribute>
 </xsl:if>
 <xsl:if test="string(/output/FORMAT)='image/jpeg'">
  <xsl:attribute name="type">JPG</xsl:attribute>
 </xsl:if>
 <xsl:if test="string(/output/FORMAT)='image/gif'">
  <xsl:attribute name="type">GIF</xsl:attribute>
 </xsl:if>
   </OUTPUT>
<!--------- TRANSPARENT & BGCOLOR ------------>
<!-- NOTE: Transparency isn't supported in jpegs. To make a color
transparent in a GIF or PNG image, both the color and transcolor
attributes of BACKGROUND must be set to the same color -->
   <BACKGROUND color="255, 255, 255"> <!-- static value -->
    <xsl:if test="string(/OWS/WMS/GetMap/TRANSPARENT)='TRUE'">
    <!-- XSL if-test function to test if the transcolor value specified
is true… -->
      <xsl:attribute
name="transcolor"><xsl:text>"255,255,255"</xsl:text>
     </xsl:attribute>
     </xsl:if>
   </BACKGROUND>
</PROPERTIES>
<!-- acetate layer parameters for label: -->
 <LAYER type="acetate" name="label_name" id="acetate">
 <OBJECT units="pixel">
  <TEXT>
   <xsl:attribute name="coords">
    <xsl:text>1 10</xsl:text>
   </xsl:attribute>
   <xsl:attribute name="label">
   <xsl:value-of select="…/@label"/>
   <xsl:text>label_name</xsl:text>
   </xsl:attribute>
   <TEXTMARKERSYMBOL/>
  </TEXT>
  </OBJECT>
```

```
    </LAYER>
   <!-- acetate layer for UserStyle colors -->
    <LAYER>
    <DATASET/>
    <VALUEMAPRENDERER lookupfield="WIDOWED">
    <VALUEMAPRENDERER>
        <xsl:attribute name="lookupfield">
        <xsl:value-of select="…/@field"/>
        </xsl:attribute>
       <!-- this info will come from SLD/User styles -->
       <!-- values specified by user inside the SqlExpression -->
       <!-- use EXACT if SqlExpression has an "=" sign" -->
       <EXACT value="132; 9; 25; 28; 79; 67; 205">
       <!-- use RANGE if SqlExpression has an "<" or ">" sign -->
       <!-- <RANGE lower="0" upper="300" label="Less than 300" -->
       <!-- shading the feature layer polygons that satisfy the
   SqlExpression query -->
        <SIMPLEPOLYGONSYMBOL fillcolor="88,100,27" filltype="solid"
         filltransparency="1" " />
       <!-- OR you can place a mark, “star,” on the selected feature layer
   polygons -->
        <SIMPLEMARKERSYMBOL color="27,0,227" width="15" type="star" />
       </EXACT>
       </VALUEMAPRENDERER>
      </LAYER>
     </GET_IMAGE>
    </REQUEST>
   </ARCXML>
   </xsl:template>
```

</xsl:transform>

2.5.2. DTD & XML Example 1: Understanding and Using Styles

The map layout is defined in the configuration file `deegree_styles.xml`. The base of the syntax of the layout instructions in **deegree** is the SLD Specification.

degree `NamedStyles` are layer oriented. A bundle of rules will be defined for each layer that will control the rendering of the features of this layer. Each <NamedStyleCollection> is identified by a name.

```
<!ELEMENT NamedStyleCollection ((NamedStyle)*) >
<!ATTLIST NamedStyleCollection name CDATA #REQUIRED >
```

There is no fixed linkage between the `<NamedStyleCollection>` and a layer (feature collection). But it could be that the definition of a `NamendStyledCollection` is so special, that - in practice - its usage is limited to one layer.

Each `NamedStyleCollection` has a collection of `NamedStyles`. `<NamedStyleCollection>` and `NamedStyle` are identifiable by name. Each `NamedStyle` can be defined dependent on the scale for realizing a zoom-level-based layout instruction.

```
<!ELEMENT NamedStyle ((%Symbols;)+) >
<!ATTLIST NamedStyle name CDATA #REQUIRED >
<!ATTLIST NamedStyle minscale CDATA #IMPLIED >
<!ATTLIST NamedStyle maxscale CDATA #IMPLIED >
```

Styles are built by the definition of symbols that are the code for the layout instructions. A `NamedStyle` can be built by several symbols to construct complex graphic elements. Symbols are defined for all basic feature types.

```
<!ENTITY % Symbols "( PolygonSymbol | LineStringSymbol | PointSymbol |
TextSymbol | ScaledPolygonSymbol | ScaledLineStringSymbol |
ScaledPointSymbol | ScaledTextSymbol )" >
```

In addition to the four basic types to describe points, lines, polygons and text, which are defined in the SLD, there are four further entities defined. These entities make it possible for features (featuretypes) to depend on the value of one of its properties. For example it is possible that the shading of states (polygons) could depend on the population density.

2.5.3. DTD & XML Example 2: Basic Symbols

Prerequisite: First, review Chapter 1, section 1.6 of if you are not familiar with XML and DTDs.

Note: This example is a follow-up to Example 1.

The description of the `<LineStringSymbol>` corresponds to the SLD. The definitions of `<PointSymbol>`, `<PolygonSymbol>` and `<TextSymbol>` differ from those mentioned in the OGC SLD specification.

```
<!ELEMENT PolygonSymbol (Geometry, FillColor?, BackgroundColor?,
FillOpacity?, BackgroundOpacity?, FillDashMatrix?, StrokeColor?,
StrokeOpacity?, StrokeWidth?, StrokeDashArray? ) >

<!ELEMENT LineStringSymbol (Geometry?, StrokeColor?, StrokeOpacity?,
StrokeWidth?, StrokeLineJoin?, StrokeLineCap?, StrokeDashArray? ) >

<!ELEMENT PointSymbol ( Geometry?, FillColor?, FillOpacity?,
StrokeColor?, StrokeOpacity?, StrokeWidth?, Mark?, MarkScale?,
MarkOrientation? )>

<!ELEMENT TextSymbol ( Geometry?, Label?, TextPosition?, FontFamily?,
FillColor?, FillOpacity?, FontSize?, FontStyle?, StrokeColor?,
StrokeOpacity? )>
```

Each symbol has an element named Geometry. This defines the geometry linked with the portrayal regulation. In contrast to the SLD specification, the definition was extended so that the hierarchies of geometry, feature and feature type, are possible.

```
<!ELEMENT Geometry
(FetchFeatureType,FetchFeature?,FetchFeatureProperty?) >
<!ELEMENT FetchFeatureType EMPTY>
<!ATTLIST FetchFeatureType name CDATA #REQUIRED>
<!ELEMENT FetchFeature EMPTY>
<!ATTLIST FetchFeature name CDATA #REQUIRED>
<!ELEMENT FetchFeatureProperty EMPTY>
```

```
<!ATTLIST FetchFeatureProperty name CDATA #REQUIRED>
```

The declaration of the featuretype is binding. All features with feature types according to the declaration in the tag `<FetchFeatureType>` will be rendered by one layout instruction, if there is no geometry (`FetchFeatureProperty`) or `feature-Id` (`FetchFeature`) declared. In case of declaring only featuretype and feature, each feature property of the type "geometry" will be rendered by the declaring layout instruction.

Apart from an extension of the `<Mark>` element for the `PointSymbol` definition, an additional convention not manageable in XML is introduced. The item '`Mark`' specifies the symbol (e.g. a circle, a square etc.), which is used for the drawing. Apart from the description via the names of the symbol, one agrees that a positive number represents the index of a character in the unicode format. I.e., `<Mark>square</Mark>` leads to the drawing of a square, `<Mark>82</Mark>` leads to the drawing of the letter `d`.

```
<!ELEMENT Mark (#PCDATA | Symbolfile )>
<!ELEMENT Symbolfile EMPTY>
<!ATTLIST Symbolfile filename CDATA #REQUIRED>
```

Additionally, the filling colour and transparency of the filling of a `FillDashMatrix` is defined, which describes the filling pattern of a polygon. Thus, the definition of a background colour and their transparency also becomes necessary.

```
<!ELEMENT PolygonSymbol (Geometry?, FillColor?, BackgroundColor?,
FillOpacity?, BackgroundOpacity?, FillDashMatrix?, StrokeColor?,
StrokeOpacity?, StrokeWidth?, StrokeDashArray? ) >
```

The following shows that the layout instruction code of a `PolygonSymbol-Definition` defines a polygon with a fill pattern that runs diagonal from the left upper edge to the opposite edge. The value ``1´´ is associated with the foreground color, the value ``0´´ is associated with the background color. In this example, the background color is transparent.

```
 <PolygonSymbol>
 <FillColor>#ff0000</FillColor>
 <BackgroundColor>#ff0000</BackgroundColor>
 <FillOpacity>0.5</FillOpacity>
 <BackgroundOpacity>0.0</BackgroundOpacity>
 <FillDashMatrix>
  (1,1,0,0,1,1,0,0)
  (0,1,1,0,0,1,1,0)
  (0,0,1,1,0,0,1,1)
  (1,0,0,1,1,0,0,1)
 </FillDashMatrix>
 <StrokeDashArray>1,0,0,1,0,0,1,</StrokeDashArray>
</PolygonSymbol>
```

The definition of the `TextSymbol` and the `PolygonSymbol` are compared to the definition in the SLD enhanced. The definition of the `TextSymbol` in **deegree** is different from the SLD.

```
<!ELEMENT TextSymbol (Geometry?, Label?, TextPosition?, FontFamily?,
FillColor?, FillOpacity?, FontSize?, FontStyle?, StrokeColor?,
StrokeOpacity? )>
```

Additionally, the element `TextPosition` is defined. It describes the label point position in a single geometry object.

```
<!ELEMENT TextPosition (CENTER | LEFT_CENTER | RIGHT_CENTER | TOP_CENTER
| BOTTOM_CENTER | LEFT_TOP | RIGHT_TOP | LEFT_BOTTOM | RIGHT_BOTTOM |
Userdefined )>
<!ELEMENT Userdefined EMPTY>
<!ATTLIST Userdefined dX CDATA #REQUIRED >
<!ATTLIST Userdefined dY CDATA #REQUIRED >
```

It is now possible to position the text flexibly in relation to the attached geometry, beside the nine fixed positions for text.

Fundamental modifications are made within the possibilities of color selection. The text color will be controlled by the `<StrokeColor>` element in contrast to SLD, where the `<FillColor>` element is the controlling element. The `<FillColor>` element describes the background color of the text. The value ``0´´ for the `<FillOpacity>` will have the same result as the visualization by the SLD.

2.5.4. DTD & XML Example 3: Scaleable Symbols

Prerequisite: First, review Chapter 1, section 1.6 if you are not familiar with XML and DTDs.

Note: This example is a follow-up to Example 1 and Example 3.

Deegree enhances the definition of layout instructions and scaleable symbols, as described earlier. The structure for all types of symbols is the same, so that in this context only the `<ScaledPointPolygon>` definition will be given as an example. A `<ScaledXXXXSymbol>` contains three elements.

```
<!ELEMENT ScaledPolygonSymbol
(Geometry?, ScaleAttribute?, LimitedSymbol*) >
```

The geometry entry is identical to the one of simple symbols. The element `ScaleAttribute` has the same structure as the geometry element, but specifies the property, which is the base of the value dependent selection of the layout instruction.

```
<!ELEMENT ScaleAttribute (#PCDATA | Geometry )*>
```

The value of the element `<ScaleAttribute>` is determined by the minimum and maximum attributes of the `<LimitedSymbol>` element. It will be compared to the `<ScaledXXXXSymbols>` for determining the layout instruction.

A `<ScaledXXXXSymbol>` element can contain various LimitedSymbol element, which are built by simple symbols.

```
<!ELEMENT LimitedSymbol (PointSymbol | LinestringSymbol | PolygonSymbol
| TextSymbol)>
<!ATTLIST LimitedSymbol min CDATA #REQUIRED>
<!ATTLIST LimitedSymbol max CDATA #REQUIRED>
```

As pointed out, a `<LimitedSymbol>` also contains two attributes determining the range of validity of the contained portrayal rule. A valid `<ScaledPolygonSymbol>` has the following structure:

```
<ScaledPolygonSymbol>
<!—- the geometry to render-->
<Geometry><FetchFeatureType name="Staat"/>
<FetchFeatureProperty name="geometry"/>
</Geometry>
```

```
    <!—- the property, that value determines the
       layout instructions -->
    <ScaleAttribute>
     <Geometry>
      <FetchFeatureType name="Staat"/>
      <FetchFeatureProperty name="Einwohnerdichte"/>
     </Geometry>
    </ScaleAttribute>
   <-- style definitions for attribute value based rendering -->
    <LimitedSymbol min=”0” max=”500”>
     <PolygonSymbol>
      <Geometry>
       <FetchFeatureType name="Staat"/>
       <FetchFeatureProperty name="geometry"/>
      </Geometry>
      <FillColor>#000000</FillColor>
     </PolygonSymbol>
    </LimitedSymbol>
    <LimitedSymbol min=”500” max=”1000”>
     <PolygonSymbol>
      <Geometry>
       <FetchFeatureType name="Staat"/>
       <FetchFeatureProperty name="geometry"/>
      </Geometry>
      <FillColor>#333333</FillColor>
     </PolygonSymbol>
    </LimitedSymbol>
    <LimitedSymbol min=”3000” max=”100000”>
     <PolygonSymbol>
      <Geometry>
       <FetchFeatureType name="Staat"/>
       <FetchFeatureProperty name="geometry"/>
      </Geometry>
      <FillColor>#ffffff</FillColor>
     </PolygonSymbol>
    </LimitedSymbol>
   </ScaledPolygonSymbol>
```

The example describes a part of an XML document for building a layout instruction to render polygons (in this case: countries). The fill color of the polygons depends on the population density of the country. Those with a low population density (0-500) will have a black color, those with a high population density (> 3000) will be white on the map.

2.5.5. DTD & XML Example 4: Complex Types

Prerequisite: First, review Chapter 1, section 1.6 if you are not familiar with XML and DTDs.

Note: This example is a follow-up to Example 2 and Example 3.

In principle, it is possible with the DTD given in the SLD specification to implement complex (i.e. from different symbols) assembled symbols. This only becomes possible, however, by the use of a large number of layers. The situation will be more complicated if the features of a feature type should use a common basic symbol with non-standard symbols for each feature. It is, for example, conceivable that all buses of the line 555 are to be represented by a red a square, and those to the line 666 by a blue square. Additionally, each bus is to be represented by a non-standard symbol that is to be drawn into the bus line identifying square. If, for each bus line, 10 buses are driving, then a map with all buses could be implemented when using SLD only by the use of at least 12 layers (without background). If two buses, even if belonging to different lines, are not allowed to share the same non-standard symbol, the number rises to 22 layers. In the case of use of the portrayal regulations presented here, such a map (without background) can be implemented with only one layer.

```
<NamedStyle name="BusLine 555">
 <PointSymbol>
  <Geometry>
   <FetchFeatureType name="BusLine555"/>
  </Geometry>
  <FillColor>#ffffff</FillColor>
  <StrokeColor>#111111</StrokeColor>
  <StrokeWidth>1.0</StrokeWidth>
  <Mark>square</Mark>
  <MarkScale>17.0</MarkScale>
 </PointSymbol>
</NamedStyle>

<NamedStyle name="Bus1">
<PointSymbol>
  <Geometry>
   <FetchFeatureType name="BusLine555"/>
   <FetchFeature name="Bus1"/>
  </Geometry>
  <FillColor>#ff0000</FillColor>
  <StrokeColor>#111111</StrokeColor>
  <StrokeWidth>1.0</StrokeWidth>
  <Mark>circle</Mark>


  <MarkScale>15.0</MarkScale>
 </PointSymbol>
</NamedStyle>

<NamedStyle name="Bus2">
 <PointSymbol>
  <Geometry>
   <FetchFeatureType name="BusLine555"/>
   <FetchFeature name="Bus2"/>
   </Geometry>
  <FillColor>#0000ff</FillColor>
 </PointSymbol>
</NamedStyle>
```

```
<NamedStyle name="Bus10">
 <PointSymbol>
  <Geometry>
<FetchFeatureType name="BusLine555"/>
   <FetchFeature name="Bus10"/>
   </Geometry>
  <FillColor>#00ff00</FillColor>
 </PointSymbol>
</NamedStyle>
```

The example above shows the layout instruction for a map of bus lines in a **deegree** style XML document. The first `<NamedStyle>` tag defines that all features (buses) that belong to the feature type "bus line 555" will be visualized by a white square with a black outline. The following `<NamedStyle>` tag additionally determines for each bus an individual layout instruction. The buses are characterized by an individually colored circle within the white square of the bus line 555.

All items of the map, i.e., in the case of the example all buses, remain individually addressable. If one implements, with **deegree**, a WFS server with thick-client or a desktop GIS application instead of the picture case of a WMS Servers, all items of each layer can be temporarily faded in/out individually or in groups, or can be modified in their graphical representation.[MM2]

A complex layer with different feature types and several layout instructions for a feature, is one option to control thematic coherent information in one common layer. This does not exclude the possibility of applying one feature in several layers, containing only one layout instruction. The possibility of partitioning in several layers and the redundant allocation of features to several layers offers even more options for graphical effects. The pleasant visualization of lines, which have an outline and cross each other with other lines (streets), is only possible by using two or more layers.

```
<NamedStyle name="BusLine 555">
 <PointSymbol>
  <Geometry>
   <FetchFeatureType name="BusLine555"/>
  </Geometry>
  <FillColor>#ffffff</FillColor>
  <StrokeColor>#111111</StrokeColor>
  <StrokeWidth>1.0</StrokeWidth>
  <Mark>square</Mark>
  <MarkScale>17.0</MarkScale>
 </PointSymbol>
</NamedStyle>

<NamedStyle name="Bus1">
<PointSymbol>
  <Geometry>
   <FetchFeatureType name="BusLine555"/>
   <FetchFeature name="Bus1"/>
  </Geometry>
  <FillColor>#ff0000</FillColor>
  <StrokeColor>#111111</StrokeColor>
  <StrokeWidth>1.0</StrokeWidth>
  <Mark>circle</Mark>
```

```
     <MarkScale>15.0</MarkScale>
    </PointSymbol>
   </NamedStyle>

   <NamedStyle name="Bus2">
    <PointSymbol>
     <Geometry>
      <FetchFeatureType name="BusLine555"/>
      <FetchFeature name="Bus2"/>
      </Geometry>
     <FillColor>#0000ff</FillColor>
    </PointSymbol>
   </NamedStyle>
   <NamedStyle name="Bus10">
    <PointSymbol>
     <Geometry>
   <FetchFeatureType name="BusLine555"/>
      <FetchFeature name="Bus10"/>
      </Geometry>
     <FillColor>#00ff00</FillColor>
    </PointSymbol>
   </NamedStyle>
```

2.5.6. DTD & XML Example 5: Create and Use Legends

Prerequisite: First, review Chapter 1, section 1.6 if you are not familiar with XML and DTDs.

A new function of the WMS 1.1.0 specification, regarding the SLD specification is the possibility of creating legends assigned to a map created by a WMS. At the moment, **deegree** does not support the WMS 1.1.0 nor the SLD specifications completely. Nonetheless, it is possible to create map legends in the way defined there.

Characteristic for the legend generation based on the SLD, is that the WMS client must send a request to the WMS server for each symbol of a legend. This offers good opportunities for designing the legend and the order of the symbols, but in the case of maps with several layers the network traffic increases.

A legend symbol will be requested to the server as following:

```
http://…/deegree/legend?VERSION=1.1.0&REQUEST=GetLegendGraphic&LAYER=wor
ld&STYLE=default&WIDTH=16&HEIGHT=16&FORMAT=gif
```

The example is demanding the default symbol for the layer '`world`'. **deegree** needs the following configuration to answer that request.

First the servlet, which accepts the request from the Tomcat or another servlet engine, must be registered. The registration will be similar to the one from the WMS servlet, resp. in the same `WEB-INF.xml` file. The example shows that the legend servlet is available by the name '`legend`'. The accompanying Parameter references the configuration file (`configuration.xml`) of the WMS.

```
<servlet>
```

```
  <servlet-name>legend</servlet-name>
  <servlet-class>
   org.deegree_impl.enterprise.WMSLegendGraphicServlet
  </servlet-class>
  <init-param>
   <param-name>configFile</param-name>
   <param-value>
    $deegree_home/xml/configuration.xml
   </param-value>
  </init-param>
  <load-on-startup>
   2
  </load-on-startup>
 </servlet>
```

Next, the configuration file of the WMS is supplemented by the line `<LegendService>`. Basically, **deegree** distinguishes between legend symbols that are statically stored and loadable from a file and legend symbols that are generated dynamically with the help of layout instructions, that are defined in the styles document (for more, see section "Controlling the layout instruction by the **deegree** Layer Descriptor interface").

Due to the fact that each of the two options are realized by their own class, the information about which style (legend symbol) will be offered from which class is essential.

```
<LegendService>
 <Class
name="org.deegree_impl.services.wms.legendservice.StaticLegendService_Im
pl">
   <Style name="WorldBorder"/>
   <Style name="world"/>
   <Style name="default"/>
   <Config>
    file:///$deegree_home/webspace/legend/legend_configuration.xml
   </Config>
 </Class>
 <Class
name="org.deegree_impl.services.wms.legendservice.DynamicLegendService_I
mpl">
   <Style name="citystyle"/>
   <Style name="BD_rheurdt_region"/>
   <Style name="BD_schwalmtal_region"/>
   <Style name="tk25s_region"/>
   <Style name="Umgebung_rheurdt_region"/>
   <Style name="Umgebung_schwalmtal_region"/>
   <Style name="Fundort_schwalmtal_font_point"/>
   <Style name="Umgebung_schwalmtal_font_point"/>
   <Config>
    file:///$deegree_home/xml/agit/agit_styles.xml
   </Config>
 </Class>
</LegendService>
```

The default style of each geometry object (point, line, polygon) and image symbol are stored statically. All other symbols/styles can be generated dynamically from the style-document.

Each of the editing classes (`DynamicLegendService_Impl`, `StaticLegendService_Impl`) are configured by a special file.

In the case of the `DynamicLegendService`, this is the styles - document that is used by the WMS. The `StaticLegendService` provides a configuration file that controls the mapping between the styles and the corresponding image files that visualize the legend symbols.

```
<WMT_MS_Capabilities>
 <Capability>
  <Layer>
   <Title>world</Title>
   <Style>
    <Name>default</Name>
    <Title>world</Title>
    <LegendURL width="22" height="20">
     <Format>gif</Format>
     <OnlineResource>
      http://localhost:8080/deegree/legend/world.gif
     </OnlineResource>
    </LegendURL>
   </Style>
  </Layer>
  <Layer>
<Title>WorldBorder</Title>
   <Style>
    <Name>default</Name>
    <Title>WorldBorder</Title>
    <LegendURL width="22" height="20">
     <Format>gif</Format>
     <OnlineResource>
      http://localhost:8080/deegree/legend/WorldBorder.gif
     </OnlineResource>
    </LegendURL>
   </Style>
  </Layer>
 </Capability>
</WMT_MS_Capabilities>
```

<Title> covers the name of a layer and <Name> describes the associated style. The request of a legend symbol for the layer `"world"` with style `"default"` will be like following: `http://localhost:8080/degree/legend/world.gif`.

3. WMS Recipes

This chapter addresses the conversion of existing software into a WMS server or WMS client. It includes contributions from organizations (i.e., vendors, universities) with their specific step-by-step instructions on implementing and using their software.

LIST OF IMPLEMENTATION RECIPES			
Recipe	**Description**	**Contributor**	**Pg. No.**
1	*ESRI OGC WMS Connector for ArcIMS*	*ESRI*	88
2	*deegree Web Map Server*	The WirelessInfo Project	93
3a	*UMN MapServer HOWTO for Setting Up a WMS Server with Mapserver*	*DM Solutions*	99
3b	*UMN MapServer HOWTO for Setting Up a WMS Client with Mapserver*	*DM Solutions*	105
4	*Intergraph GeoMedia WebMap WMS Adapter Kit*	*Intergraph*	108
5	*How to build a WMS from free parts*	*International Interfaces*	116
6	*GSN 3D Client*	*York University GeoICT Lab*	120

Table 7: List of Implementation Recipes

There are several possible ways to implement a WMS server and WMS client. These include purchase of WMS software from a vendor, using an available open source package, or developing your own. The purpose of this chapter is to illustrate as many of these ways as possible.

Products implementing the WMS interface have begun to meet the pent-up demand among enterprise-level user organizations who want to decentralize the maintenance of map data and decentralize the generation of viewable maps. There exist a number of GIS servers implementing the WMS Specification. For a detailed list see OGC product page[60].

Note: Some of the recipes only address the steps needed to install and deploy a WMS server or WMS client. Preliminary steps such as the installation and configuration of a Web server or servlet (i.e., Tomcat) are generally referenced to the corresponding documentation.

Note: The level of detail of the recipes is based on what the corresponding organization provided as the contribution to this Cookbook. Some of the contributions however, have been elaborated and expanded upon.

[60] *OGC WMS Compliant Product List: http://www.opengis.org/testing/product/indedx.php*

3.1. Recipe 1: The ArcIMS OGC WMS Connector

The ArcIMS OGC WMS Connector enables the ArcIMS GIS server to provide Web Map Services that adhere to the OGC WMS interfaces implementation specification. This connector allows any OGC WMS compliant "Viewer Client" (any WMS compatible Web browser or client) the ability to access an ArcIMS MapService. MapServices reside on the ArcIMS Spatial Server, which is called the Map Server by the OGC.

The ArcIMS OGC WMS Connector:

- Produces maps of georeferenced data (maps are rendered in an image format such as PNG, GIF, or JPEG).
- Creates a standard means for users to request maps on the Web using the OGC WMS interface. The request in converted into an ArcXML format, the XML communication protocol for ArcIMS. In a response, the ArcXML response is converted back to a format understood by an OGC WMS View Client.
- Creates a standard means for servers to describe data holdings by the use of the OGC WMS interface Get_Capabilities parameter.

Note: the OGC WMS Connector is separate from and not an upgrade to the WMS connector that is included with ArcIMS 3.2. ESRI is continuously improving the OGC WMS connectors. Check the ESRI Interoperability and Standards page for the latest updates. However, the WMS connector that comes with ArcIMS 4 is compliant with the earlier OGC spec (WMS 1.0.0).

3.1.1. Step 1: Preliminary Requirements

The following preliminary requirements are needed to run the ESRI OGC WMS Connector:

- ArcIMS should be installed, configured and running. To do so, follow the detailed instructions in the ArcIMS Install Guide[61] or see below for main steps.
- Install Apache 1.3.26 with Tomcat 3.2.3 and ArcIMS 4.0 (on Windows). ESRI's support page includes the necessary instructions[62]. The online-version of the installation documents included with ArcIMS 4.0 can be found on ArcOnline[63].
- Download the OGC WMS connector from ESRI Interoperability and Standards page (ArcIMS 4 comes with the Connector already)

NOTE: The User Experience section in Chapter 2 from CSP Ploenzke (Example 1) points out some additional recommendations regarding ArcIMS.

[61] *ArcIMS Install Guide: http://arconline.esri.com/arconline/installation.cfm?PID=6*
[62] *Installation of Apache and Tomcat: http://support.esri.com/Search/kbDocument.asp?dbid=23232*
[63] *ArcOnline: http://arconline.esri.com/installation.cfm?PID=6*

3.1.2. Step 2: Installation of the ESRI OGC WMS Connector

Note: This step is ONLY necessary if your ArcIMS version is earlier than version 4. If you have ArcIMS version 4, you can skip this step.

ArcIMS handles WMS requests through the so-called "ESRI WMS Connector for ArcIMS". Versions before ArcIMS 4 had to install this connector separately. In ArcIMS 4, the WMS Connector is built-in to the required Servlet Connector, meaning you do not to have to install it separately. In either case, you still have to configure it to use it properly on your system.

Unzip the donwloaded zip file to c:\esriwms (the zip file contains 5 directories)

- classes
- config
- docs
- jars
- test_files

Edit C:\esriwms\config\ogc_wms.properties

- Change host=HOSTNAME, and replace HOSTNAME with your website domain name or ip address.
- Change servicename=world, world will be used as default map service for wms connector.
- If you prefer to use other map service as default, change "world" to other map service name.
- In the MapService AXL file setup Default Projection Coord System ID: e. g.(Geographic Coordinates)

```
<PROPERTIES>
 <FEATURECOORDSYS id="4326" />
 <FILTERCOORDSYS id="4326" />
 ....
</PROPERTIES>
```

- Setup WMS Map Service Default Projection Coord System ID based on dataset projection:

```
# Geographic Coordinates ID=4326;
# Robinson ID=54030;
# Sinusoidal ID=54008;
e.g.
wms_filtercoordsys_ID=4326
```

- Edit c:\esriwms\config\wms_capabilities_100.xsl

```
c:\esriwms\config\wms_capabilities_110.xsl
```

- Change "http://www.esri.com", and replace "www.esri.com" with your website domain name or IP address.

- Copy folder to appropriate location:
- Copy content of classes directory (i.e. com folder) to the servlet directory.
  1. (for iplanet, it will be C:\Netscape\docs\servlet)
  2. (for servletexec , it will be C:\Program Files\NewAtlanta\ServletExec ISAPI\Servlets)
- Copy content of config and test_files directories to the website root directory
  1. (for iplanet, it will be C:\Netscape\docs)
  2. (for iis, it will be C:\inetpub\wwwroot)
- Copy content of jars directory to c:\jars
- Add the 4 jar files in c:\jars folder to the jvb classpath or classpath environment variable. If classpath is empty before, it will become:
- CLASSPATH=c:\jars\saxon.jar;c:\jars\xalan.jar;c:\jars\crimson.jar;c:\jars\jaxp.jar;
- Restart Web server. The WMS connector is installed at:
- http://HOSTNAME/servlet/com.esri.ogc.wms.WMSServlet
- where HOSTNAME is the Web server domain name or IP address
- Test templates:
  1. `GetMap` URL: http://HOSTNAME/`GetMap`.htm
  2. `GetCapabilities` URL: http://HOSTNAME/`GetCapabilities`.htm
- where HOSTNAME is the Web server domain name or IP address

3.1.3. Step 3: Configure the WMS Connector

In a text editor, open the file "WMSEsrimap_prop" in the location of your Servlets directory where the Servlet Connector is installed.

For example: <Tomcat Install Dir>\webapps\ROOT\WEB-INF\classes

Change the following parameters:

- Find enable and change it to enable=true.
- Find appServerMachine and set it to the name of the computer where the ArcIMS Application Server is installed.
- Find appServerClientPort and set it to the name of the port on which the Application Server is running (default is 5300).

- Create a capabilities Directory, where you'll later on save `GetCapabilities`-XML documents - you might create "capabilities" under c:/ArcIMS/ as default
- Find capabilitiesDir and provide the path to the capabilities directory you just created, for example, capabilitiesDir=c:/ArcIMS/capabilities/
- Find debug and set it to debug=true if you want detailed debug information. If you set it to true, the log file will be created inside the working directory.
- Create a working Directory, where the log files will be saved - you might create "workingdir" under c:/ArcIMS/ as default
- Find workingDirectory and provide the path to the working directory, for example, workingDirectory=c:/ArcIMS/workingdir/
- Find reaspect and set it to reaspect=true if you want to re-aspect the generated map images.
- Find defaultService and set it to a default ArcIMS MapServiceName as they are listed in ArcIMS Administrator. If the WMS client doesn't specify the Service, then this default Service will be used.
- Save and close the edited file.
- Create a virtual directory called capabilities pointing to the location where you created the capabilities directory.

Using Apache, you will have to open and edit the file "httpd.conf" to set an alias to this directory. Find the section where you have created the aliases for ArcIMS similar to those for the output folder and add the appropriate lines:

```
Alias /capabilities "c:/ArcIMS/capabilities"
   Alias /capabilities/ "c:/ArcIMS/capabilities"
   <Directory "C:/ArcIMS/capabilities">
         Options Indexes MultiViews
         AllowOverride None
         Order allow,deny
         Allow from all
     </Directory>
```

Restart your Web Server and your Servlet Engine. To test if WMS is installed and working, open your Web browser and type:

```
http://<appServerMachine>/servlet/
        com.esri.wms.Esrimap?Cmd=ping
```

The following should appear:

```
   ArcIMS WMS-OGC Connector Version 4.0
```

Your system now knows that there is a WMS Connector and it will handle WMS requests.

```
Alias /wmsviewer
"<ArcIMS_Install_Dir>\ArcIMS\Samples\WMS\DHTML_wmtclient"
Alias /wmsviewer/
"<ArcIMS_Install_Dir>\ArcIMS\Samples\WMS\DHTML_wmtclient"
```

```
<Directory "<ArcIMS_Install_Dir>\ArcIMS\Samples\WMS\DHTML_wmtclient">
         Options Indexes MultiViews
         AllowOverride None
         Order allow,deny
         Allow from all
         </Directory>
```

3.1.4. Step 4: Setup a MapService and Client for your WMS

You might want to use the WMS samples that are part of the ArcIMS samples provided on the ArcIMS installation CD (They can be installed separately from the setup routine):

- Create a MapService called SanFrancisco pointing to the file SanFrancisco.axl, located under:

  `<ArcIMS_Install_Dir>\ArcIMS\Samples\TutorialData\AXL`

  (You might also have to install the Tutorial Data if you did not install it initially)

- Edit the file "WMSEsrimap_prop" located under:

  `<Tomcat_Install_Dir>\webapps\ROOT\WEB-INF\classes`

  and change the parameter defaultService to:

  `defaultService=SanFrancisco`

- Create a virtual directory called "wmsViewer" pointing to the location where you installed the WMS sample:

Using Apache, you will have to open and edit the file "httpd.conf" to set an alias to this directory. Find the section where you have created the aliases for ArcIMS (the output folder) and add the appropriate lines:

```
Alias /wmsviewer
"<ArcIMS_Install_Dir>\ArcIMS\Samples\WMS\DHTML_wmtclient"
Alias /wmsviewer/
"<ArcIMS_Install_Dir>\ArcIMS\Samples\WMS\DHTML_wmtclient"
<Directory
"<ArcIMS_Install_Dir>\ArcIMS\Samples\WMS\DHTML_wmtclient">
         Options Indexes MultiViews
         AllowOverride None
         Order allow,deny
         Allow from all
         </Directory>
```

3.1.5. Step 5: Test your WMS Service

If you have completed the steps above, you should have a MapService called SanFrancisco running which can be viewed and tested through the WMS interface using the ArcIMS WMS sample client.

Open your web browser, type the URL:

```
http://<localhost>/wmsViewer
```

To view the sample. You should see a Map of SanFrancisco and some available tools. You should be able to navigate this map and to get feature information using the identify tool.

You might also want to try to get the results of the `GetCapabilities` request typing the following URL:

```
http://<localhost>/servlet/com.esri.wms.Esrimap?SERVICENAME=SanFrancisco
&VERSION=1.1.0&REQUEST=getcapabilities&
```

3.2. Recipe 2: deegree Web Map Server

deegree is a Java Framework product for the implementation of local and web based GIS applications. Its interfaces and architecture guarantee optimized interoperability due to the compliance with OpenGIS standards. **deegree** is available as an open source project at www.deegree.org (under the terms of the GNU Lesser General Public License since August 2002).

Besides a WMS, **deegree** comprises a WCS and a WFS services.

deegree's WMS server is flexible concerning the possibilities of configuration, the adaptation to different data sources and formats, layouts and server environments. The configuration of the WMS, the Catalog Service and the WFS takes place by customizing different XML files that control the functionality of the **deegree** server.

The web services of **deegree** are realized as Java servlets, in case of the WMS, there is one general WMS servlet and a special servlet for legend graphic. These servlets have to be integrated in the respective web server. Most of the common web servers support servlet technology, so that **deegree** is not limited to special products. The Apache-Tomcat (4.x) Servlet-Engine is recommended due to the widespread of installations and the status as open-source product.

NOTE: Additional contribution on using **deegree** comes from feedback from Professor Stephan Winter and his students from his class on GI Interoperability. See the last section of Recipe 2 for this very useful feedback.

3.2.1. Step 1: Preliminary Requirements

The following preliminary requirements are needed to run the **deegree** WMS:

- Download **deegree** WMS package from **deegree** WMS page[64] appropriate for your platform (LINUX or Windows) and archive extraction. See **deegree**'s online documentation[65] (Section 2.1) for instructions.

- Tomcat integration. Download and install a current release of Tomcat. See **deegree**'s online documentation (Section 2.2) for Tomcat deployment with **deegree** WMS.

3.2.2. Step 2: Installation of deegree WMS

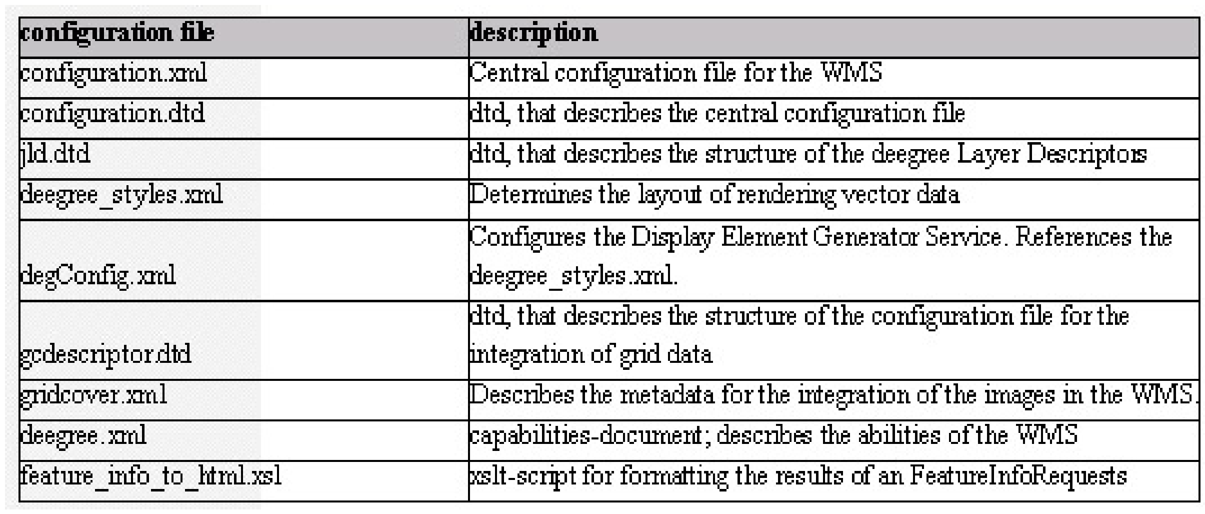
- WMS Configuration

During the initializing process, the name and position of the central configuration file (configuration.xml) will be transferred. The data sources will be registered and some other configuration files will be

[64] deegree WMS page: http://deegree.sourceforge.net
[65] http://www.deegree.org/demo/WMS-Konfiguration_english.htm

referenced directly or indirectly. One of these is the WMS capabilities document (WMS capabilities.dtd), that describes the abilities of the WMS. For a detailed description of the structure of the WMS capabilities document, refer to the WMS Specification.

Table 6, below, lists required configuration files and their associated DTD files supplied with **deegree** WMS. See Appendix A for full definition of these files, and examples.



configuration file	description
configuration.xml	Central configuration file for the WMS
configuration.dtd	dtd, that describes the central configuration file
jld.dtd	dtd, that describes the structure of the deegree Layer Descriptors
deegree_styles.xml	Determines the layout of rendering vector data
degConfig.xml	Configures the Display Element Generator Service. References the deegree_styles.xml.
gcdescriptor.dtd	dtd, that describes the structure of the configuration file for the integration of grid data
gridcover.xml	Describes the metadata for the integration of the images in the WMS.
deegree.xml	capabilities-document; describes the abilities of the WMS
feature_info_to_html.xsl	xslt-script for formatting the results of an FeatureInfoRequests

Table 6: deegree WMS Required Configuration Files and Associated DTDs

Note: In the case of using a database as a geospatial data source, a configuration file is required for describing the database linkage.

In case you did not change (as part of the preliminary requirement step - Tomcat integration) the standard settings of the supplied XML files, you now must adjust the directory settings in configuration.xml and then replace the server names in capabilities.xml with the name or IP number of your server, and then you are finished. Otherwise you have to edit `degConfig.xml` and `gridcover_world.xml` in the xml directory as well.

Note: In the configuration files supplied with the current version of **deegree** WMS paths, you often have the form:

```
http://localhost:8080/deegree/pathname/filename
```

This is an optional way to reference configuration files with URLs. If you are not familiar or comfortable with referencing files in this way, stay with the standard (absolute) syntax supplied in the **deegree's** online documentation.

3.2.2.1. deegree's Internal Service Structure

According to A. Cuthbert's portrayal model, a WMS is divided into four services. The filter service arranges the communication between the data sources and supplies a standard interface providing Features. The Display Element Generator (DEG-Service) takes these and combines them with symbolization instructions and supplies display elements. The render service compiles a map from the display elements using meta-information describing the layout. The map will be visualized by the display service.

The display service is realized by a web browser. The other services will be registered by the WMS with one or several java classes. Particularly in case of using several data sources, the WMS will register one java class for each data source. The structure of the central configuration file will be described in the configuration.dtd.

3.2.2.2. DEG- and Render Service(s) Registration

The configuration.dtd describes the architecture of the central **deegree** WMS configuration file.

```
<!ELEMENT deegreeConfiguration
(Capabilities,FilterService*,DEGService*,RenderService?)>
```

Each `<DEGService>` tag and each `<RenderService>` tag will get a `<Class>` tag, that knows the name of the java class, which supports a service. The name of the class is stored by the attribute name of the `<Class>` tag.

```
<!ELEMENT DEGService (class)>
<!ELEMENT RenderService (class)>
<!ELEMENT Class (Config?,Layer*)>
<!ATTLIST Class name CDATA #REQUIRED>
```

The DEG service uses the `<Config>` tag that is part of the `<Class>`. The `<Config>` tag refers to a configuration file that embodies which layout rules will be realized by the `DEG` service.

```
<?xml version="1.0" encoding="iso-8859-1" standalone="no"?>
<DEGConfiguration>
 <StyleDocument>
  $deegree_home/xml/deegree_styles.xml
 </StyleDocument>
</DEGConfiguration>
```

3.2.2.3. Configuration File for a DEG Service

The `<Layer>` tag is not important for the `DEG` service and the render service at this time. The registration of both services in the WMS could be like the following:

```
<DEGService>
<Class name="org.deegree_impl.services.wms.degservice.DisplayElementGene
rator_Impl">
 <Config>file:///$deegree_home/xml/degConfig.xml</Config>
 </Class>
</DEGService>
<RenderService>
 <Class
name="org.deegree_impl.services.wms.renderservice.Renderer_Impl"/>
</RenderService>
```

The position of the deg-configuration file must be given in the form of an URL address. The java archive degree.jar contains both of the referenced classes. It is possible to register self-defined classes as services. For this purpose, they have to serve the defined logs and interfaces in **deegree**.

3.2.2.3.1. *Testing the Basic Configuration*

To test if the basic configuration was done correctly, re-start Tomcat. At this point, the WMS should already be integrated. The following request should return a map of the world (it is assumed that Tomcat uses port 8080):

```
http://localhost:8080/deegree/deegreewms?WMTVER=1.0.0&
REQUEST=map&SRS=EPSG%3A4326&BBOX=-180,-90,180,90&
WIDTH=641&HEIGHT=481&
LAYERS=world,WorldBorder,WorldCities&
STYLES=default,default,citystyle&FORMAT=jpg&
BGCOLOR=0xfff8ff&TRANSPARENT=FALSE&EXCEPTIONS=INIMAGE
```

If you zoom into the map using the following request, you will see that now the cities are visible. This is because the city-layer is defined (see `configuration.xml`) as valid only for small scales.

```
http://localhost:8080/deegree/deegreewms?WMTVER=1.0.0&
REQUEST=map&SRS=EPSG%3A4326&BBOX=4,47,13,58&WIDTH=641&
HEIGHT=481&LAYERS=world,WorldBorder,WorldCities&
STYLES=default,default,citystyle&FORMAT=jpg&
BGCOLOR=0xfff8ff&TRANSPARENT=FALSE&EXCEPTIONS=INIMAGE
```

3.2.2.3.2. *Add your own data to the WMS*

All geospatial data to be visualized by **deegree** WMS have to be registered in the capabilities.xml and the main configuration file (configuration.xml). While the first one is the capabilities document described by the OGC WMS Specifications, the later one is a **deegree** WMS specific file.

3.2.2.3.3. *Registration of the Filter Service and Data Sources*

For the registration of data sources within the `configuration.xml` file, information about the corresponding filter service is necessary. The `<Filterservice>` tag contains 0..n `<FeatureInfoFormat>` tags.

```
<!ELEMENT FilterService (FeatureInfoFormat*,Class+)>
```

The `<FeatureInfoFormat>` tag provides information about the XSLT script that is responsible for the formats of the results of a `FeatureInfoRequest`. Different XSLT scripts for different return formats can be registered. Default is the MIME (HTML) format.

3.2.2.4. Registration of Image Data as Data Source

The **deegree** WMS provides support for images in the following formats: TIFF-, GIF-, BMP-, JPEG- or PNG-image format. For this intention, the class:

```
org.deegree_impl.services.wms.filterservice.GridCoverageFilter
```

must be registered by the `<Class>` tag in the `<Filterservice>` tags of the central configuration file `(configuration.xml)`. The **deegree**.jar archive contains that class. Further on, the position of the configuration file with the detailed information of the image data that should be displayed, is described in the `<Config>` tag. Finally, the WMS gets information about the names of the layer and the assignment of data source and layer.

```
<!ELEMENT Class (Config?,Layer*)>
<!ELEMENT Layer (KnownDataSource+)>
```

```
<!ATTLIST Layer name CDATA #REQUIRED>
<!ELEMENT KnownDataSource (#PCDATA)>
<!ATTLIST KnownDataSource minscale CDATA #IMPLIED>
<!ATTLIST KnownDataSource maxscale CDATA #IMPLIED>
```

The referenced data sources (`<KnownDataSource>`) are specified by the `<Config>` tag in the XML file. Several data sources can be declared for one layer. Scale dependent visibility may be defined by the attributes `minscale` and `maxscale`. This allows for the usage of different image data sources with an optimized resolution for different zoom levels. The complete process of referencing the three layers that have access to image data, could be similar to the following:

```
<Class
name="org.deegree_impl.services.wms.filterservice.GridCoverageFilter">
<Config>file:///$deegree_home/xml/gridcover.xml</Config>
 <Layer name="sa_hs">  <KnownDataSource>SachsenAnhalt</KnownDataSource>
 </Layer>
 <Layer name="geomis">
  <KnownDataSource>geomis</KnownDataSource>
 </Layer>
 <Layer name="euro">
  <KnownDataSource>euro</KnownDataSource>
 </Layer>
</Class>
```

In the WMS-Request, the layer will be addressed by "`sa_hs`", "`geomis`" and "`euro`". The images that are assigned to the layer are referenced in the file `gridcover.xml`. The structure of `gridcover.xml` is described by `gcdescriptor.dtd`.

The idea of the configuration file for the implementation of images is the definition of layers. Their names will be referenced in the central configuration file (see above).

```
<!ELEMENT GCLayer (Preview?,ScaledTileCollection+,Property+) >
<!ATTLIST GCLayer name CDATA #REQUIRED>
<!ATTLIST GCLayer minx CDATA #REQUIRED>
<!ATTLIST GCLayer miny CDATA #REQUIRED>
<!ATTLIST GCLayer maxx CDATA #REQUIRED>
<!ATTLIST GCLayer maxy CDATA #REQUIRED>
<!ATTLIST GCLayer cs CDATA #REQUIRED>
```

The further attributes of the `<Layer>` tag determine the bounding box and the projection parameters. Each layer contains one or more properties for further descriptions. In the case of declaring only one property, a unique Id of the layer is necessary.

```
<!ELEMENT Property EMPTY>
<!ATTLIST Property name CDATA #REQUIRED>
<!ATTLIST Property value CDATA #REQUIRED>
```

Another important aspect is the possibility to define several so called "`ScaledTileCollections`" within a layer. The `ScaledTileCollections` have information about the association of image files and layers. The assignment can depend on the zoom level. For each zoom level, a `<ScaledTileCollection>` will be created. minscale and maxscale correspond to the length of the diagonal of one pixel from the image center, expressed in units of the used spatial reference system. Similar to the layer, a `ScaledTileCollection` contains properties for further descriptions.

```
<!ELEMENT ScaledTileCollection (Tile+,Property+)>
```

```
<!ATTLIST ScaledTileCollection minscale CDATA #REQUIRED>
<!ATTLIST ScaledTileCollection maxscale CDATA #REQUIRED>
```

Each `ScaledTileCollection` is build by 1 : n image files. This makes it possible to tile large sized maps for optimizing the access of the WMS. Only required tiles will be loaded.

```
<!ELEMENT Tile (#PCDATA)>
<!ATTLIST Tile name CDATA #REQUIRED>
<!ATTLIST Tile minx CDATA #REQUIRED>
<!ATTLIST Tile miny CDATA #REQUIRED>
<!ATTLIST Tile maxx CDATA #REQUIRED>
<!ATTLIST Tile maxy CDATA #REQUIRED>
```

Each tile is described by its name and its bounding box. The coordinate reference system used for the tile-boundingbox must be identical to the one defined for the whole layer.

The `<Preview>` tag of the layers makes it possible to reference an overview map for a quick visualization of large areas.

Example for a layer definition:

```
<GCLayer name="world" minx="-20.9" miny="37" maxx="35.7" maxy="70"
  cs="EPSG:4326">
 <Preview>
  <Tile name="preview"
     minx="-20.9" miny="37" maxx="35.7" maxy="70">
     file:///$deegree_home/data/raster/world_new_3.jpg
  </Tile>
 </Preview>
 <ScaledTileCollection minscale="0" maxscale="40000">
  <Tile name="level1"
     minx="-20.9" miny="37" maxx="35.7" maxy="70">
     file:///$deegree_home/data/raster/level1_4_2.gif
  </Tile>
  <Property name="projekt" value="TestIt"/>
 </ScaledTileCollection>
 <Property name="ID" value="#212"/>
</GCLayer>
```

3.2.2.5. Registration of ESRI-Shape-Files as Data Sources (in deegree)

Similar to the process of reading image data is the registration of a class for connection to ESRI ShapeFiles. In the default setting, the following class is responsible for this:

```
org.degree_impl.services.wms.filterservice.ShapeFilter
```

Subsequently, the definition of layer and data source are necessary. The declaration of an own configuration file is not essential.

```
Class
name="org.deegree_impl.services.wms.filterservice.CachedShapeFilter">
 <Layer name="WorldBorder">
 <KnownDataSource>$deegree_home/data/shape/country</KnownDataSource>
```

```
    </Layer>
    <Layer name="cities">
    <KnownDataSource>$deegree_home/data/shape/worldcity</KnownDataSource>
    </Layer>
   </Class>
```

The two registered layers are accessible via the WMS by the names "`EuroBorder`" and "`heller`". The tag `<KnownDataSource>` describes the positions of the Shape Files (without extension). Note that this will be realized by the declaration of the path of the file and not by the declaration of a URL (no `file:///` in front). Similar to the registration of image data, the definition of the Shape File layer dependent on the zoom level is possible.

Note: For the following databases registration, see **deegree** online documentation, Section 3, "Add your own Data to the WMS."

- Registration of an Oracle-(Spatial) Database as data source
- Registration of standard JDBC capable [jf5]databases for point data as data source
- Registration of other WebMapServers as data source

3.3. Recipe 3: University of Minnesota (UMN) MapServer

MapServer v3.5 implements WMS features. At the time this document was written, Mapserver supports the following WMS versions: 1.0.0, 1.0.7, and 1.1.0 (a.k.a. 1.0.8).

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 define map object) with appropriate metadata in it.

3.3.1. Recipe 3A: UMN Mapserver HOWTO for Setting Up a WMS Server with Mapserver

3.3.1.1. Step 1: Preliminary Requirements

The following preliminary requirements are needed to run a WMS Server using UMN MapServer:

- MapServer should be installed, configured and running. To do so please follow the UMN MapServer online instructions[66].
- If you have a MapServer currently installed, check that your mapserv executable includes WMS support. One way to verify this is to use the "`-v`" command-line switch and look for "`SUPPORTS=WMS`".

On Unix:

```
$ ./mapserv -v
  MapServer version 3.5 (pre-alpha)
```

[66] UMN Map Server homepage:http://mapserver.gis.umn.edu/)

```
    OUTPUT=PNG OUTPUT=JPEG
    OUTPUT=WBMP SUPPORTS=PROJ SUPPORTS=TTF
    SUPPORTS=WMS
    INPUT=EPPL7 INPUT=JPEG INPUT=OGR
    INPUT=GDAL INPUT=SHAPEFILE
```

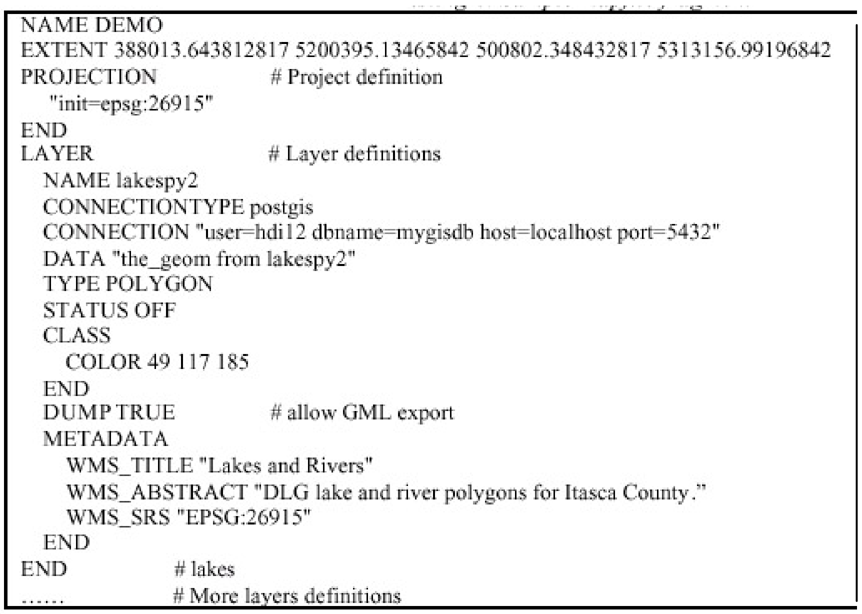
On Windows:

```
  C:\apache\cgi-bin> mapserv -v
    MapServer version 3.5 (pre-alpha)
    OUTPUT=PNG OUTPUT=JPEG OUTPUT=WBMP SUPPORTS=PROJ
    SUPPORTS=TTF SUPPORTS=WMS
    INPUT=EPPL7 INPUT=JPEG INPUT=OGR
    INPUT=GDAL INPUT=SHAPEFILE
```

3.3.1.2. Step 2: Setup a Mapfile For Your WMS

Each instance of a WMS server that you setup needs to have its own mapfile. This is an ordinary MapServer mapfile in which some parameters and some metadata entries are mandatory. Most of the metadata is required in order to produce a valid `GetCapabilites` output.

Figure 28, below, below shows an example fragment of a mapfile:



```
NAME DEMO
EXTENT 388013.643812817 5200395.13465842 500802.348432817 5313156.99196842
PROJECTION              # Project definition
  "init=epsg:26915"
END
LAYER                   # Layer definitions
  NAME lakespy2
  CONNECTIONTYPE postgis
  CONNECTION "user=hdi12 dbname=mygisdb host=localhost port=5432"
  DATA "the_geom from lakespy2"
  TYPE POLYGON
  STATUS OFF
  CLASS
    COLOR 49 117 185
  END
  DUMP TRUE             # allow GML export
  METADATA
    WMS_TITLE "Lakes and Rivers"
    WMS_ABSTRACT "DLG lake and river polygons for Itasca County."
    WMS_SRS "EPSG:26915"
  END
END            # lakes
......         # More layers definitions
```

Figure 28: Sample UMN MapServer Mapfile Fragment

Here is the list of parameters and metadata items that usually are optional with MapServer, but are required (or strongly recommended) for a WMS configuration:

At the map level:

- Map NAME & Map PROJECTION
- Map Metadata (in the WEB Object):

- wms_title
- wms_onlineresource
- wms_srs (unless PROJECTION object is defined using "init=epsg:...")

And for each layer:

- Layer NAME & Layer PROJECTION
- Layer Metadata
- wms_title
- wms_srs (optional since the layers inherit the map's SRS value)

Below, each parameter is discussed in more detail:

- **Map NAME and wms_title:**

WMS Capabilities requires a Name and a Title tag for every layer. The Map's NAME and `wms_title` metadata will be used to set the root layer's name and title in the `GetCapabilities` XML output. The root layer in the WMS context corresponds to the whole mapfile.

- **Layer NAME and wms_title metadata:**

Every individual layer needs its own unique name and title. Layer names are also used in `GetMap` and `GetFeatureInfo` requests to refer to layers that should be included in the map output and in the query.

- **Map PROJECTION and wms_srs metadata:**

WMS servers have to advertise the projection in which they are able to serve data using EPSG projection codes[67]. Recent versions of the PROJ4 library come with a table of EPSG initialization codes and allow users to define a projection like this:

```
PROJECTION
   "init=epsg:4269"
   END
```

The above is sufficient for MapServer to recognize the EPSG code and include it in SRS tags in the capabilities output (`wms_srs` metadata is not required in this case). However, it is often impossible to find an EPSG code to match the projection of your data. In those cases, the "`wms_srs`" metadata is used to list one or more EPSG codes that the data can be served in, and the PROJECTION object contains the real PROJ4 definition of the data's projection.

Here is an example of a server whose data is in a Lambert Conformal Conic projection (for which there is no EPSG code). It's capabilities output will advertise EPSG:4269 and EPSG:4326 projections (lat/lon), but the PROJECTION object is set to the real projection that the data is in:

```
NAME DEMO
  ...
  WEB
  ...
```

[67] *EPSG projection codes: http://inovagis.dcea.fct.unl.pt/giserver/epsg.asp.*

```
    METADATA
     "wms_title"      "WMS Demo Server"
     "wms_onlineresource" "http://my.host.com/cgi-
  bin/mapserv?map=wms.map&"
     "wms_srs"       "EPSG:4269 EPSG:4326"
    END
    END

    PROJECTION
     "proj=lcc"
     "ellps=GRS80"
     "lat_0=49"
     "lon_0=-95"
     "lat_1=49"
     "lat_2=77"
    END
    ...
```

- **Layer PROJECTION and wms_srs metadata:**

By default in the WMS context, layers inherit the SRS or their parent layer (the map's projection in the MapServer case). For this reason, it is not necessary (but still strongly recommended) to provide `PROJECTION` and `wms_srs` for every layer. However, if your server wants to advertise multiple projections, then at least a `PROJECTION` object is required in every layer, otherwise the layers won't be reprojected. This is the way on-the-fly reprojection works in MapServer. Layer `PROJECTION` and `wms_srs` metadata are defined exactly the same way as the map's `PROJECTION` and `wms_srs` metadata.

- **The "wms_onlineresource" metadata:**

The `wms_onlineresource` metadata is set in the map's Web object metadata and specifies the URL that should be used to access your server. This is required for the `GetCapabilities` output. If `wms_onlineresource` is not provided, then MapServer will try to provide a default one using the script name and hostname, but this is not completely reliable. It is strongly recommended that you provide the `wms_onlineresource` metadata.

See the WMS Specification for the whole story about the online resource URL. Basically, what you need is a complete HTTP URL including the `http://` prefix, hostname, script name, potentially a "`map=`" parameter, and terminated by "?" or "&".

Here is a valid online resource URL:

```
http://my.host.com/cgi-bin/mapserv?map=mywms.map&
```

By creating a wrapper script on the server, it is possible to hide the "`map=`" parameter from the URL and then your server's online resource URL could be something like:

```
http://my.host.com/cgi-bin/mywms?
```

This is covered in more detail in Section 3.3.1.4. below, "More About the Online Resource URL."

3.3.1.3. Step 3: Test Your WMS Server

3.3.1.3.1. Validate the Capabilities Metadata

With the mapfile in place, you have to check the XML capabilities returned by your server to make sure nothing is missing. Using a Web browser, access your server's online resource URL to which you add the parameter "REQUEST=`GetCapabilities`" to the end, e.g.:

```
http://my.host.com/cgi-bin/mapserv?map=mywms.map&
REQUEST=GetCapabilities
```

This should return a document of MIME type `application/vnd.ogc.wms_xml`, so your Web browser is likely to prompt you to save the file. Save it and open it in a text editor (Emacs, Notepad, etc.)

If you get an error message in the XML output, then take necessary actions. Common problems and solutions are the following:

Q: How can I find the EPSG code for my data's projection?

A: If you know the parameters of your data's projection, then you can browse the "`epsg`" file that comes with PROJ4 and look for a projection definition that matches your data's projection. It's a simple text file and the EPSG code is inside brackets (`<...>`) at the beginning of every line.

Q: My WMS server produces the error "`msProcessProjection(): no system list, errno: ..`"

A: That's likely PROJ4 complaining that it cannot find the "`epsg`" projection definition file. Make sure you have installed PROJ 4.4.3 or more recent and that the "epsg" file is installed at the right location. On Unix it should be under `/usr/local/share/proj/`, and on Windows PROJ looks for it under `C:\PROJ\`.

If everything went well, you should have a complete XML capabilities document. Search it for the word "WARNING"... MapServer inserts XML comments starting with "`<!--WARNING: `" in the XML output if it detects missing mapfile parameters or metadata items. If you notice any warning in your XML output, then you have to fix all of them before you can register your server with a WMS client, otherwise you will probably run into problems.

3.3.1.3.2. Test With a GetMap Request

Knowing that the server can produce a valid XML `GetCapabilities` response, you can now test the `GetMap` request.

Simply adding "VERSION=1.1.0&REQUEST=`GetMap`" to your server's URL should generate a map with the default map size and all the layers with STATUS ON or DEFAULT displayed, e.g.

```
http://my.host.com/cgi-bin/mapserv?map=mywms.map&VERSION=1.1.0&
REQUEST=GetMap
```

Note: this works with MapServer's WMS interface even if it would be an incomplete `GetMap` request according to the WMS spec. It lacks several mandatory parameters such as SRS, `BBOX`, etc, but it is good enough for testing at this point..

3.3.1.3.3. Test With a Real Web Client

If you have access to a WMS client, then register your new server's online resource with it and you're running. If you don't have your own WMS client, then CubeWerx's WMS interface (cubeview) can be useful to test a new server. Access the following page:

```
http://www.cubewerx.com/demo/cubeview/cubeview.cgi
```

and enter your server's URL at the bottom of the page and click "GO". The default set of layers in the interface should be replaced with your server's layers.

3.3.1.4. More About the Online Resource URL

As mentioned in Section 3.3.1.2. "Setup a Mapfile for Your Web Map Service" above, the following Online Resource URL is perfectly valid for a MapServer WMS according to section 6.2.1 of the WMS 1.1.0 specification:

```
http://my.host.com/cgi-bin/mapserv?map=mywms.map&
```

Some people will argue that the above URL contains mandatory vendor-specific parameters and that this is illegal. First we would like to point out that "`map=...`" is not considered a vendor-specific parameter in this case since it is part of the Online Resource URL which is defined as an opaque string terminated by "?" or "&"

However, even if it's valid, the above URL is still ugly and you might want to use a nicer URL for your WMS Online Resource URL. Here are some suggestions; some of them will work only on Unix, but at least #2 will work for Windows/Apache users as well.

On Unix servers, you can setup a wrapper shell script that sets the `MS_MAPFILE` environment variable and then passes control to the mapserv executable that results in a cleaner `OnlineResource` URL:

```
#! /bin/sh
MS_MAPFILE=/path/to/demo.map
export MS_MAPFILE
/path/to/mapserv
```

Another option is to use the "`setenvif`" feature of Apache: use symbolic links that all point to a same mapserv binary, and then for each symbolic link, test the URL, and set the MAP environment accordingly. For Windows and Apache users, the steps are as follows (this requires Apache 1.3 or newer):

- Copy `mapserv.exe` to a new name for your WMS, such as "`mywms.exe`".
- In `httpd.conf`, add:

```
SetEnvIf Request_URI "/cgi-bin/mywms" MS_MAPFILE=/path/to/mymap.map
```

3.4. Recipe 3B: UMN MapServer HOW TO for Setting Up a WMS Client with Mapserver

3.4.1. Step 1: Compilation / Installation

The WMS connection type is enabled by the `--with-wmsclient` configure switch. It requires PROJ4 and W3C's libwww (libwww v5.3.2 or newer. This is required because there was a bug that caused an infinite loop in older versions), and GDAL is optional (see below).

- For PROJ4 and GDAL installation, see the MapServer Compilation HOWTO: UNIX[68] or Win32[69]
- For W3C's libwww, download v5.3.2 (or newer) from CVS or in a tarball and compile/install manually[70].

You might want to also include GDAL support if you want your application to be able to reproject map slides received from remote servers. This is because raster resampling works only when used with the GDAL library in MapServer. If GDAL is not included in your MapServer build, then your application can only serve maps in the subset of the projections supported by all the remote servers (this should be sufficient for most applications). If you compile with GDAL, then make sure that your GDAL includes GIF and/or PNG support, depending on which image format you request from remote servers.

Once the required libraries are installed, then configure MapServer using the `--with-wms` client switch (plus all the other switches you used to use) and recompile.

This will give you a new set of executables (and possibly `php_mapscript` if you requested it). See the *MapServer Compilation HOWTO* for installation details.

3.4.2. Step 2: Check your MapServer executable

To check that your mapserv executable includes WMS support, use the "`-v`" command-line switch and look for "`SUPPORTS=WMS_CLIENT`".

Example 1. On Unix:

```
$ ./mapserv -v
  MapServer version 3.5 (pre-alpha)
  OUTPUT=PNG OUTPUT=JPEG OUTPUT=WBMP
  SUPPORTS=PROJ SUPPORTS=TTF
  SUPPORTS=WMS_CLIENT INPUT=EPPL7 INPUT=JPEG
  INPUT=OGR INPUT=GDAL INPUT=SHAPEFILE
```

Example 2. On Windows:

```
C:\apache\cgi-bin> mapserv -v
  MapServer version 3.5 (pre-alpha)
  OUTPUT=PNG OUTPUT=JPEG OUTPUT=WBMP
```

[68] *UMN Map Server Compilation HOWTO (for UNIX) http://mapserver.gis.umn.edu/doc/unix-install-howto.html*
[69] *UMN Map Server Compilation HOWTO (for Win32) http://mapserver.gis.umn.edu/doc/win32compile-howto.htm*
[70] *W3C's libwww: http://www.w3.org/Library/*

```
SUPPORTS=PROJ SUPPORTS=TTF SUPPORTS=WMS_CLIENT
INPUT=EPPL7 INPUT=JPEG INPUT=OGR
INPUT=GDAL INPUT=SHAPEFILE
```

Step 3: MapFile configuration - `CONNECTIONTYPE WMS`

A `PROJECTION` must be set in the mapfile for the MAP unless you are sure that all your WMS layers support only a single projection which is the same as the `PROJECTION` of the map. The `MAP PROJECTION` can be set using "`init=epsg:xxxx`" codes or using regular PROJ4 parameters. Failure to set a MAP PROJECTION may result in blank maps coming from remote WMS servers (because of inconsistent BBOX+SRS combination being used in the WMS connection URL).

WMS layers are accessed via the WMS connection type. Here is an example of a layer using this connection type:

```
LAYER
   NAME bathymetry
   METADATA
    "wms_title" "Elevation/Bathymetry"
    "wms_srs"  "EPSG:42304 EPSG:4269 EPSG:4326"
   END
   TYPE RASTER
   STATUS ON
   CONNECTIONTYPE WMS
   CONNECTION "http://www2.dmsolutions.ca:8099/cgi-
bin/mswms_gmap?VERSION=1.1.0&LAYERS=bathymetry&FORMAT=image/png"
   PROJECTION
    "init=epsg:42304"
   END
  END
```

The following items are required for the WMS connection type:

- "`wms_srs`" metadata: a space-delimited list of EPSG projection codes supported by the remote server. You normally get this from the server's capabilities output.
- `CONNECTIONTYPE WMS`: of course!
- `CONNECTION` parameter: This is the remote server's online resource URL to which you append some of the `getMap/getFeatureInfo` request parameters. At this point MapServer sets only the following request parameters:

```
REQUEST

SRS

BBOX

WIDTH

HEIGHT
```

The connection string should contain all other required params, including:

```
VERSION

LAYERS

FORMAT

TRANSPARENT
```

The following layer parameters are optional:

- `PROJECTION` object: This is optional at this point. MapServer will create one internally if needed.

- `MINSCALE, MAXSCALE`: If the remote server's capabilities contain a `ScaleHint` value for this layer, then you might want to set the `MINSCALE` and `MAXSCALE` in the `LAYER` object in the mapfile. This will allow MapServer to request the layer only at scales where it makes sense.

- "`wms_latlonboundingbox`" metadata: The bounding box of this layer in geographic coordinates in the format "`lon_min lat_min lon_max lat_max`". If it is set, then MapServer will request the layer only when the map view overlaps that bounding box. You normally get this from the server's capabilities output.

  e.g.

  ```
     METADATA
     "wms_latlonboundingbox" "-124 48 -123 49"
     END
  ```

- "`wms_connectiontimeout`" metadata: The maximum time to load a remote WMS layer, set in seconds (default is 30 seconds). This metadata can be added at the layer level so that it affects only that layer, or it can be added at the map level (in the web object) so that it affects all of the layers. Note that `wms_connectiontimeout` at the layer level has priority over the map level.

  ```
  ...
    CONNECTIONTYPE WMS
      METADATA
       "wms_srs" "EPSG:4269 EPSG:32182"
       "legend_icon" "Icons/none.gif"
       "wms_title" "NF Water (Lakes) (a)"
       "wms_boundingbox" "EPSG:32182 42708.6 5.27438e+06
  5270155.72117e+06"
       "wms_latlonboundingbox" "-59.7807 47.5557 -52.7934 51.6261"
       "wms_connectiontimeout" "60"
       ...
      END
      ...
  ```

In addition to the above layer parameters, you have to set the `IMAGEPATH` value in the `WEB` object of your mapfile to point to a valid and writable directory. MapServer will use this directory to store temporary files downloaded from the remote servers. The temporary files are automatically deleted by MapServer so you won't notice them... but a valid `IMAGEPATH` is still required.

3.4.3. Limitations/To Do

1. MapServer WMS connections always request exceptions "`INIMAGE`", but it may be nice to support XML exceptions at some point and return the message via the MapServer error reporting mechanisms.

2. PNG format with `transparent=true` does not work well with all servers. For instance, some servers use the alpha channel for transparency (RGBA images), but this is not well supported by MapServer at the moment, so you may be forced to request GIF format maps in those cases. Transparent PNGs produced by the MapServer WMS work well though, since they are platted images.

3. `GetFeatureInfo` is not fully supported yet since the output of `GetFeatureInfo` is left to the discretion of the remote server. A method `layer.getWMSFeatureInfoURL()` has been added to MapScript for applications that want to access `featureInfo` results and handle them directly.

4. MapServer does not attempt to fetch the layer's capabilities. Doing so at every map draw would be extremely inefficient, and caching that information does not belong in the core of MapServer. This is better done at the application level, in a script, and only the necessary information is passed to the MapServer core via the `CONNECTION` string and metadata.

Note: DM Solutions will soon release the MapBrowser PHP application which demonstrates the use of PHP to fetch and parse remote WMS server capabilities.

3.4.4. Additional Information: Using ArcExplorer 4 as a Client for WMS-compliant UMN Mapserver

ArcExplorer is a freely downloadable GIS viewer offered by ESRI. With version 4, ArcExplorer has an extension that supports the WMS and WFS specifications. Because UMN Mapserver also supports these two standards, you can use ArcExplorer as a client application. This means you need not be limited by the event/response model commonly seen in Web browser based viewers, or worry about writing any client side code (such as Javascript) that may be difficult to support on a variety of browsers. As a plus, ArcExplorer is written in Java and will run on Windows, Linux, Solaris, MacOS X, Irix, HP-UX and AIX. Besides, it's a good example of how the adoption of standards can greatly increase the options available to GIS providers and users. This example also shows how to use PHP to modify responses and make Web clients WMS 1.1.1 compliant. For setup information see the following URL:

```
http://mapserver.gis.umn.edu/cgi-bin/wiki.pl?WMSMapserverArcExplorer
```

3.5. Recipe 4: Intergraph GeoMedia WebMap WMS Adapter Kit

The GeoMedia WMS Adapter Kit provides a web interface to facilitate the creation of OGC WMS servers using GeoMedia WebMap.

The WebMap WMS Adapter Kit is used to generate OGC WMS version 1.1.0. The WMS created by this Adapter Kit will support three operations: `GetCapabilities`, `GetMap` and `GetFeatureInfo`, the first two of which are required of every WMS. The Kit allows the user to specify display rules (symbology and display range), service information, operations information, layers information etc.

The WMS creates a picture (image) of the data/layers requested. GeoMedia WebMap based WMS sites can supply images in the form of JPG and PNG. PNG allows for transparency and therefore enables several different layers to be displayed in a single viewer. You can become familiar with the kit by creating a WMS using the default dataset provided in the kit. The kit also provides instructions on how to customize the adapter kit in order to create WMS sites with other datasets.

3.5.1. Step 1: Preliminary Requirements

Note: If you download the latest hot fix to the product (available from the "WebMap Users Support" line available on same page), you will get any updates to the WMS Adapter Kit as well[71]. See the following URL:

```
http://www.Intergrraph.com/gis/support/GepMeddiaWebMap.asp
```

Note: It is recommended to read the User Experience section (Example 3) in Chapter 2 from Harvard University that outlines useful guidelines for installing and using the GeoMedia WebMap WMS Adapter Kit.

1. Create a virtual directory for the WMS. For example, when using a Microsoft IIS http server, under: `.{Drive}:\inetpub\wwwroot`. And, make a directory called `wms`.

2. Extract the WMS Kit zip file into the directory created in Step 1. The extracted files are:

- `global.asa` – used to store the physical path to the Cache folder used by GeoMedia WebMap
- `\USSample`
- `setup.asp` – the interface for creating the WMS
- `createCMDF.asp` – used by `setup.asp` to create the `Wmscmdf.cmdf` file for the WMS in the `.\Warehouses` folder
- `createCap.asp` – used by `setup.asp` to create the capabilities document, `capabilities.xml` in the .\Capabilities folder.
- `createStyles.asp` – used by `setup.asp` to create a style (`DisplayRule`) for each Layer in SLD format in the `.\Styles` folder, and also create the legend image (20x15pixels) for each style in the `.\Legends` folder
- `request.asp` – the asp file used to handle the client's request/response
- coordinate systems files – the coordinate systems files supported by the WMS are located at `.\CSF` folder
- sample Dataset (`USSample.mdb`) – the USSample dataset is located in the `.\Warehouses` folder.
- `Legend.mdb` – used for creating Exception messages in image format and also for creating legend images in step 7. Do not move or modify the `legend.mdb` file.

[71] *GeoMedia WebMap Users Support website: http://www.Intergrraph.com/gis/support/GepMeddiaWebMap.asp.*

3. Setup the website using Microsoft Internet Information Server (IIS) as in the following steps:

   a. Run Start>Settings>Control Panel > Administrative Tools>Internet Service Manager

   b. Create a New Virtual Directory on your Default Web Site with the following properties

      i. Alias: `wms`

      ii. Directory: the directory created in step 1

      iii. Properties: Read, Run Scripts

   c. Right click the virtual directory you just created and select Properties

      i. In the Application Settings frame, select Configuration

      ii. In the App Options tab, "Enable parent paths"

      iii. In the App Debugging tab, select/enable both debug options (after setup is successful, deselect both debug options)

   d. Click OK twice.

   e. Close IIS.

4. Edit the following variables in the global.asa file

Variable	***Data Assigned***
Cache	*The physical path to the directory used for the GeoMedia WebMap cache.*

Note: The files mentioned in the next steps are located in the `wms\USSample` directory.

5. Put all the coordinate system files (`.csf`) that your WMS must support in the folder `./CSF`. The naming rule of the CSF file is: if the CS file is `EPSG:4326`, then name the CS file `EPSG4326.csf`.

Some sample CSF files are delivered in the folder; you can add or remove files according to your needs. However, the file `EPSG4326.csf` is required by the WMS Adapter Kit, so it should not be removed.

6. Set permissions to allow the `IUSER_<YourNodeName>` (Internet Guest Account) to have Modify and Write permissions for the `wms/USSample` folder.

   a. Use the Windows File Explorer to locate the `wms/USSample` folder.

   b. Right click the USSample folder and choose Properties.

   c. Click the Security tab on the Properties dialog box.

   d. Click Add.

   e. Select <Your Computer Name> to list the accounts for your system.

f. Select the Internet Guest Account for your machine and set Permissions to allow Modify and Write.

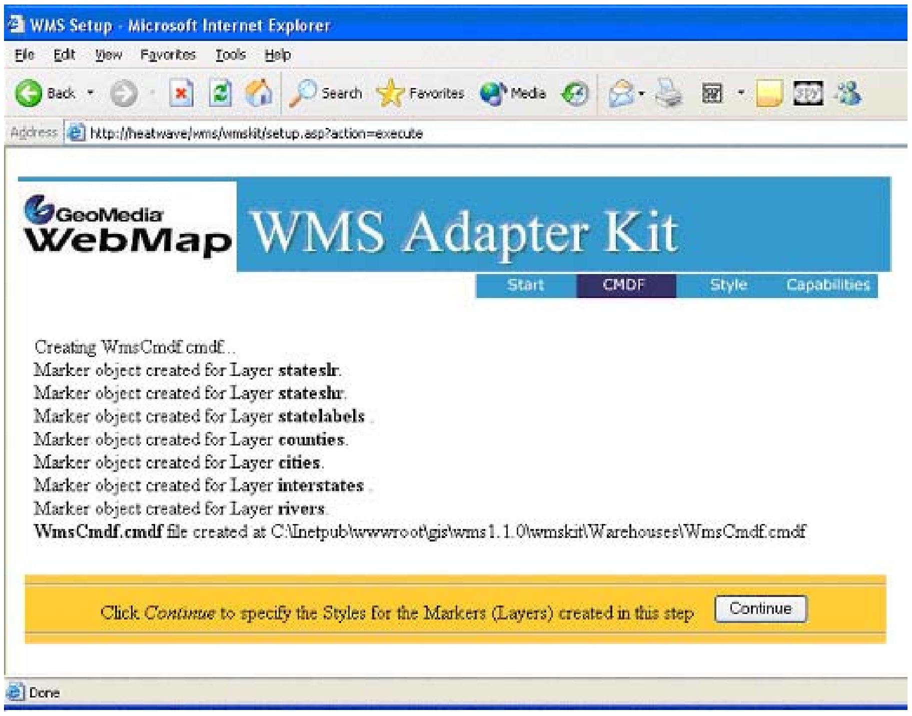
This is to allow the interface set up process (setup.asp in step 7) to have sufficient permissions to the `wms/USSample` directory. To close this security hole, Step 10 will instruct you to roll back the security setting of the created WMS directory to be Read and Execute only.

3.5.2. Step 2: Creating your WMS using the GeoMedia WebMap WMS Adapter Kit

7. Enter the following URL in the address box of a Web browser and follow the instructions step-by-step on the interface to create your WMS:

```
http://{YourNodeName}/wms/USSample/setup.asp
```

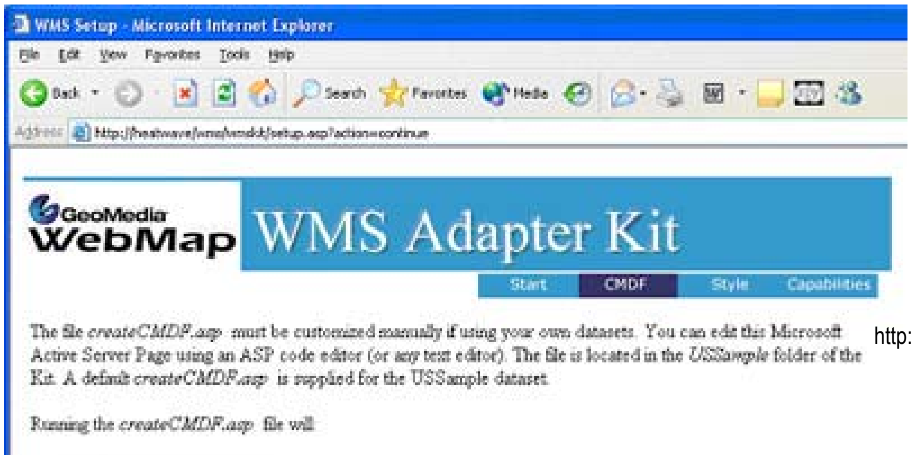
The first step will create the file `WmsCmdf.cmdf` (Figure 29) for you. (If you will be using your own dataset, you will need to customize the createCMDF.asp (Figure 30) script. See the section below, which gives you more information about setting up your own dataset.


WMS Setup - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://heatwave/wms/wmskit/setup.asp?action=execute
GeoMedia WebMap
WMS Adapter Kit
Start
CMDF
Style
Capabilities
Continue
Done

Creating WmsCmdf.cmdf...
Marker object created for Layer **stateslr**.
Marker object created for Layer **stateshr**.
Marker object created for Layer **statelabels**.
Marker object created for Layer **counties**.
Marker object created for Layer **cities**.
Marker object created for Layer **interstates**.
Marker object created for Layer **rivers**.
WmsCmdf.cmdf file created at C:\Inetpub\wwwroot\gis\wms1.1.0\wmskit\Warehouses\WmsCmdf.cmdf

Click *Continue* to specify the Styles for the Markers (Layers) created in this step

Figure 29:WmsCmdf.cmdf

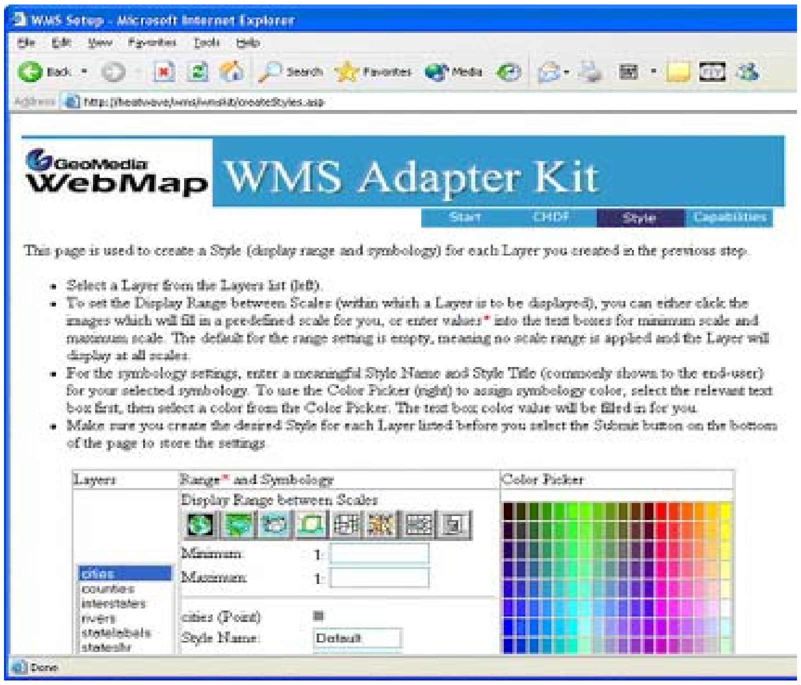

WMS Setup - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://heatwave/wms/wmskit/setup.asp?action=continue
GeoMedia WebMap
WMS Adapter Kit
Start
CMDF
Style
Capabilities

The file *createCMDF.asp* must be customized manually if using your own datasets. You can edit this Microsoft Active Server Page using an ASP code editor (or any text editor). The file is located in the *USSample* folder of the Kit. A default *createCMDF.asp* is supplied for the USSample dataset.

Running the *createCMDF.asp* file will

Figure 30: CreateCMDF.asp Window

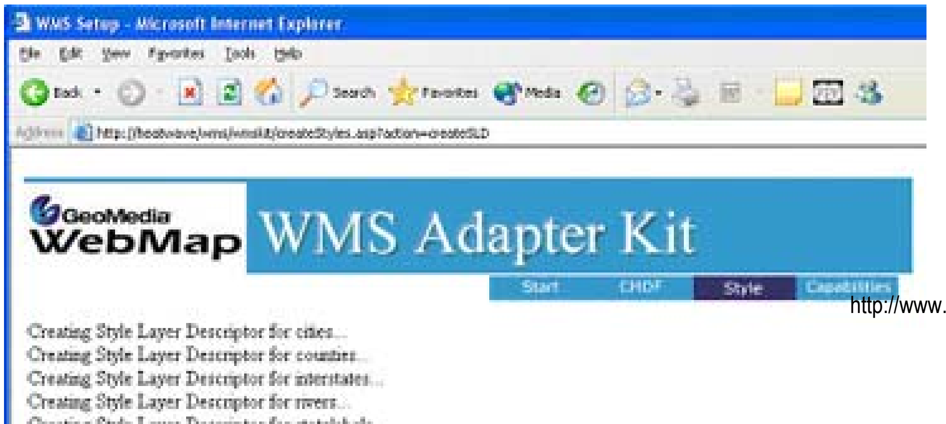
The second step (Figure 31 & 32) will create the file `WmsStyles.xml` in SLD (Styled Layer Descriptor) format for you. This file is used to store display Style (symbology) information (WebMap DisplayRules) for all the layers (WebMap Marker objects).


WMS Setup - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://heatwave/wms/wmskit/createStyles.asp
GeoMedia WebMap
WMS Adapter Kit
Start CMDF Style Capabilities

This page is used to create a Style (display range and symbology) for each Layer you created in the previous step.

- Select a Layer from the Layers list (left).
- To set the Display Range between Scales (within which a Layer is to be displayed), you can either click the images which will fill in a pre-defined scale for you, or enter values* into the text boxes for minimum scale and maximum scale. The default for the range setting is empty, meaning no scale range is applied and the Layer will display at all scales.
- For the symbology settings, enter a meaningful Style Name and Style Title (commonly shown to the end-user) for your selected symbology. To use the Color Picker (right) to assign symbology color, select the relevant text box first, then select a color from the Color Picker. The text box color value will be filled in for you.
- Make sure you create the desired Style for each Layer listed before you select the Submit button on the bottom of the page to store the settings.

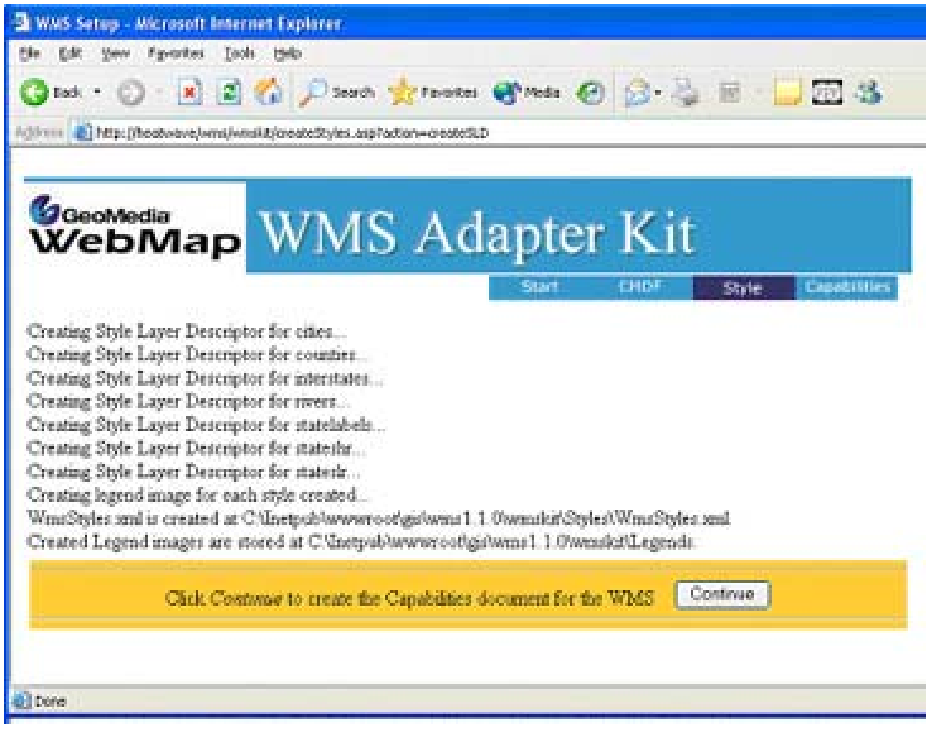
Figure 31: Style SLD


WMS Setup - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://heatwave/wms/wmskit/createStyles.asp?action=createSLD
GeoMedia WebMap
WMS Adapter Kit
Start CMDF Style Capabilities

Creating Style Layer Descriptor for cities...
Creating Style Layer Descriptor for counties...
Creating Style Layer Descriptor for interstates...
Creating Style Layer Descriptor for rivers...

Figure 32: WmsStyles.xml Window

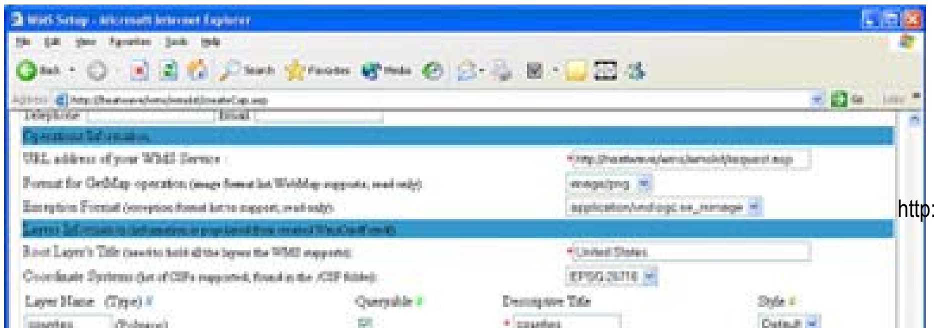
The third step will create the file `Capabilities.xml` for the WMS.


WMS Setup - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://heatwave/wms/wmskit/createStyles.asp?action=createSLD
GeoMedia WebMap
WMS Adapter Kit
Start
CMDF
Style
Capabilities
Continue
Done

Creating Style Layer Descriptor for cities...
Creating Style Layer Descriptor for counties...
Creating Style Layer Descriptor for interstates...
Creating Style Layer Descriptor for rivers...
Creating Style Layer Descriptor for statelabels...
Creating Style Layer Descriptor for statehr...
Creating Style Layer Descriptor for statelr...
Creating legend image for each style created...
WmsStyles.xml is created at C:\Inetpub\wwwroot\gis\wms1.1.0\wmskit\Styles\WmsStyles.xml
Created Legend images are stored at C:\Inetpub\wwwroot\gis\wms1.1.0\wmskit\Legends

Click *Continue* to create the Capabilities document for the WMS

Figure 33: Capabilities.xml Window


WMS Setup - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://heatwave/wms/wmskit/createCap.asp
Go

Telephone Email

Operations Information

URL address of your WMS Service : * http://heatwave/wms/wmskit/request.asp

Format for GetMap operation (image format list WebMap supports, read only) image/png

Exception Format (exception format list to support, read only) application/vnd.ogc.se_inimage

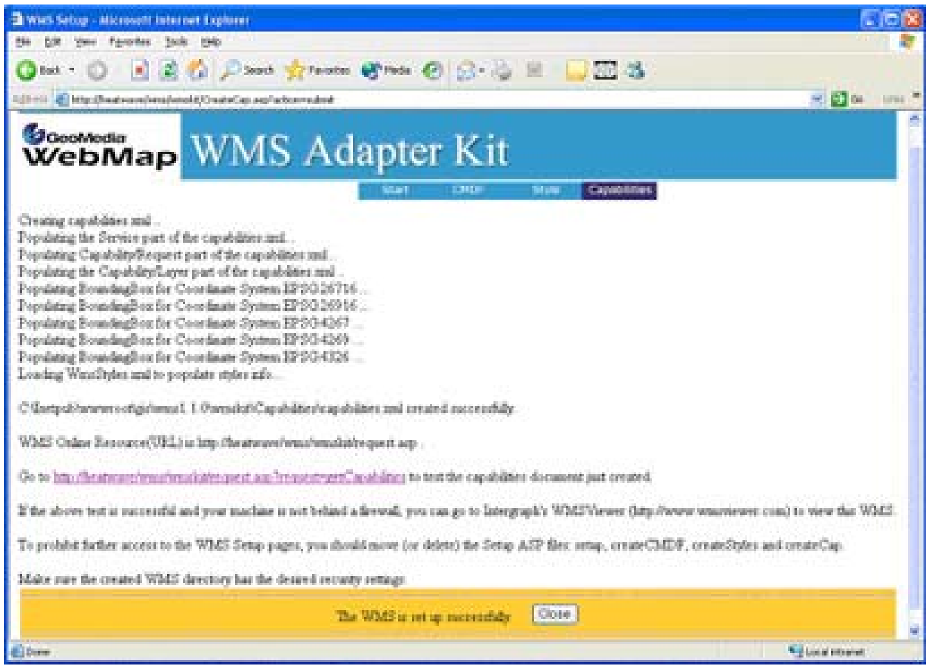
Layer Information (information is populated from created WmsConf.xml)

Root Layer's Title (used to hold all the layers the WMS supports) * United States

Coordinate Systems (list of CSFs supported, found in the /CSF folder) EPSG:26716

Layer Name	(Type)	Queryable	Descriptive Title	Style
counties	(Polygon)	✓	counties	Default

Figure 34: Capabilities.XML Window (Part 2)


WMS Setup - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
http://heatwave/wms/wmskit/CreateCap.asp?action=submit
Go
GeoMedia WebMap
WMS Adapter Kit
Start
CMDF
Style
Capabilities

Creating capabilities.xml ...
Populating the Service part of the capabilities.xml ...
Populating Capability/Request part of the capabilities.xml ...
Populating the Capability/Layer part of the capabilities.xml ...
Populating BoundingBox for Coordinate System EPSG:26716 ...
Populating BoundingBox for Coordinate System EPSG:26916 ...
Populating BoundingBox for Coordinate System EPSG:4267 ...
Populating BoundingBox for Coordinate System EPSG:4269 ...
Populating BoundingBox for Coordinate System EPSG:4326 ...
Loading WmsStyles.xml to populate styles info ...

C:\Inetpub\wwwroot\ogis\wms1.1.0\wmskit\Capabilities\capabilities.xml created successfully.

WMS Online Resource(URL) is http://heatwave/wms/wmskit/request.asp .

Go to http://heatwave/wms/wmskit/request.asp?request=getCapabilities to test the capabilities document just created.

If the above test is successful and your machine is not behind a firewall, you can go to Intergraph's WMSViewer (http://www.wmsviewer.com) to view this WMS.

To prohibit further access to the WMS Setup pages, you should move (or delete) the Setup ASP files: setup, createCMDF, createStyles and createCap.

Make sure the created WMS directory has the desired security settings.

The WMS is set up successfully

Figure 35: Capabilities.XML Window (Part 3)

3.5.3. Step 3: Test your WMS Service

If your computer is available on the World Wide Web and is not behind a restrictive firewall, you can test your WMS setup by visiting the Intergraph WMS Viewer and adding your WMS URL as a Server:

1. Visit Intergraph's WMSViewer[72].

2. Select the Servers button on the left frame

3. Click Add New, enter your URL http://{YourDomain}/wms/USSample/Request.asp,

   and select OK.

Or, to test the default site (sample) locally, the following URLs can be entered in the a browser:

```
http://{YourNodeName}/wms/USSample/request.asp?REQUEST=capabilities
```

or

```
http://{YourNodeName}/wms/USSample/request.asp?VERSION=1.1.0&
REQUEST=GetMap&BBOX=-125,20,-65,55&SRS=EPSG:4326&HEIGHT=467&
WIDTH=800&FORMAT=image/png&BGCOLOR=0xFFFFFF&LAYERS=stateslr,stateshr&
STYLES=White,White&TRANSPARENT=TRUE&
EXCEPTIONS=application/vnd.ogc.se_inimage
```

4. The files setup.asp, createCMDF.asp, createStyles.asp, and createCap.asp, which are only used in setup, may be moved to another location so they are not accessible from Web clients.

5. Roll back the security setting of the created WMS directory in Step 6 so Permissions are Read and Execute only.

6. Deselect the debug setting for the wms directory in the Internet Information Service Manager dialog box.

3.5.4. Setting Up Your Own Dataset

If you wish to create a new WMS with your own dataset, you only need to edit some parts of the createCMDF.asp file.

a) Follow these installation instructions through Step 6. Then, modify the createCMDF.asp to accommodate your own dataset. In general, you will need to connect to each dataset you want to support (no restriction on the physical location of the dataset), then, create a WebMap Marker (not Query) object for each layer you want to support for your WMS.

The sections of code you will need to modify or customize for your data are contained inside the commented areas marked as follows:

```
'@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
the code you need to modify appears here
'@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
```

b) Continue with the WMS Installation Instructions at step 7.

[72] *Intergraph's WMSViewer at* http://www.wmsviewer.com

3.6. Recipe 5: How to build a WMS from Free Parts

This example examines a WMS built using Python. This WMS works by keeping a 3600x1800 JPEG image which contains a full map of the world from -180,-90 to 180,90 and sending chunks of it out in response to map requests by clients.

The list of what's needed to build it is:

- A Linux box (RedHat 7)[73]
- Python 2.1.1c1 - get it at Python.org[74]
- Apache 1.3.20 - get it at Apache.org[75]
- mod_python 2.7.5 - get it at modpython.org[76]
- Netpbm - get it at SourceForge[77]
- A decent JPEG image of the world (note: that's a 1.7 MByte file). One can be obtained from CubeWerx[78] by running their CubeWerx Web Demo[79]. Depending on your hardware/software setup, you may have to pull in things like Tcl/Tk, JPEG libraries, etc. in order to build some of this stuff.

Here's the result:

basic-wms2.py --> see Appendix B. Also see the license page[80]. This could probably also be done in PERL or Tcl or the programming environment of your choice.

The general flow from a client's perspective is to first ask the WMS to return a list of what map layers it can draw, then the client can ask for maps by requesting specific layers or combinations of layers over specific geographic regions. In practice, the hard parts of this are done by people who set up web pages that provide user interface controls. Two examples of this are the CubeWerx demo and the NASA Digital Earth viewer[81].

These are known as "viewer clients." A viewer client can be as simple as a web browser that you paste a fully formed WMS request into or as complex as a commercial GIS system that can make WMS requests based on the context of what a user is doing.

In this example, the viewer client is a Web browser. All examples will be shown as URLs that you can click on.

```
http://www.intl-interfaces.net/cookbook/WMS/basic-wms2/basic-wms2.py?
WMTVER=1.0.0&REQUEST=map&LAYERS=RELIEF&STYLES=default&SRS=EPSG:4326&BBOX
```

[73] *Linux Box from Reddhat☹: http://www.redhat.com/*
[74] *Python.org: http://www.python.org/*
[75] *Apache.org: http://www.apache.org/*
[76] *Modpython.org http://modpython.org/*
[77] *SourceForge: http://sourceforge.net/projects/netpbm/*
[78] *Cubewerx: http://www.cubewerx.com/*
[79] *Cubewerx WMS demo: http://www.cubewerx.com/demo/cubeview.cgi*
[80] *International-Interfaces license: fttp://www.intl-interfaces.net/cookbook/WMS/#license*
[81] *NASA Digital Earth viewer: http://viewer.digitalearth.gov/*

```
=-2.197265625,39.55078125,20.302734375,50.80078125
&WIDTH=256&HEIGHT=128&FORMAT=PNG
```

Note: All examples of URLs or other code that are too wide are broken up into separate lines. In the case of a URL such as this one, if you were to copy it and paste it into a browser, you'd need to copy each of the lines and tack them together with no white space. The examples will be linked to the same URL so you can just click on them instead of having to cut and paste.

WMS has a base URI prefix (or URL prefix if you prefer)

```
http://www.intl-interfaces.net/cookbook/WMS/basic-wms2/basic-wms2.py?
```

In order to find out what layers it supplies and what projections it supports, a client makes a "Capabilities Request." Here's the prefix with a Capabilities request:

```
http://www.intl-interfaces.net/cookbook/WMS/basic-wms2/basic-
wms2.py?request=GetCapabilities&wmtver=1.1.1
```

The response is formatted according to the WMS Appendix A. Comments in that DTD are considered normative and must be followed by WMS providers. The response has to be valid XML, meaning that it should pass a validation test[82]. The response has a MIME type of text/xml. Let's take a quick look at the parts of the XML document. First there's some XML info:

```
<?xml version='1.0' encoding="UTF-8" standalone="no"?>
<!DOCTYPE WMT_MS_Capabilities SYSTEM
 "http://www.digitalearth.gov/wmt/xml/capabilities_1_0_0.dtd">
<WMT_MS_Capabilities version="1.0.0" [
  <!ELEMENT VendorSpecificCapabilities EMPTY>
```

The DOCTYPE entry states that this is a WMT_MS_Capabilities document and that you can find the DTD for this file at the DigitalEarth website[83]. The part about VendorSpecificCapabilities means that this WMS has none.

The next section describes the overall service. (There's a lot of work going on within OGC about service descriptions and service models[84]. The WMS 1.0.0 spec predates most of this work. Thus, if you look at the 1.0.7 spec or at other materials coming from OGC, you will see different points on an evolutionary path. The ultimate goal is to develop a service model that can be used to describe many spatial services.)

```
<!-- Service Metadata -->
<Service>
 <!-- The WMT-defined name for this type of service -->
 <Name>GetMap</Name>
```

A WMS must be named `GetMap`. It's stated so in the DTD.

```
<!-- Human-readable title for pick lists -->
 <Title>Basic Map Server</Title>
 <!-- Narrative description providing additional information -->
 <Abstract>Basic WMS Map Server built as an example for a WMS cookbook
      Contact: adoyle@intl-interfaces.com.</Abstract>
 <Keywords>Demo WMS Cookbook</Keywords>
```

As a WMS implementer/provider, you pick the Title, Abstract, and Keywords. The Title is meant to be used in user interfaces software (i.e. in Viewer Clients) as part of a list of map servers that the Viewer Client can access. Keep it short. Abstract is meant to provide a longer description of the service. Keep it

[82] XML validation test tool: http://www.stg.brown.edu/service/xmlvalid/
[83] Capabilities 1.0.0 DTD file: http://www.digitalearth.gov/wmt/xml/capabilities 1 0 0.dtd
[84] About OGC work on service descriptions and service models: http://www.intl-interfaces.net/servicemodel/

informative. The Keywords are meant to be useful if someone were searching for your service. These are hard to select.

```
<!-- Top-level address of service or service provider.
See also onlineResource attributes of <dcpType> children.-->
<OnlineResource>
   http://www.intl-interfaces.net/cookbook/WMS/
 </OnlineResource>
 <!-- Fees or access constraints imposed. -->
 <Fees>none</Fees>
 <AccessConstraints>none</AccessConstraints>
</Service>
```

The OnlinResource should contain a URI that leads to a description of the service. In this case, it points to this WMS Cookbook. The Fees and AccessConstraints elements are really not well defined except that the spec states that the string none indicates no constraint exists.

Note that all the elements of the Service section can be gathered into a searchable catalog of WMS implementations. In such a catalog, it would be possible to look for WMS implementations that have no associated Fees, or to find those whose Keywords do not include the term "Demo" and so on. If you searched a catalog and found an entry that interests you, you could use the OnlineResource to find out more about that implementation. In fact, the rest of the capabilities document, the Capability section is used in catalogs as well.

The Capability section for basic-wms has three subsections: Request, Exception, and Layer.

```
<Request>
  <Map>
   <Format>
    <PNG />
    <JPEG />
    <PPM />
    <TIFF />
   </Format>
   <DCPType>
    <HTTP>
     <Get onlineResource=
     "http://www.intl-interfaces.net/cookbook/WMS/basic-wms2/basic-
ms2.py?" />
    </HTTP>
   </DCPType>
  </Map>
  <Capabilities>
   <Format>
    <WMS_XML />
   </Format>
   <DCPType>
    <HTTP>
     <Get onlineResource=
        "http://www.intl-interfaces.net/cookbook/WMS/basic-wms2/basic-
wms2.py?" />
    </HTTP>
   </DCPType>
  </Capabilities>
 </Request>
```

As you can see, the Request section is split into two sections, Map and Capabilities. These describe the two operations that this WMS supports. For the Map request, it can handle PNG, JPEG,

PPM, and TIFF return formats, and is listening for Map requests at the URI specified by onlineResource. These requests must be made using the HTTP Get request (as opposed to HTTP Put or SOAP or anything else). For the Capabilities request, it can return WMS_XML (the tag "XML" was already reserved, hence the WMS_ prefix) and again, listens at the onlineResource URI.

It's worth noting that the WMS spec allows the onlineResource values for each request type to be different. Depending on how you set up your WMS implementation, it may be more convenient for the two to be the same or it may be easier for them to be different. A WMS client should always start with a Capabilities request to find the Map request URI. Never assume they are the same.

```
  <Exception>
   <Format>
    <INIMAGE />
    <BLANK />
   </Format>
  </Exception>
```

The Exception tag tells about the exception formats a client can ask for when making requests. The basic-wms advertises that it can return INIMAGE or BLANK style exceptions.

At last we get to the Layer section. This is how the map server tells the clients what kinds of maps they can request.

```
    <Layer>
     <Title>Demo Map Server</Title>
     <SRS>EPSG:4326</SRS>
     <LatLonBoundingBox minx="-180" miny="-90" maxx="180" maxy="90" />
     <Layer queryable="0">
      <Name>RELIEF</Name>
      <Title>Relief (ETOPO/GTOPO)</Title>
      <Abstract>Colored relief map with political boundaries and
   coastlines</Abstract>
     </Layer>
     </Layer>
```

Our map server says that it has a top level Layer with a Title, SRS, and LatLonBoundingBox, and a single sub-Layer. Note that the top level Layer has no Name. That means that it's just being used to gather other layers into a logical group. Instead of that grouping, we could have used the form below.

```
    <Layer queryable="0">
      <SRS>EPSG:4326</SRS>
      <Name>RELIEF</Name>
      <Title>Relief (ETOPO/GTOPO)</Title>
      <Abstract>Colored relief map with political boundaries and
   coastlines</Abstract>
     <LatLonBoundingBox minx="-180" miny="-90" maxx="180" maxy="90" />
    </Layer>
```

However, the WMS 1.0.0 DTD (Appendix A) shows that there can only be zero or one Layer elements inside a Capability. By using the top level grouping Layer, we ensure that later additions of new Layers will be easier to do. So here's the Layer info again:

```
    <Layer>
     <Title>Demo Map Server</Title>
     <SRS>EPSG:4326</SRS>
     <LatLonBoundingBox minx="-180" miny="-90" maxx="180" maxy="90" />
     <Layer queryable="0">
      <Name>RELIEF</Name>
      <Title>Relief (ETOPO/GTOPO)</Title>
      <Abstract>Colored relief map with political boundaries and
   coastlines</Abstract>
     </Layer>
     </Layer>
```

The top level Layer has an SRS and a LatLonBoundingBox. These are inherited by the sub Layer elements. The sub Layer elements can add SRS values to this list by including its own SRS element. It can replace the LatLonBoundingBox by including one of its own. These inheritance/replacement rules are given in a table in the DTD in Appendix A. Our single sub Layer must have a Name and a Title. We also include an Abstract. The queryable="0" attribute says that our WMS will not respond to `GetFeatureInfo` requests.

In order to actually test this WMS implementation, you can make use of a service offered by CubeWerx. If you go to their demo page, down at the bottom is a text entry box labeled "'URL of server: " If you enter the prefix of your WMS implementation in there and hit one of the "Go" buttons on the page, it will query your WMS for its capabilities, fill in the user-interface elements on the page, and let you exercise your WMS.

3.7. Recipe 6: GeoServNet (GSN) 3D Client

GeoServNet (GSN) is a suite of software that enables the distribution of both 2D and 3D geospatial information over a network. GSN has a Services Oriented Architecture (SOA) that is scalable, transparent and distributed. It utilizes Java and Java 3D technology and can be deployed with any platform. With GSN, geospatial information can be accessed anytime anywhere. The latest version of GeoServNet version 2.0 supports many 3D GIS navigation and query functions.

GSN 3D Client was developed to be OGC compatible for geospatial data access, display, processing and analysis in the Open GIS Critical Infrastructure Protection Initiative (CIPI). This client offers the end users of the CICE (Critical Infrastructure Collaborative Environment) to efficiently retrieve, visualize, query, and analyze integrated geospatial information (terrain, image, vector and other related information). The GSN 3D Client is a plug-and-play application client that can be viewed in any desktop web browser with authorization connecting to DOD, USGS, and GeoConnections Node. Consequently, this application client will also be ready to be a 3D client to the CGDI portal.

GSN 3D Client is a 3D visualization tool based on OGC WMS and WCS[85]. It includes two main parts. One is a simple 2D navigation tool that provides a visual interface based on OGC WMS for defining the target area and features. Another is a smart 3D viewer that addresses the geospatial information of the target area and features. The smart 3D viewer also provides an interface for 3D operations including analysis. The 2D navigation tool starts the 3D viewer each time. All necessary data are retrieved from the CIPI

[85] *WCS (Web Coverage Service) Interoperability Program Report: http://www.opengis.org/techno/discussions/02-024.pdf*

Nodes via OGC specifications standardized interfaces. The 3D viewer can be further transplanted and started by any other 2D viewer that is compatible with OpenGIS Specifications.

GSN 3D Client is capable of performing the following functions:

- 3D view operation: pan, roam, zoom in, zoom out, rotate, fly through, translate etc.
- 3D view enhancement operation: show layer, hide layer, add layer, remove layer etc.
- 3D query operation: layer property etc.
- 3D analysis operation: profile, visibility analysis, visible area analysis and simple flood simulation etc.

3.7.1. Steps for Installing the OGC 3D Viewer

3.7.1.1. Step 1: Preliminary Requirements

Check your computer system first. We recommend that it meets the following requirements:

1. CPU: 1G Hz

2. Random Memory: 128 Mb

3. Video Card: Gforce 2 or above

3.7.1.2. Step 2: Download

Download and save, in any folder, the client package ogc_3d_viewer.exe from the GeoServNet OGC version link from the GeoICT website[86].

3.7.1.3. Step 3: Install

Install the OGC 3D Viewer by run or by double-clicking on ogc_3d_viewer.exe file. This will automatically start to install this package in your computer. You just need to follow the installation guide to choose the install folder and shortcut folder as well as the name for the OGC 3D Viewer, and finally click the install button to finish the installation.

By default, the install folder is C:\Program Files\ OGC 3D Viewer, the shortcut folder is your \Start Menu\Programs, and the name is OGC 3D Viewer. To start your OGC 3D Viewer, by default, click on Start\Programs\OGC 3D Viewer. To uninstall your installed OGC 3D Viewer application, you need to go to your Control Panel\ Add/Remove Programs to remove it.

3.7.2. 2D Viewer Operation

After you install the OGC 3D Viewer, you can start the OGC 2D Viewer from Start\Programs\OGC 3D Viewer. The following sections introduce the 2D Viewer functions and operations:

[86] *GeoServNet OGC 3D client download: http://www.geoict.yorku.ca/plugin/ogc3dviewer.exe*

- GSN OGC Client V1.5
- OGC Web Services

3.7.2.1. GSN OGC Client V1.5

If you run the OGC 2D/3D Viewer executable application, the Graphical User Interface of the OGC 2D Viewer, GSN OGC Client V1.5, as shown in Figure 36 below, will be launched.

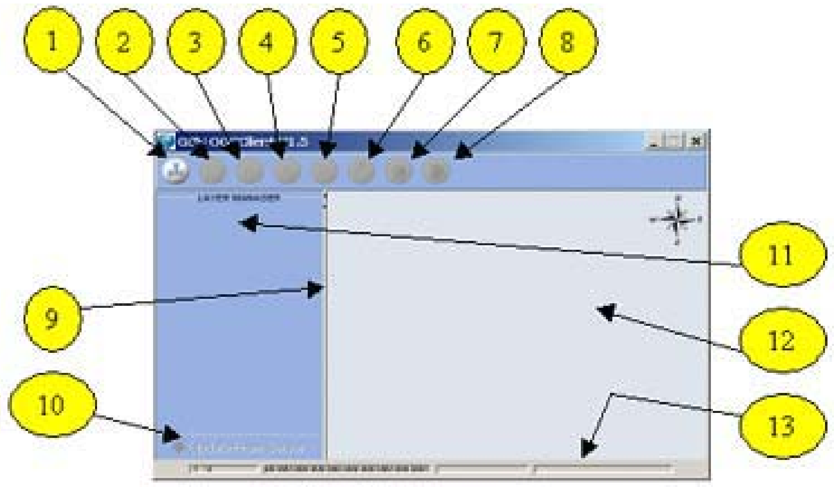


Figure 36: GSN OGC Client V1.5

1. *Connect to OGC Server button*: open the OGC Web Service dialog box by left-clicking on it first, or open the OGC Web Service dialog box and abandon all its stored data by left-clicking on it in the middle.

2. *Open OGC Server Properties Dialog Box button*: open the OGC Web Service dialog box by left-clicking on it.

3. *Zoom in by specified window / times button*: enlarge the selected area by left-clicking on it first, and then left-clicking on the map. Or by pressing and holding the left mouse button and then moving the mouse downwards to select the zoom in area on the map.

4. *Zoom out 2x button*: shrink the map to a half by first left-clicking on it and then left-clicking on the map.

5. *To Full Extent button*: restore the map to its original size by left-clicking on it.

6. *Pan In Current Scale button*: move the map within the map area by left-clicking on it first and then pressing and holding the left mouse button and dragging.

7. *Identify the selected single feature button*: Query for selected single layer by first left-clicking on it and then left-clicking on the specific location on the map.

8. *Load 3D Analysis Window button*: start 3D Analysis Window by left-clicking on it.

9. *Split bar:* adjusts the space between the layer manager box and 2D map area by resting the mouse on the bar and pressing, holding the left mouse button and then dragging to left or right, or left-clicking on one of the arrows on the bar.

10. *Update From Server button*: update the map from the server remotely by left-clicking on it.

11. *Layer Manager pane*: use this to contain, select and set for a layer or layer group, as in Figure 37. All the operations on the layer or layer groups can be accessed from the popup menu by right-clicking on layer or layer group and listed below:

    - set current layer: set the selected layer as current layer
    - move layer to: move the selected layer to the top of the layer which you click on next
    - delete this layer: Delete selected layer or layer group
    - set transparent: set the layer's background color to be transparent.

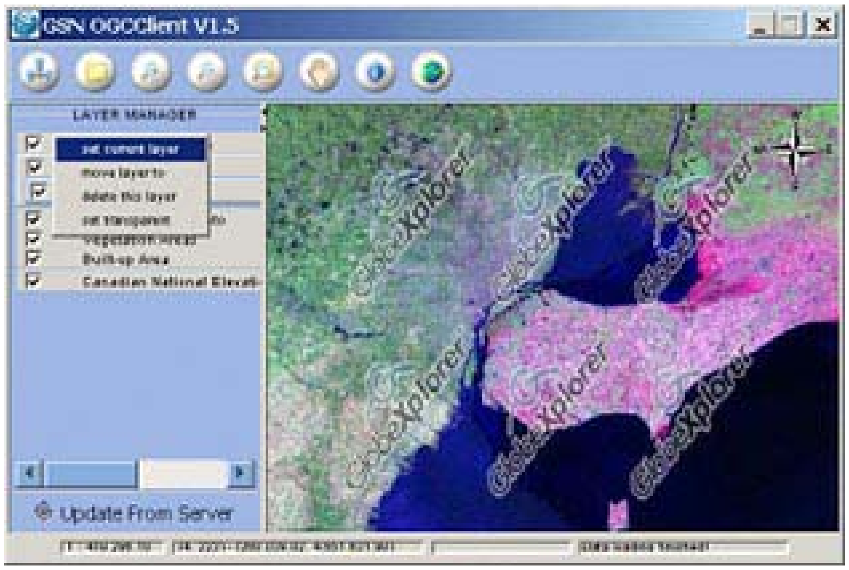


Figure 37: Layer Manager pane with a pop up menu

12. *2D Map viewer*: the area used to display the 2D map. All the operations on the 2D map can be accessed from item 3 to item 8 or from the Map viewer popup menu (Figure 38) by clicking the right mouse button on the map area; available functions are listed below:

    - Zoom in (Same as No. 3 above)

- Zoom out (Same as No. 4 above)
- Pan (Same as No. 6 above)
- Query (Same as No. 7 above)
- 3D Viewer (Same as No. 8 above)

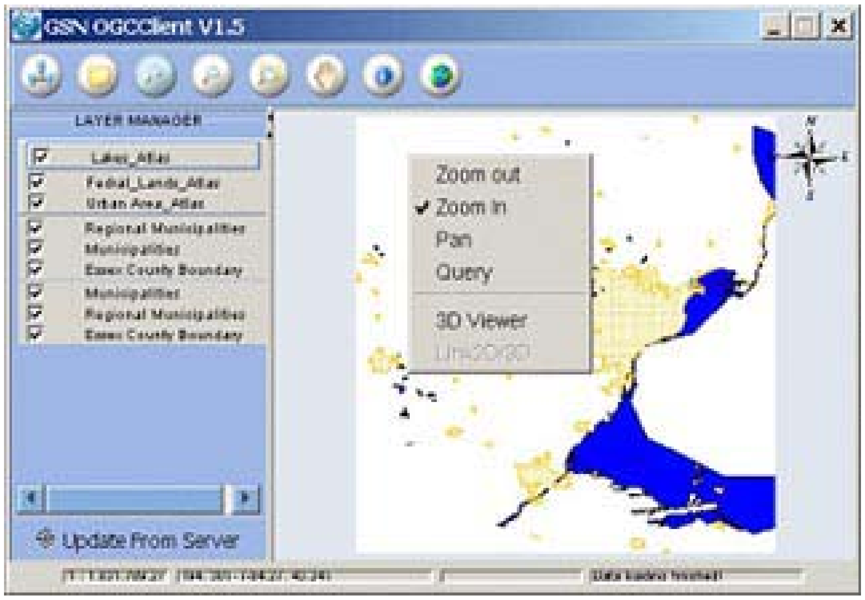
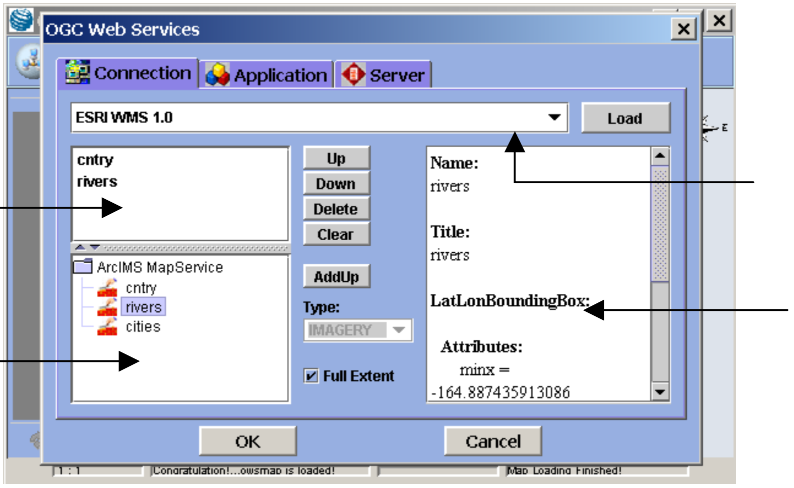


Figure 38: Map viewer popup menu

13. *States bar*: display the map scale, relative and absolute coordinates, progress bar and state message

3.7.2.1.1. *OGC Web Services*

After Connecting to the server by left mouse clicking on the OGC Server tab or by opening the OGC Server Properties Dialog Box Button on the 2D Viewer, the OGC Web Services dialog box will look as shown in Figure 39.



Name:
rivers

Title:
rivers

LatLonBoundingBox:

Attributes:
minx =
-164.887435913086

1 : 1 Congratulation!...owsmap is loaded! Map Loading Finished!

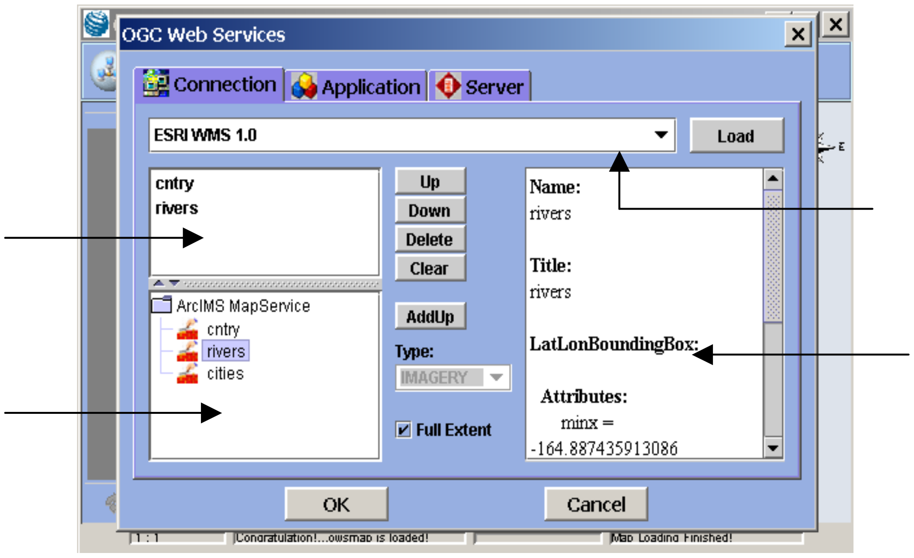
Figure 39: OGC Web Services dialog box

The OGC Web Service dialog box contains three tabbed panes and two buttons: Connection Pane, Application Pane, Server Pane, OK and Cancel as follows:

- Connection Pane: Used to connect and communicate with remote server.
- Application Pane: Used to display and edit user application.
- Server Pane: Used to display, add and edit the servers.
- Ok button: Used to validate and store the data of all the layers in the layer list pane within Connection pane, and to close the OGC Web Service dialog box.
- Cancel button: Used to abandon the selected layers and close the OGC Web Service dialog box.

Next, we will look at the details from each of three tabbed panes.

The Connection Pane (Figure 40) contains following components:



ESRI WMS 1.0

cntry
rivers

ArcIMS MapService
cntry
rivers
cities

IMAGERY

Name:
rivers

Title:
rivers

LatLonBoundingBox:

Attributes:
minx =
-164.887435913086

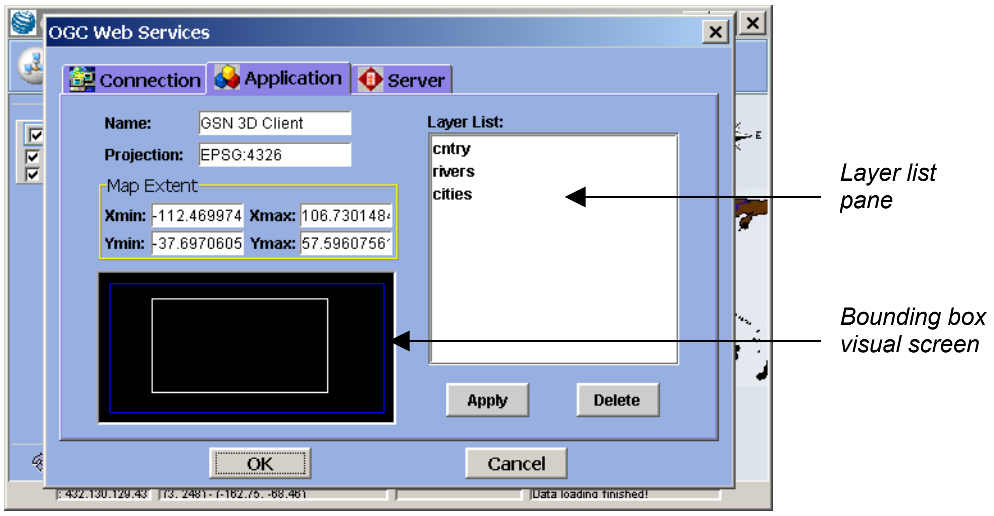
1 : 1 | Congratulation!...owsmap is loaded! | Map Loading Finished!

Figure 40: Connection Pane

- Selected server combo box: Select and display the selected server name from the server list
- Layer tree pane: Displays the layer tree. The operations involving the Layer tree pane:

  - Display layer data: Clicking on a layer once to select, high light and show its data in the layer data pane.
  - Add layer up to layer list: Double left click to add a layer to Selected layer list, or click AddUp button to add up the selected layer
- Layer list pane: Lists all the selected layers. Operations involving the Layer list pane:
  - Select a layer: Click on a layer to select, highlight and show its data.
  - Select layers: Ctrl + mouse clicking to select and highlight multiple layers.
- Layer data pane: Displays selected layer's data
- Load button: Connects to the selected remote server and load data from it.
- Up button: Moves the selected layer one row up in the Selected layer list pane.
- Down button: Moves the selected layer one row down in the Selected layer list pane.
- Delete button: Deletes one or multiple selected layers in the Selected layer list pane.
- Clear button: Clears all layers once in the Selected layer list pane.
- AddUp button: Adds a selected layer in the layer tree to in the Selected layer list pane.
- Map type combo box: Used to specify the map type.
- Full extent check box

The Application Pane (Figure 41) contains the following components:

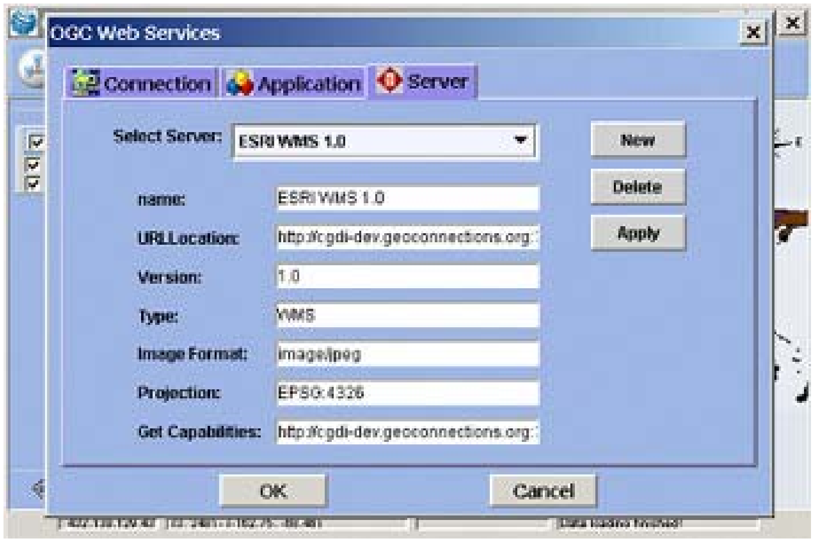


Name: GSN 3D Client
Projection: EPSG:4326
Map Extent
Xmin: -112.469974 Xmax: 106.730148
Ymin: -37.6970605 Ymax: 57.5960756
cntry
rivers
cities

Figure 41: Application Pane

- Name: Display user application's name.
- Projection: Display user application's projection.
- Map extent: Display user application's map extent.
- Layer list pane: Display all the layers selected by user from the connection pane.
    - Clicking on a layer: to select, high light and visual its bounding box in the Bounding box visual screen
    - Ctrl + mouse click: to select and high light multiple layers
- Bounding box visual screen: display layer bounding boxes. Its functions and operations:
    - Display all the bounding boxes of the layers in the layer list pane with multiple colors.
    - Draw and display the user defined bounding box by pressing and holding the left mouse button, then dragging it from left up to right down in the valid area.
    - Clear the user bounding box by clicking on the screen.
- Apply button: Validate and store the data of the user defined bounding box, application name and projection.
- Delete button: Delete one or multiple selected layers in the layer list pane.

The Server Pane (Figure 42) contains the following components:



Select Server: ESRI WMS 1.0

name: ESRI WMS 1.0

URLLocation: http://cgdi-dev.geoconnections.org:

Version: 1.0

Type: WMS

Image Format: image/jpeg

Projection: EPSG:4326

Get Capabilities: http://cgdi-dev.geoconnections.org:

Figure 42: Server Pane

- Select Server combo box: List all the stored server names.
- Name text area: Display and edit the new or selected server's name.
- URL location text area: Display and edit the new or selected server's URL location.
- Version text area: Display and edit the new or selected server's version.
- Type text area: Display and edit the new or selected server's type.
- Image format text area: Display and edit the new or selected server's image format.
- Projection text area: Display and edit the new or selected server's projection.
- Get Capabilities text area: Display and edit the URL location of the new or selected server's. capabilities.
- New button: Clear all the text areas to add a new server.
- Delete button: Delete the selected server and all its data from the server list combo box.
- Apply button: Validate and store the data from all the text areas to add in the new server.

3.7.2.2. Operate 2D Viewer by Example

The functions and operations of the 2D Viewer were discussed in the previous section. This section shows a step by step example of the typical operation of the 2D Viewer.

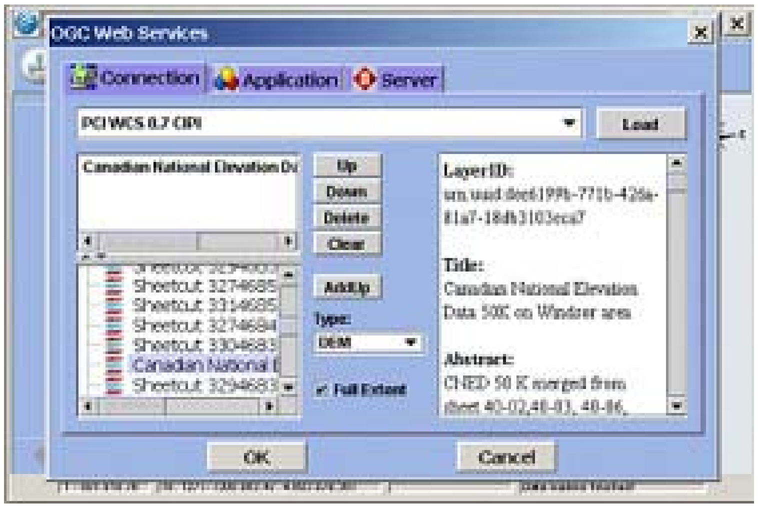
Step 1: Start or run the OGC 2D/3D Viewer: Run or double-click on ogc_3d_viewer.exe or the name you have given it. The GSN OGC Client V1.5 will appear on your screen.

Step 2: Open the OGC Web Services: Left-click on the Connect to OGC Server button to open the OGC Web Services window.

Step 3: Load data from server(s) using the Connection pane.

Step 3a: Select PCI WCS 0.7 CIPI server.

Step 3b: Load layer data: press Load button and wait for the layer tree to appear in the layer tree pane as in Figure 43:



PCI WCS 0.7 CIPI

Canadian National Elevation D

Sheetcut 3274685
Sheetcut 3314685
Sheetcut 3274684
Sheetcut 3304683
Canadian National E
Sheetcut 3294683

Type: DEM

✓ Full Extent

Layer ID:
urn:uuid:dee6199b-771b-42da-81a7-18db3103eca7

Title:
Canadian National Elevation Data 50K on Windsor area

Abstract:
CNED 50 K merged from sheet 40-02,40-03, 40-06,

Figure 43: Load Data & Select Layers

Step 3c: Select layer(s): traverse the layer tree to select a layer, check layer data from the right layer data pane, add the selected layer up to the layer list pane by double-click on it or push AddUp button. Manipulate the selected layers in the layer list pane using Up, Down, Delete and Clear buttons. Select map Type as DEM and let Full Extent be checked.

Step 3d: Store layers data and load map data: Press OK button (Figure 44). Press Cancel button to deselect.

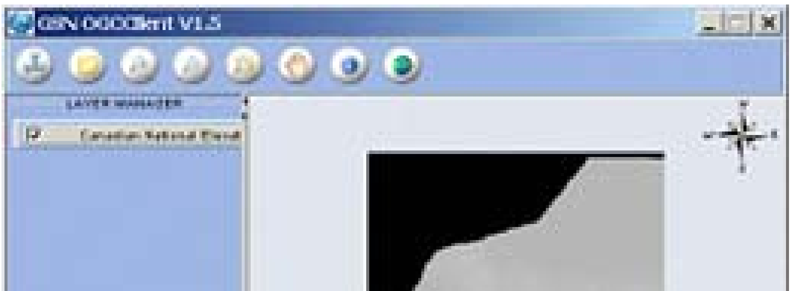


Figure 44: Store Layer Data and Load Map Data

Step 3e: Select layer(s) from another servers: repeat processes from Step 1 to Step 4 above for each server. For example, select layers from Cubewerx WMS server, USGS WMS server, GlobeXplorer WMS server.

Step 4: Manipulate the layers and 2D maps in the 2D Viewer.

Main operations:

- To display one or more map layers within one layer group which surrounded by blue rectangle and are from the same server, check the layer or layers you want to show under Layer Manager, uncheck all the rest and push the Update From Server to reload wanted map data.

- To display map layers not in one layer group (from different servers), only if the top layer or layers are transparent, the transparent layers and the first non-transparent map layer can be seen. The rest under them will be blocked.

- Use the buttons in the toolbar to zoom in, zoom out, restore full extent, to move the map around, to query to the current layer and to launch 3D Viewer. To add another layer into the Layer Manager, use the Open OGC Server Properties Dialog Box button, and repeat the steps in 3 above. To give up all the loaded layers in the 2D Viewer and start from very beginning, push Connect to OGC Server button.

Step 5: To create and edit your own application, use the Application pane introduced under section 'OGC Web Services' by pressing Open OGC Server Properties Dialog Box button, and clicking on Application pane tab.

Step 6: To add a new server or edit an existed one, use the Server pane introduced under section 'OGC Web Services' by pressing Open OGC Server Properties Dialog Box button, and clicking on Server pane tab.

3.7.2.3. 3 D Viewer Operations

After launching the 3D Analysis Window by left-clicking on the Load button or selecting 3D Viewer from the popup menu, or selecting Launch 3D from the menu that appears by right-clicking on the map area on the 2D Viewer, the 3D Viewer will be launched as shown Figure 45:

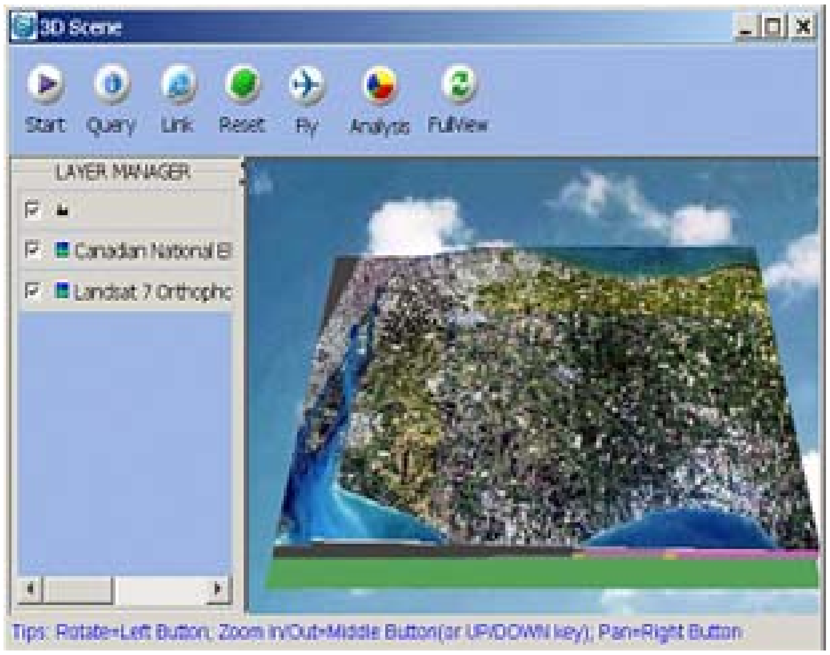


Figure 45: 3D Viewer - 3D Scene

Main functions and operations are the following:

1. Start/Stop: start/stop rotating 3D map by left-clicking on it

2. Query: Query from 3D viewer for current layer (Same as No. 7 in section 9.6.1)

3. Link: Show Hyper Link Document by left-clicking on it

4. Reset: Set All Layer on to Terrain by left-clicking on it

5. Fly: left-click on it to prompt up the Flight Control Panel to set for auto-flying

6. Analysis: there are four GIS Analysis Functions under its submenu. The submenu appears after a left-click on it.

   - *3D Location*: Get 3D location coordinates from the 2D map. To get 3D location coordinates, first left-click on the Analysis button and select the 3D Location submenu. Then left-click on the on the 2D map location of the 3D location whose coordinates you want to get. A pop up box shows you the 3D location coordinates and a red flag indicate the location on the 3D map.

   - *Profile & Distance*: Show the height change and distance information of multiple points on the selected path from 2D map. After choosing the Profile & Distance submenu, select the path you want from 2D map. Single click to select point, double click to end. Then a pop up box Profile will show you the diagram of the profile and distance.

- *Critical Surface:* Add a critical layer, such as the surface of water, to the 3D map, and the layer also is added into the Layer Manager pane.

- *Viewshed*: Find the viewable areas on the *DEM* layer with selected shed color on 3D map from dedicated location selected from 2D map. To show the viewable areas on the 3D map, after selected the *Viewshed* submenu, first select the location you would like to view from 3D Map. View from on the 2D map, a pop up box *Color Chooser* will show up. Use the scroll bar to a color and left-click on the *OK* button. Then you can find your view location indicated by a red flag and the viewable areas from the location are showed with your selected color on the 3D map.

7. *FullView*: Show 3D Full View by left-clicking on it.

8. *The moveable toolbar*: All the 7 buttons above are located on the moveable toolbar. It can be moved out and back from the 3D Scene. To move out the toolbar, hold the left mouse button on the toolbar and drag the mouse. To restore the remote toolbar, left-click on the close button on the upper right corner of the toolbar.

9. *Split bar*: Adjusts the space between the Layer Manager box and the 3D Map Viewer, like the one on the 2D Viewer.

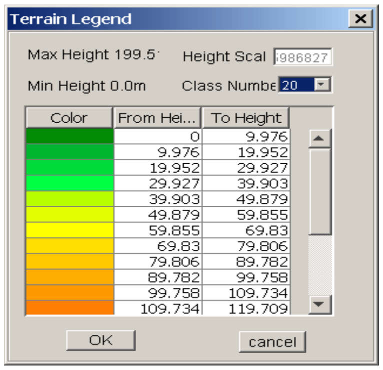
10. *Layer Manager*: List and manipulate all the selected layers in a group manner.

    Operations as follows:

    - *No DEM Data check box*: Add or not add DEM Data to the 3D map by checking or not checking.

    - Settings for the No DEM Data layer:

      Color setting: Color setting for height slots. To set color, first check the No DEM Data check box and set it as current layer, then right-click on the name of the layer and select the color submenu from the pop up menu. The color setting box named Terrain Legend will pop up as in Figure 46. Then choose a number from the class number combo box and press the OK button.



Max Height 199.5
Height Scal 986827
Min Height 0.0m
Class Numbe 20

Color	From Hei...	To Height
🟩	0	9.976
🟩	9.976	19.952
🟩	19.952	29.927
🟩	29.927	39.903
🟩	39.903	49.879
🟨	49.879	59.855
🟨	59.855	69.83
🟨	69.83	79.806
🟨	79.806	89.782
🟧	89.782	99.758
🟧	99.758	109.734
🟧	109.734	119.709

Figure 46: Terrain Legend

Height scale setting: Change height scale. To set height scale, first check the No DEM Data check box and set it as current layer; then right-click on the name of the layer and select the height scale submenu from the pop up menu. The Change Height Scale box will pop up. Use the scroll bar to set the height scale and press the OK button.

- *Set current layer*: Select a layer within the Layer Manager pane and set it as the current layer by left-click on the layer name make sure there is a pink rectangle surrounding the layer name.

- *BaseHeight setting*: Set base height for selected layer group. To set base height for selected layer group, first set the layer group as current layer group, then right-click on the names of the layer group and select the BaseHeight submenu from the pop up menu. The Change Base Height pop up box shows up. Use the scroll bar to set the base height and press the OK button.

11. *3D map viewer*: This is used to display the 3D map. To manipulate the 3D map by pressing and holding the left mouse button and moving the mouse around.

3.7.2.4. Operate 3D Viewer by Example

The previous sections showed how to use the 2D Viewer and all the functions and operations of the 3D Viewer. The next section shows how to use the 3D Viewer, using examples in a typical operation order.

Step 1: Launch 3D Viewer: After all the wanted map layers are loaded and managed under the Layer Manager in the 2D Viewer, left-click on Load 3D Analysis Window button on the toolbar or select 3D Viewer from the popup menu after right-clicking in the map area on the 2D Viewer. The 3D Viewer will be launched.

Step 2: Start/Stop rotating the 3D map by clicking on Start/Stop.

Step 3: Query to the current layer: First set a current layer to query, then click the Query button, then left mouse-click on a specific location on the 2D map or 3D map that you want information about. A pop up information box will give you the queried message.

Step 4: Reset all layers with base height not zero on to Terrain by left-clicking on Reset button. For example, in Figure 47, set ArcIMS MapService layer's base height to 1000 to separate it out of the Terrain. Push the Reset button to set the separated layers back to Terrain.

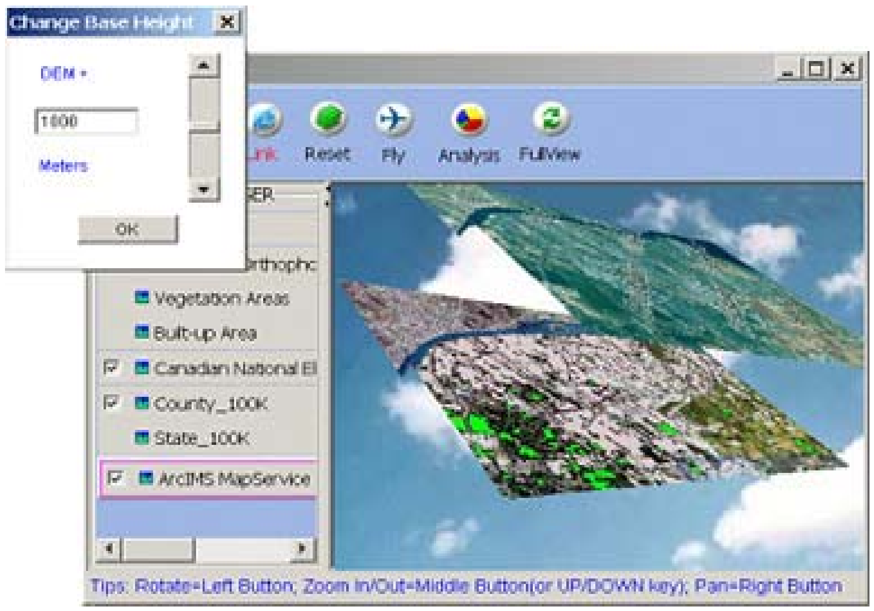


Figure 47: Set base height for ArcIMS MapService layer

Step 5: Auto-flying with selected path, height, speed and view deegree with the following steps:

Step 5a: Push Fly button, then set Height, Speed and Pitch in the Flight Control Panel.

Step 5b: Set path by push Path button on Flight Control Panel, then by single-clicks to select a specific fly path, double-click to end the selection (Figure 48).

Step 5c: Push the Start or Stop button on Flight Control Panel to start or stop the fly.

Step 6: Analysis: its functions and operations.

Step 7: Push FullView button at any time to set 3D map to its original full extent view.

Step 8: To manipulate the 3D map by you mouse: within 3D map area, press and hold the left mouse button and move the mouse around to display the 3D map in any direction.

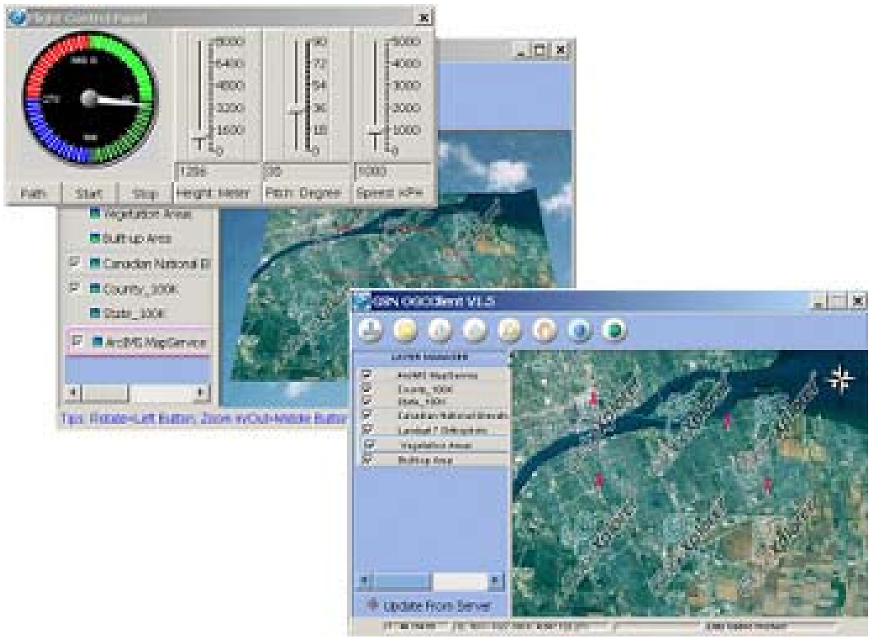


Figure 48: Setting Path, Height, Speed and Pitch

Appendix 1. deegree WMS Configuration Files (Recipe 2)

Appendix 1.A. Configuration.dtd

```
<!- d e e g r e e C O N F I G U R A T I O N   D E S C R I P T O R -->
<!--
This DTD defines the structure of the central configuration file
(configuration.xml) of the Dispatcher. It is assumed that not only WMS
servers but also stand alone applications makes use of a configuration
file that's valid to the DTD.
Last Update: July 30, 2002
-->
<!-- root element of the configuration document -->
<!ELEMENT deegreeConfiguration (Capabilities, FilterService*,
DEGService*, RenderService?)>
<!-- defines which capabilities document is accossiated to the
application. -->
<!ELEMENT Capabilities EMPTY>
<!ATTLIST Capabilities
resource CDATA #REQUIRED
>
<!-- defines which filter services are used by the appliction -->
<!ELEMENT FilterService (FeatureInfoFormat*, Class+)>
<!-- defines the deg service used by the appliction -->
<!ELEMENT DEGService (Class)>
<!-- defines the render service used by the appliction -->
<!ELEMENT RenderService (Class)>
<!-- defines the legendservice used by the appliction -->
<!ELEMENT LegendService (Class+)>
<!-- definies the xslt-stylesheets used for processing the
internal xml-documents containing the result to an
feature info request -->
<!ELEMENT FeatureInfoFormat EMPTY>
<!ATTLIST FeatureInfoFormat
format CDATA #REQUIRED
xsltsource CDATA #REQUIRED
>
<!-- defining name a known datasources of a special filter servive.
using the Config tag a additional configuration file (not based on
this dtd) can be definied for each filter service -->
<!ELEMENT Class (Config?, Layer*, Style*)>
<!ATTLIST Class
```

```
name CDATA #REQUIRED
>
<!-- a layer includes one or more data sources. each data source can
be limited to a range of scales using the minscale and maxscale
attributes. each layer is characterized by a name. the name should
be unique -->
<!ELEMENT Layer (KnownDataSource+)>
<!ATTLIST Layer
name CDATA #REQUIRED
>
<!--
a style element associates a named style to a LegendService class. If
a getLegend request orders a legend symbol for a style this is handle
by the LegendHandler class the style is associated to.
-->
<!ELEMENT Style EMPTY>
<!ATTLIST Style
name CDATA #REQUIRED
>
<!-- defines a data source a file or a database table for example
known by a filterservice -->
<!ELEMENT KnownDataSource (#PCDATA)>
<!ATTLIST KnownDataSource
minscale CDATA #IMPLIED
maxscale CDATA #IMPLIED
>
<!-- defines the source of the configuration file for service -->
<!ELEMENT Config (#PCDATA)>
```

Appendix 1.B. Configuration.xml

```
<?xml version="1.0" encoding="iso-8859-1" standalone="yes"?>
<!--DOCTYPE deegreeConfiguration SYSTEM "configuration.dtd"-->
<deegreeConfiguration>
<!-- location of the capabilities XML-file -->
<Capabilities
resource="file:///d:/java/source/deegree/xml_files/deegree.xml"/>
<!-- classes that will be registered to the Dispatcher as services.
Notice that there can be more then one class for each service
that can be registered. For example you may register a class
as filter service for accessing a oracle spatial database
and another for accessing ESRI shape file etc. -->
<FilterService>
<FeatureInfoFormat format="MIME"
xsltsource="file:///d:/java/source/deegree/xml_files/feature_info_to_htm
l.xsl"/>
<!-- Each filterservice can enable access to one or more data sources --
>
<!-- filter service to get data from a oracle spatial database -->
```

```
<Class name="org.deegree_impl.services.wms.filterservice.OracleFilter2">
<!-- file containing specific elements for configuration
the filter service -->
<Config>file:///d:/java/source/deegree/xml_files/oracle.xml</Config>
<Layer name="SA_FFH_FL">
<KnownDataSource minscale="0"
maxscale="1">ANDREAS.FFH_FL</KnownDataSource>
<KnownDataSource minscale="1"
maxscale="9999">ANDREAS.GP010100</KnownDataSource>
</Layer>
<Layer name="SA_Parks">
<KnownDataSource>ANDREAS.GP010100</KnownDataSource>
</Layer>
<Layer name="Border">
<KnownDataSource>AGIT.AWSWRLDA</KnownDataSource>
</Layer>
<Layer name="River">
<KnownDataSource>AGIT.AWRIV3ML</KnownDataSource>
</Layer>
</Class>
<!-- filter service to get raster data (grid coverages) -->
<Class
name="org.deegree_impl.services.wms.filterservice.GridCoverageFilter">
<Config>file:///d:/java/source/deegree/xml_files/gridcover.xml</Config>
<Layer name="geomis">
<KnownDataSource>geomis</KnownDataSource>
</Layer>
<Layer name="euro">
<KnownDataSource>euro</KnownDataSource>
</Layer>
</Class>
<!-- filter service to get data from a access database -->
<Class name="org.deegree_impl.services.wms.filterservice.PointDBFilter">
<!-- file containing specific elements for configuration
the filter service -->
<Config>file:///d:/java/source/deegree/xml_files/access.xml</Config>
<Layer name="Staedte">
<KnownDataSource>tab_cities</KnownDataSource>
</Layer>
</Class>
<!-- filter service to get data from esri shape files -->
<Class name="org.deegree_impl.services.wms.filterservice.ShapeFilter">
<Layer name="EuroBorder">
<KnownDataSource>D:/java/source/deegree/shape/EURNUTS0</KnownDataSource>
</Layer>
<Layer name="heller">
<KnownDataSource>D:/java/source/deegree/shape/heller</KnownDataSource>
</Layer>
</Class>
```

```
<!-- filter service to get data from other wms compatible servers
realizing a
cascading wms server -->
<Class name="org.deegree_impl.services.wms.filterservice.CascadeFilter">
<Config>file:///d:/java/source/deegree/xml_files/cascade.xml</Config>
<Layer name="GTOPO30:CubeWerx">
<KnownDataSource>http://www.cubewerx.com/demo/cubeserv/cubeserv.cgi?laye
rs=GTOPO30:CubeWerx&amp;styles=default</KnownDataSource>
</Layer>
</Class>
</FilterService
<!-- defining the deg services to register to the dispatcher.
notice that there is possibly more then one deg service. maybe you like
to use different implementations for diffenrent datasources. -->
<DEGService>
<Class
name="org.deegree_impl.services.wms.degservice.DisplayElementGenerator_I
mpl">
<!-- file containing specific elements for configuration
the filter service -->
<Config>file:///d:/java/source/deegree/xml_files/degConfig.xml</Config>
</Class>
</DEGService>
<!-- defining the render services to register to the dispatcher.
notice that there is possibly more then one render service. maybe you
like
to use different implementations for diffenrent datasources. -->
<RenderService>
<Class
name="org.deegree_impl.services.wms.renderservice.Renderer_Impl"/>
</RenderService>
</deegreeConfiguration>
```

Appendix 1.C. Gcdescriptor.dtd

```
<!- d e e g r e e G R I D  C O V E R A G E  D E S C R I P T O R -->
<!--
This dtd defines the structure of the description document for embedding
grid coverage data into a deegree based application.
Last Update: July 27, 2002
-->
<!-- root element -->
<!ELEMENT GCDescriptor (GCLayer*)>
<!-- the GCLayer element describes a Grid Coverage Layer. A GCLayer
contains of one or more Attributes that describes the layer. Since by
definition of OGC a Grid Coverage is a Feature the attributes are part
of features properties. each gclayer contains one or more
ScaledTileCollection -->
<!ELEMENT GCLayer (Preview?,ScaledTileCollection+,Property+) >
<!ATTLIST GCLayer name CDATA #REQUIRED>
```

```
<!ATTLIST GCLayer minx CDATA #REQUIRED>
<!ATTLIST GCLayer miny CDATA #REQUIRED>
<!ATTLIST GCLayer maxx CDATA #REQUIRED>
<!ATTLIST GCLayer maxy CDATA #REQUIRED>
<!-- coordinate system of the layer -->
<!ATTLIST GCLayer cs CDATA #REQUIRED>
<!-- defines a single tile that contains a preview image for
a layer -->
<!ELEMENT Preview (Tile)>
<!-- a ScaledTileCollection describes properties and tiles
for a defined range of map scales -->
<!ELEMENT ScaledTileCollection (Tile+,Property+)>
<!ATTLIST ScaledTileCollection minscale CDATA #REQUIRED>
<!ATTLIST ScaledTileCollection maxscale CDATA #REQUIRED>
<!-- properties describing the GCLayer and each scaled tile. -->
<!ELEMENT Property EMPTY>
<!ATTLIST Property name CDATA #REQUIRED>
<!ATTLIST Property value CDATA #REQUIRED>
<!-- a tile is the basic part of a gclayer. it names the source of one
image being part of the layer and additional required paremeters for it.
the source of the tile should be formated as URL -->
<!ELEMENT Tile (#PCDATA)>
<!ATTLIST Tile name CDATA #REQUIRED>
<!-- upper left and lower right corner of the tile in
units of the coordinate system of the tile -->
<!ATTLIST Tile minx CDATA #REQUIRED>
<!ATTLIST Tile miny CDATA #REQUIRED>
<!ATTLIST Tile maxx CDATA #REQUIRED>
<!ATTLIST Tile maxy CDATA #REQUIRED>
```

Appendix 1.D. Gcdescriptor.xml

```
<?xml version="1.0" encoding="iso-8859-1" standalone="no"?>
<!--DOCTYPE GCDescriptor SYSTEM "gcdescriptor.dtd"-->
<GCDescriptor>
  <!--
defines a layer that can be accessed by the deegree-WMS using its name
"SaschenAnhalt".
  -->
<GCLayer name="SachsenAnhalt" minx="250000" miny="520000" maxx="250000"
maxy="520000" cs="EPSG:31492">
   <!--
defines the image that is used for previewing the layer
   -->
<Preview>
<Tile name="preview" minx="250000" miny="520000" maxx="250000"
maxy="520000">
file:///d:/java/source/deegree/rasterdata/sachsenanhalt.gif
</Tile>
```

```
</Preview>
  <!-
as defined at the GCDescriptor dtd a GCLayer is build from several
scaled tile collections. the term "scaled" targets the validity of the
tile collection. The tiles defined within a tile collection will only be
used if the scale of the requested map is between the min- (incl.) and
maxscale (excl.) value.
  -->
  <!--
defines the sources for the layer for a detailed view
  -->
<ScaledTileCollection minscale="0" maxscale="5000">
    <!--
tiles (images) building the layer for the scale range of this
tile collection
    -->
<Tile name="3834hs" minx="4454323.247" miny="5777898.518"
maxx="4461523.247" maxy="5785211.0176">
file:///d:/java/source/deegree/rasterdata/3834hs.tif
</Tile>
<Tile name="3934hs" minx="4454220.880" miny="5763132.804"
maxx="4465474.48" maxy="5774086.307">
file:///d:/java/source/deegree/rasterdata/3934hs.tif
</Tile>
   <!--
properties of the layer that are specific for the scale range of this
tile collection
   -->
<Property name="level" value="detail"/>
</ScaledTileCollection>
  <!--
defines the sources for the layer for a general view
  -->
<ScaledTileCollection minscale="1" maxscale="9999">

   <!--
tiles (images) building the layer for the scale range of this
tile collection
    -->
<Tile name="sachsenanhalt" minx="250000" miny="520000" maxx="250000"
maxy="520000">
file:///d:/java/source/deegree/rasterdata/sachsenanhalt.gif
</Tile>
   <!--
properties of the layer that are specific for the scale
range of this tile collection
   -->
<Property name="level" value="overview"/>
</ScaledTileCollection>
```

```
  <!--
properties of the layer that are constant for every scale
  -->
<Property name="projekt" value="GISPool"/>
<Property name="Träger" value="DelphiIMM"/>
</GCLayer>
<GCLayer name="euro" minx="-20.9" miny="37" maxx="35.7" maxy="70"
cs="EPSG:4326"
<Preview>
<Tile name="preview" minx="-20.9" miny="37" maxx="35.7" maxy="70">
file:///d:/java/source/deegree/rasterdata/euro.gif
</Tile>
</Preview>
<ScaledTileCollection minscale="0" maxscale="40000">
<Tile name="euro" minx="-20.9" miny="37" maxx="35.7" maxy="70">
file:///d:/java/source/deegree/rasterdata/euro.gif
</Tile>
</ScaledTileCollection>
<Property name="projekt" value="TestIt"/>
</GCLayer>
<GCLayer name="geomis" minx="3271910.0634942907"
miny="5206712.5389424879" maxx="3956222.5634942907"
maxy="6113025.038942487" cs="EPSG:4326">
<Preview>
<Tile name="preview" minx="3271910.0634942907" miny="5206712.5389424879"
maxx="3956222.5634942907" maxy="6113025.038942487" >
nichts
</Tile>
</Preview>
<ScaledTileCollection minscale="400" maxscale="40000000000">
<Tile name="Kachel" minx="3271910.0634942907" miny="5961972.955609154"
maxx="3385962.146827624" maxy="6113025.038942487" >
file:///D:/java/source/deegree/rasterdata/geomis/brd_geomis_level0_0_0.j
pg
</Tile>
<Tile name="Kachel" minx="3271910.0634942907" miny="5810920.872275821"
maxx="3385962.146827624" maxy="5961972.955609154" >
file:///D:/java/source/dreegree/rasterdata/geomis/brd_geomis_level0_0_1.
jpg
</Tile>
<Tile name="Kachel" minx="3271910.0634942907" miny="5659868.788942488"
maxx="3385962.146827624" maxy="5810920.872275821" >
file:///D:/java/source/dreegree/rasterdata/geomis/brd_geomis_level0_0_2.
jpg
</Tile>
<Tile name="Kachel" minx="3271910.0634942907" miny="5508816.705609155"
maxx="3385962.146827624" maxy="5659868.788942488" >
file:///D:/java/source/dreegree/rasterdata/geomis/brd_geomis_level0_0_3.
jpg
</Tile>
```

```
<Tile name="Kachel" minx="3271910.0634942907" miny="5357764.622275821"
maxx="3385962.146827624" maxy="5508816.705609154" >
file:///D:/java/source/dreegree/rasterdata/geomis/brd_geomis_level0_0_4.
jpg
</Tile>
</ScaledTileCollection>
<Property name="projekt" value="GeoMis Bund"/>
</GCLayer>
</GCDescriptor>
```

Appendix 1.E. jld.dtd

```
<!- d e e g r e e L A Y E R  D E S C R I P T O R -->
<!--
The deegreeLayerDescriptor is a sequence of styled layers, represented
at the first level by NamedLayer elements.
-->
<!ELEMENT dreegreeLayerDescriptor (NamedStyleCollection*) >
<!-- L A Y E R S  A N D  S T Y L E S -->
<!--
NamedLayer: a NamedLayer uses the 'name' attribute to identify a layer
known to the WMS and can contain zero or more NamedStyles. In the
absence of any styles the default style for the layer is used.
-->
<!ELEMENT NamedStyleCollection (NamedStyle+) >
<!ATTLIST NamedStyleCollection name CDATA #REQUIRED >
<!ENTITY % Symbols "(
PolygonSymbol |
LineStringSymbol |
PointSymbol |
TextSymbol |
ScaledPolygonSymbol |
ScaledLineStringSymbol |
ScaledPointSymbol |
ScaledTextSymbol )" >
<!--
NamedStyle: a NamedStyle uses the 'name' attribute to identify
a style. A NamedStyle contains one or more Symbols of indentic
types.
-->
<!ELEMENT NamedStyle ((%Symbols;)+) >
<!ATTLIST NamedStyle name CDATA #REQUIRED >
<!ATTLIST NamedStyle minscale CDATA #IMPLIED >
<!ATTLIST NamedStyle maxscale CDATA #IMPLIED >
<!-- S Y M B O L S -->
<!-- Polygon Symbol -->
<!ELEMENT PolygonSymbol (
Geometry,
FillColor?,
BackgroundColor?,
FillOpacity?,
BackgroundOpacity?,
FillDashMatrix?,
StrokeColor?,
StrokeOpacity?,
StrokeWidth?,
StrokeDashArray? ) >
<!-- Line String Symbol -->
<!ELEMENT LineStringSymbol (
Geometry,
StrokeColor?,
StrokeOpacity?,
StrokeWidth?,
StrokeLineJoin?,
StrokeLineCap?,
StrokeDashArray? ) >
<!-- Point Symbol -->
<!ELEMENT PointSymbol (
Geometry,
FillColor?,
FillOpacity?,
StrokeColor?,
StrokeOpacity?,
StrokeWidth?,
Mark?,
MarkScale?,
MarkOrientation? )>
<!-- Text Symbol
the text symbol definition is slightly different than the OGC SLD
definition.
the OGC SLD ohter than deegree doesn't know StrokeColor resp.
StrokeOpacity of a
text; it definies the text color and its opacity using the
fillcolor/opacity tag.
deegree instead uses the fill color to draw the text background and the
stroke
color to draw the text, each with its own opacity.
-->
<!ELEMENT TextSymbol (
Geometry,
Label,
TextPosition?,
FontFamily?,
FillColor?,
FillOpacity?,
FontSize?,
FontStyle?,
StrokeColor?,
StrokeOpacity? )>
<!-- Scaled Polygon Symbol -->
<!ELEMENT ScaledPolygonSymbol (
Geometry,
ScaleAttribute,
LimitedSymbol+) >
<!-- Scaled Line String Symbol -->
<!ELEMENT ScaledLineStringSymbol (
Geometry,
ScaleAttribute,
LimitedSymbol+) >
<!-- Scaled Point Symbol -->
<!ELEMENT ScaledPointSymbol (
Geometry,
ScaleAttribute,
LimitedSymbol+) >
<!-- Scaled Text Symbol -->
<!ELEMENT ScaledTextSymbol (
Geometry,
ScaleAttribute,
LimitedSymbol+) >

<!-- Grid Coverage Symbol -->
<!ELEMENT GridCoverage (
Opacity?,
ColorManipulation?
)>
<!-- P A R A M E T E R S -->
<!--
Color parameters, currently colors are RGB encoded using two hex values
per channel and prefixed with a hash (#). For example full red is
#ff0000.
-->
<!ELEMENT FillColor (#PCDATA)>
<!ELEMENT StrokeColor (#PCDATA)>
<!--
Opacity parameters, currently opacity is encoded as a double with 0.0
representing transparent and 1.0 representing completely opaque.
-->
<!ELEMENT FillOpacity (#PCDATA)>
<!ELEMENT StrokeOpacity (#PCDATA)>
<!--
this tag indicates the position in relation to the geometry point where
to draw the text
-->
<!ELEMENT TextPosition
(CENTER |
LEFT_CENTER |
RIGHT_CENTER |
TOP_CENTER |
BOTTOM_CENTER |
LEFT_TOP |
RIGHT_TOP |
LEFT_BOTTOM |
RIGHT_BOTTOM |
Userdefined )>
<!ELEMENT CENTER EMPTY>
<!ELEMENT LEFT_CENTER EMPTY>
<!ELEMENT RIGHT_CENTER EMPTY>
<!ELEMENT TOP_CENTER EMPTY>
<!ELEMENT BOTTOM_CENTER EMPTY>
<!ELEMENT LEFT_TOP EMPTY>
<!ELEMENT RIGHT_TOP EMPTY>
<!ELEMENT LEFT_BOTTOM EMPTY>
<!ELEMENT RIGHT_BOTTOM EMPTY>
<!ELEMENT Userdefined EMPTY>
<!ATTLIST Userdefined dX CDATA #REQUIRED >
<!ATTLIST Userdefined dY CDATA #REQUIRED >
<!ELEMENT FontFamily (#PCDATA)>
<!ELEMENT StrokeWidth (#PCDATA)>
<!ELEMENT FontSize (#PCDATA)>
<!ELEMENT MarkScale (#PCDATA)>
<!ELEMENT MarkOrientation (#PCDATA)>
<!ELEMENT StrokeDashArray (#PCDATA)>
<!ELEMENT FillDashMatrix (#PCDATA | PatternFile)*>
<!--
instead of a FillDashMatrix coded as sequence of '0' and '1' the
FillDashMatrix-Element references a graphic file that contains a pattern
-->
<!ELEMENT PatternFile EMPTY>
<!ATTLIST PatternFile filename CDATA #REQUIRED>
<!ELEMENT StrokeLineJoin (MITRE | ROUND | BEVEL)>
<!ELEMENT StrokeLineCap (BUTT | ROUND | SQUARE)>
<!ELEMENT MITRE EMPTY>
<!ELEMENT ROUND EMPTY>
<!ELEMENT BEVEL EMPTY>
<!ELEMENT BUTT EMPTY>
<!ELEMENT SQUARE EMPTY>
<!ELEMENT FontStyle (NORMAL | ITALIC | OBLIQUE)>
<!ELEMENT NORMAL EMPTY>
<!ELEMENT ITALIC EMPTY>
<!ELEMENT OBLIQUE EMPTY>
<!-- The mark parameter identifies a mark by name or code. -->
<!ELEMENT Mark (#PCDATA | SymbolFile)*>
<!--
instead of a symbol the Mark-Element references a graphic file that
```

```
contains the symbol
-->
<!ELEMENT SymbolFile EMPTY>
<!ATTLIST SymbolFile filename CDATA #REQUIRED>
<!--
Symbols containing a Geometry tag. The Geometry tag specifies which
property of which feature of which feature type defines the geographic
position of a symbol. For that a Geometry tag can contain one
FetchFeatureType tag and one or zero FetchFeature resp.
FetchFeatureProperty tag.
This offers several opportunities. At more than one feature type within
a layer can be used. Second the specialization of portrayal rules.
For example you can define a style for all features thats tpye is
"myFType".
Than you can specialize the portayal rules for a serveral named features
thats type is "myFType".
At third you can create complex portrayal realiations by combining two
or
more rules for drawing a geometry. For example you can define to draw
each point which feature type is "myFType" with a large square.
Additionaly
each single point (feature) got its own portrayal rule, so its drawn
twice.
It results a map that shows each point as a large square containing an
individual additional symbol.
 -->
<!--
Parameter to provide the geometry to be symbolized.
It is most commonly 'fetched' from a named property on the feature.
-->
<!ELEMENT Geometry
(FetchFeatureType,FetchFeature?,FetchFeatureProperty?) >
<!--
Parameter to provide label content. When label content is 'fetched' from
a feature property, the type of the property is unimportant and the
symbol is expected to provide a text version of the property whatever
its type.
-->
<!ELEMENT Label (#PCDATA | FetchFeatureProperty)*>
<!--
Elements indicating that the 'named' property should be fetched
from the feature type / feauter being symbolized.
-->
<!ELEMENT FetchFeatureType EMPTY>
<!ATTLIST FetchFeatureType name CDATA #REQUIRED>
<!ELEMENT FetchFeature EMPTY>
<!ATTLIST FetchFeature name CDATA #REQUIRED>
<!ELEMENT FetchFeatureProperty EMPTY>
<!ATTLIST FetchFeatureProperty name CDATA #REQUIRED>
```

```
<!--
defines a symbol that's validity is limited by the values
of the min and max attributes
-->
<!ELEMENT LimitedSymbol (PointSymbol+ | LinestringSymbol+ |
PolygonSymbol+ | TextSymbol+)>
<!ATTLIST LimitedSymbol min CDATA #REQUIRED>
<!ATTLIST LimitedSymbol max CDATA #REQUIRED>
<!--
definies the property thats value is used to define the LimitedSymbol
used for rendering
-->
<!ELEMENT ScaleAttribute (#PCDATA | FetchFeatureProperty )*>
<!-- definies the opacity of a grid coverage been part of the produced
map. currently opacity is encoded as a double with 0.0 representing
transparent and 1.0 representing completely opaque.
-->
<!ELEMENT Opacity (#PCDATA) >
<!-- enables the manipulation of the color composition of a grid
coverage.
it is assumend the colors are represented in a RGB-color model. each
color
components can be modified increasing or decreasing it value. the values
of the Red, Green and Blue elements are interpred as percent to increase
or decrease each pixels color value. for example:
<ColorManipulation>
<Red>-50</Red>
<Green>-50</Green>
<Blue>30</Blue>
<ColorManipulation>
this will lead to an decrease of the red and green component of each
pixel by
50% and and increase of the blue component by 30%
 -->
<!ELEMENT ColorManipulation (Red?, Green?, Blue?)>
<!ELEMENT Red (#PCDATA)>
<!ELEMENT Green (#PCDATA)>
<!ELEMENT Blue (#PCDATA)>
```

Appendix 1.F. deegree.xml layer descriptor

```
<?xml version="1.0" encoding="UTF-8"?>
<!--DOCTYPE deegreeLayerDescriptor SYSTEM "dld.dtd"-->
<!--
A deegreeLayerDescriptor may contain one or more NamedLayers. In order
to
leaf the clarity usually no more than one NamedLayer should not be
defined
```

```
within a DeegreeLayerDescriptor. The idea is to define a set of
DeegreeLayerDescriptors
for each project defined at the Deegree WMS.
Each contains one or more NamedStyles. A Style defines the
portrayal rule(s) for one or more Symbols. A symbol is associated
with an exactly one basic geometry-type. So if a style contains
more than one symbol definition to create complex graphical
realisations of the defines properties it the symbol types
must be equal.
Each symbol is assigned to a featuretype-feature-property using
the Geometry tag. The FetchFeatureType tag is required. The
FetchFeature- and the FeatureFeatureProperty tag are optional. Default
values are FeatureFeatureProperty="geometry" and
FeatureFeature="default".
It is possible to specialize the portrayal rules for defined features
and feature properties. For example all properties of a feature that's
id
equals "XXXX" and which type is point will be painted
not with the default rule for points but with the complex rule of
a NamedStyle that contains a special rule for feature "XXXX".
Last Update: Jan 1, 2001
-->
<deegreeLayerDescriptor>
<!-- this an example layer for the deegree wms server
considered are point, linestring (curve), polygon (surface),
multipoint, multilinestring (multicurve) and multipolygon
(multisurface) geomertries.
   -->
<NamedStyleCollection name="default">
<!-- do not remove or rename because this is the default
protrayal rule for (multi)point features -->
<NamedStyle name="point" minscale="0" maxscale="9999999">
<PointSymbol>
<Geometry>
<FetchFeatureType name="point"/>
<FetchFeature name="default"/>
<FetchFeatureProperty name="geometry"/>
</Geometry>
<FillColor>#00aaff</FillColor>
<StrokeColor>#111111</StrokeColor>
<StrokeWidth>1.0</StrokeWidth>
<Mark>circle</Mark>
<MarkScale>7.0</MarkScale>
</PointSymbol>
</NamedStyle>
<!-- do not remove or rename because this is the default
protrayal rule for (multi)linestring features -->
<NamedStyle name="linestring" minscale="0" maxscale="9999999">
<LineStringSymbol>
```

```
<Geometry>
<FetchFeatureType name="curve"/>
<FetchFeature name="default"/>
<FetchFeatureProperty name="geometry"/>
</Geometry>
<StrokeColor>#000000</StrokeColor>
<StrokeOpacity>0.9</StrokeOpacity>
<StrokeWidth>1.0</StrokeWidth>
<StrokeLineJoin><MITRE/></StrokeLineJoin>
<StrokeLineCap><BUTT/></StrokeLineCap>
<StrokeDashArray>1</StrokeDashArray>
</LineStringSymbol>
</NamedStyle>
<!-- do not remove or rename because this is the default
protrayal rule for (multi)polygon features -->
<NamedStyle name="polygon" minscale="0" maxscale="9999999">
<PolygonSymbol>
<Geometry>
<FetchFeatureType name="surface"/>
<FetchFeature name="default"/>
<FetchFeatureProperty name="geometry"/>
</Geometry>
<FillColor>#ffff00</FillColor>
<BackgroundColor>#0000ff</BackgroundColor>
<FillOpacity>0.5</FillOpacity>
<!--FillDashMatrix>(1,1,1,0,0,0)
(0,1,1,1,0,0)(0,0,1,1,1,0)(0,0,0,1,1,1)
(1,0,0,0,1,1)(1,1,0,0,0,1)</FillDashMatrix-->
<StrokeColor>#222222</StrokeColor>
<StrokeOpacity>1.0</StrokeOpacity>
<StrokeWidth>1.0</StrokeWidth>
<StrokeDashArray>1</StrokeDashArray>
</PolygonSymbol>
</NamedStyle>
</NamedStyleCollection>
<!-- style for border features -->
<NamedStyleCollection name="border">
<NamedStyle name="polygon1" minscale="0" maxscale="1">
<PolygonSymbol>
<Geometry>
<FetchFeatureType name="EuroBorder"/>
<FetchFeature name="default"/>
<FetchFeatureProperty name="geom"/>
</Geometry>
<FillColor>#cccccc</FillColor>
<FillOpacity>0.0</FillOpacity>
<FillDashMatrix>1</FillDashMatrix>
<StrokeColor>#000000</StrokeColor>
```

```
<StrokeOpacity>1.0</StrokeOpacity>
<StrokeWidth>5.0</StrokeWidth>
<StrokeDashArray>1</StrokeDashArray>
</PolygonSymbol>
</NamedStyle>
<NamedStyle name="polygon2" minscale="0" maxscale="1">
<PolygonSymbol>
<Geometry>
<FetchFeatureType name="EuroBorder"/>
<FetchFeature name="default"/>
<FetchFeatureProperty name="geom"/>
</Geometry>
<FillColor>#ff0000</FillColor>
<FillOpacity>0.0</FillOpacity>
<FillDashMatrix>1</FillDashMatrix>
<StrokeColor>#FFFFFF</StrokeColor>
<StrokeOpacity>1.0</StrokeOpacity>
<StrokeWidth>1.0</StrokeWidth>
<StrokeDashArray>1</StrokeDashArray>
</PolygonSymbol>
</NamedStyle>
<NamedStyle name="polygon3" minscale="1" maxscale="10">
<PolygonSymbol>
<Geometry>
<FetchFeatureType name="EuroBorder"/>
<FetchFeature name="default"/>
<FetchFeatureProperty name="geom"/>
</Geometry>
<FillColor>#ff0000</FillColor>
<BackgroundColor>#ffffff</BackgroundColor>
<FillOpacity>1.0</FillOpacity>
<!--FillDashMatrix>(1,1,0,0)(1,1,0,0)(0,0,1,1)(0,0,1,1)</FillDashMatrix-
->
<!--FillDashMatrix>
<PatternFile filename="file:///d:/java/source/deegree/images/info.jpg"/>
</FillDashMatrix-->
<StrokeColor>#000000</StrokeColor>
<StrokeOpacity>1.0</StrokeOpacity>
<StrokeWidth>1.0</StrokeWidth>
</PolygonSymbol>
</NamedStyle>
</NamedStyleCollection>
<!-- style for city features -->
<NamedStyleCollection name="cities
<NamedStyle name="point">
<PointSymbol>
<Geometry>
<FetchFeatureType name="Staedte"/>
```

```
<FetchFeature name="default"/>
<FetchFeatureProperty name="geom"/>
</Geometry>
<FillColor>#ff0000</FillColor>
<StrokeColor>#000000</StrokeColor>
<StrokeWidth>1.0</StrokeWidth>
<Mark>circle</Mark>
<!--SymbolFile filename="file:///d:/temp/hearts0a.gif"/-->
<MarkScale>7.0</MarkScale>
</PointSymbol>
</NamedStyle>
<NamedStyle name="texttop">
<TextSymbol>
<Geometry>
<FetchFeatureType name="Staedte"/>
<FetchFeature name="default"/>
<FetchFeatureProperty name="GEOM"/>
</Geometry>
<Label>
<FetchFeatureProperty name="CITY_NAME"/>
</Label>
<TextPosition><Userdefined dX="0" dY="-10"/></TextPosition>
<FontFamily>arial</FontFamily>
<FillColor>#dddddd</FillColor>
<FillOpacity>1</FillOpacity>
<FontSize>12</FontSize>
<FontStyle><ITALIC/></FontStyle>
<StrokeColor>#000000</StrokeColor>
<StrokeOpacity>1.0</StrokeOpacity>
</TextSymbol>
</NamedStyle>
<!--NamedStyle name="textbottom">
<TextSymbol>
<Geometry>
<FetchFeatureType name="Staedte"/>
<FetchFeature name="default"/>
<FetchFeatureProperty name="GEOM"/>
</Geometry>
<Label>
<FetchFeatureProperty name="POP_CLASS"/>
</Label>
<TextPosition><BOTTOM_CENTER/></TextPosition>
<FontFamily>arial</FontFamily>
<FillColor>#dddddd</FillColor>
<FillOpacity>1</FillOpacity>
<FontSize>11</FontSize>
<FontStyle><ITALIC/></FontStyle>
<StrokeColor>#000000</StrokeColor>
```

```
<StrokeOpacity>1.0</StrokeOpacity>
</TextSymbol>
</NamedStyle-->
</NamedStyleCollection>
<!-- style for water features -->
<NamedStyleCollection name="water"
<NamedStyle name="linestring">
<LineStringSymbol>
<Geometry>
<FetchFeatureType name="River"/>
<FetchFeature name="default"/>
<FetchFeatureProperty name="geom"/>
</Geometry>
<StrokeColor>#0000FF</StrokeColor>
<StrokeOpacity>0.9</StrokeOpacity>
<StrokeWidth>2.0</StrokeWidth>
<StrokeLineJoin><MITRE/></StrokeLineJoin>
<StrokeLineCap><BUTT/></StrokeLineCap>
<StrokeDashArray>1</StrokeDashArray>
</LineStringSymbol>
</NamedStyle>
<NamedStyle name="polygon">
<PolygonSymbol>
<Geometry>
<FetchFeatureType name="River"/>
<FetchFeature name="default"/>
<FetchFeatureProperty name="geom"/>
</Geometry>
<FillColor>#0000FF</FillColor>
<FillOpacity>0.5</FillOpacity>
<FillDashMatrix>1</FillDashMatrix>
<StrokeColor>#222222</StrokeColor>
<StrokeOpacity>1.0</StrokeOpacity>
<StrokeWidth>1.0</StrokeWidth>
<StrokeDashArray>1</StrokeDashArray>
</PolygonSymbol>
</NamedStyle>
</NamedStyleCollection>
<!-- polygon style for Sachen-Anhalt surfaces -->
<NamedStyleCollection name="SAFlaeche"
<NamedStyle name="polygon">
<PolygonSymbol>
<Geometry>
<FetchFeatureType name="Andreas.FFH_FL"/>
<FetchFeature name="default"/>
<FetchFeatureProperty name="GEOM"/>
</Geometry>
<FillColor>#00FF00</FillColor>
```

```
<FillOpacity>0.5</FillOpacity>
<FillDashMatrix>1</FillDashMatrix>
<StrokeColor>#222222</StrokeColor>
<StrokeOpacity>1.0</StrokeOpacity>
<StrokeWidth>1.0</StrokeWidth>
<StrokeDashArray>1</StrokeDashArray>
</PolygonSymbol>
</NamedStyle>
</NamedStyleCollection>
<!-- Label style for Sachen-Anhalt map -->
<NamedStyleCollection name="SALabel"
<NamedStyle name="text">
<TextSymbol>
<Geometry>
<FetchFeatureType name="ANDREAS.GP010100"/>
<FetchFeature name="default"/>
<FetchFeatureProperty name="GEOM"/>
</Geometry>
<Label>
<FetchFeatureProperty name="NA"/>
</Label>
<TextPosition><Userdefined dX="0" dY="0"/></TextPosition>
<FontFamily>arial</FontFamily>
<FillColor>#dddddd</FillColor>
<FillOpacity>0.5</FillOpacity>
<FontSize>10</FontSize>
<FontStyle><ITALIC/></FontStyle>
<StrokeColor>#002200</StrokeColor>
<StrokeOpacity>1.0</StrokeOpacity>
</TextSymbol>
</NamedStyle>
</NamedStyleCollection>
<NamedStyleCollection name="SC">
<NamedStyle name="scaledpolygon">
<ScaledPolygonSymbol>
<Geometry>
<FetchFeatureType name="EuroBorder"/>
<FetchFeature name="default"/>
<FetchFeatureProperty name="geometry"/>
</Geometry>
<ScaleAttribute>
<FetchFeatureProperty name="POPDENKM"/>
</ScaleAttribute>
<LimitedSymbol min="0" max="100">
<PolygonSymbol>
<Geometry>
<FetchFeatureType name="EuroBorder"/>
</Geometry>
```

```
<FillColor>#222222</FillColor>
<FillOpacity>1.0</FillOpacity>
<FillDashMatrix>1</FillDashMatrix>
<StrokeColor>#000000</StrokeColor>
<StrokeOpacity>1.0</StrokeOpacity>
<StrokeWidth>1.0</StrokeWidth>
<StrokeDashArray>1</StrokeDashArray>
</PolygonSymbol>
</LimitedSymbol
<LimitedSymbol min="100" max="200">
<PolygonSymbol>
<Geometry>
<FetchFeatureType name="EuroBorder"/>
</Geometry>
<FillColor>#666666</FillColor>
<FillOpacity>1.0</FillOpacity>
<FillDashMatrix>1</FillDashMatrix>
<StrokeColor>#000000</StrokeColor>
<StrokeOpacity>1.0</StrokeOpacity>
<StrokeWidth>1.0</StrokeWidth>
<StrokeDashArray>1</StrokeDashArray>
</PolygonSymbol>
</LimitedSymbol>
<LimitedSymbol min="200" max="300">
<PolygonSymbol>
<Geometry>
<FetchFeatureType name="EuroBorder"/>
</Geometry>
<FillColor>#999999</FillColor>
<FillOpacity>1.0</FillOpacity>
<FillDashMatrix>1</FillDashMatrix>
<StrokeColor>#000000</StrokeColor>
<StrokeOpacity>1.0</StrokeOpacity>
<StrokeWidth>1.0</StrokeWidth>
<StrokeDashArray>1</StrokeDashArray>
</PolygonSymbol>
</LimitedSymbol
<LimitedSymbol min="300" max="330">
<PolygonSymbol>
<Geometry>
<FetchFeatureType name="EuroBorder"/>
</Geometry>
<FillColor>#ccccc</FillColor>
<FillOpacity>1.0</FillOpacity>
<FillDashMatrix>1</FillDashMatrix>
<StrokeColor>#000000</StrokeColor>
<StrokeOpacity>1.0</StrokeOpacity>
<StrokeWidth>1.0</StrokeWidth>
```

```
<StrokeDashArray>1</StrokeDashArray>
</PolygonSymbol>
</LimitedSymbol>
<LimitedSymbol min="300" max="200000">
<PolygonSymbol>
<Geometry>
<FetchFeatureType name="EuroBorder"/>
</Geometry>
<FillColor>#0000ff</FillColor>
<FillOpacity>1.0</FillOpacity>
<FillDashMatrix>1</FillDashMatrix>
<StrokeColor>#000000</StrokeColor>
<StrokeOpacity>1.0</StrokeOpacity>
<StrokeWidth>1.0</StrokeWidth>
<StrokeDashArray>1</StrokeDashArray>
</PolygonSymbol>
</LimitedSymbol
</ScaledPolygonSymbol>
</NamedStyle>
</NamedStyleCollection>
<NamedStyleCollection name="kulaka_c">
<NamedStyle name="point">
<PointSymbol>
<Geometry>
<FetchFeatureType name="KULAKA_C"/>
<FetchFeature name="default"/>
<FetchFeatureProperty name="geometry"/>
</Geometry>
<FillColor>#ff0000</FillColor>
<StrokeColor>#000000</StrokeColor>
<StrokeWidth>1.0</StrokeWidth>
<Mark>circle</Mark>
<!--SymbolFile filename="file:///d:/temp/hearts0a.gif"/-->
<MarkScale>7.0</MarkScale>
</PointSymbol>
</NamedStyle>
<NamedStyle name="texttop">
<TextSymbol>
<Geometry>
<FetchFeatureType name="KULAKA_C"/>
<FetchFeature name="default"/>
<FetchFeatureProperty name="geometry"/>
</Geometry>
<Label><FetchFeatureProperty name="ID_GEOMETRIE"/></Label>
<TextPosition><Userdefined dX="0" dY="-10"/></TextPosition>
<FontFamily>arial</FontFamily>
<FillColor>#dddddd</FillColor>
<FillOpacity>1</FillOpacity>
```

```
<FontSize>12</FontSize>
<FontStyle><ITALIC/></FontStyle>
<StrokeColor>#000000</StrokeColor>
<StrokeOpacity>1.0</StrokeOpacity>
</TextSymbol>
</NamedStyle>
</NamedStyleCollection>
</deegreeLayerDescriptor>
```

Appendix 1.G. XSLT-script for formatting the results of a FeatureInfo Request

```
<?xml version='1.0'?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
version="1.0">
<xsl:template match="/">
<HTML>
<HEAD>
</HEAD>
<BODY>
<xsl:apply-templates select="/RootElement/Table"/>
</BODY>
</HTML>
</xsl:template>
<xsl:template match="Table">
<xsl:apply-templates select="Row"/>
</xsl:templa
<xsl:template match="Row">
<TABLE border="1">
<TR>
<TD colspan="2"><xsl:value-of select="/RootElement/Table/@name"/></TD>
</TR>
<TR BGCOLOR="#0000ff">
<TH>Name</TH>
<TH>Value</TH>
</TR>
<xsl:apply-templates select="Column"/>
</TABLE>
<BR/>
</xsl:template
<xsl:template match="Column">
<TR BGCOLOR="#CCCCCC">
<TD><xsl:value-of select="@name"/></TD>
<TD><xsl:value-of select="@value"/></TD>
</TR>
</xsl:template>
</xsl:stylesheet>
```

Appendix 2. International Interfaces (Recipe 5): Basic-wms2.py file

```
# basic-wms2.py : A very small WMS implementation
# V2 - removed PIL, trying PBM instead
#
#=======================================================================
=
# LICENSE -- This is the "MIT License"
#
# Copyright (c) 2001 Allan Doyle
#
# Permission is hereby granted, free of charge, to any person obtaining
# a copy of this software and associated documentation files (the
# "Software"), to deal in the Software without restriction, including
# without limitation the rights to use, copy, modify, merge, publish,
# distribute, sublicense, and/or sell copies of the Software, and to
# permit persons to whom the Software is furnished to do so, subject to
# the following conditions:
#
# The above copyright notice and this permission notice shall be
# included in all copies or substantial portions of the Software.
#
# THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
# EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
# MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND
# NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE
# LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION
# OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION
# WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
#=======================================================================
=
#
# Allan Doyle - adoyle@intl-interfaces.com
#
# This WMS works by keeping a 3600x1800 JPEG image which contains a full
# map of the world from -180,-90 to 180,90 and sending chunks of it out
# in response to map requests by clients.
#
# Things are deliberately hardcoded in this WMS to keep it simple.
#
# The image was generated using the CubeWerx WMS demo which you can find
# at http://www.cubewerx.com
#
# Debugging help was provided by Jeff de La Beaujardiere at NASA
```

```
#
#-----------------------------------------------------------------------
-
### The big chunks of functionality come from mod_python and PIL
#
# mod_python is available from www.modpython.org
#
#    It provides the connection from Apache CGI to python code
#
from mod_python import apache
#
# Python imports
#
import sys
import os
import string
# Open the map image once and load it (this may be causing a memory
leak)
map = "/home/apache/www.intl-interfaces.net/htdocs/images/cubeserv-
best.pnm"
# The WMS version that this WMS implements
version = '1.0.0'
# This is the capabilities XML
# Thanks to Jeff de La Beaujardiere of the NASA Digital Earth program
# for a good one that I used as a template
#
capabilities = """<?xml version='1.0' encoding="UTF-8" standalone="no"
?>
<!DOCTYPE WMT_MS_Capabilities SYSTEM
"http://www.digitalearth.gov/wmt/xml/capabilities_1_0_0.dtd"
 [
<!ELEMENT VendorSpecificCapabilities EMPTY>
 ]>
<WMT_MS_Capabilities version="1.0.0" updateSequence="0">
<!-- Service Metadata -->
<Service>
<!-- The WMT-defined name for this type of service -->
<Name>GetMap</Name>
<!-- Human-readable title for pick lists -->
<Title>Basic Map Server</Title>
<!-- Narrative description providing additional information -->
<Abstract>Basic WMS Map Server built as an example for a WMS cookbook
Contact: adoyle@intl-interfaces.com.</Abstract>
<Keywords>Demo WMS Cookbook</Keywords>
<!-- Top-level address of service or service provider. See also
onlineResource attributes of <dcpType> children. -->
<OnlineResource>http://www.intl-
interfaces.net/cookbook/WMS/</OnlineResource>
<!-- Fees or access constraints imposed. -->
```

```
<Fees>none</Fees>
<AccessConstraints>none</AccessConstraints>
</Service>
<Capability>
<Request>
<Map>
<Format>
<PNG />
<JPEG />
<PPM />
<TIFF />
</Format>
<DCPType>
<HTTP>
<Get onlineResource="http://www.intl-interfaces.net/cookbook/WMS/basic-
wms2/basic-wms2.py?" />
</HTTP>
</DCPType>
</Map>
<Capabilities>
<Format>
<WMS_XML />
</Format>
<DCPType>
<HTTP>
<Get onlineResource="http://www.intl-interfaces.net/cookbook/WMS/basic-
wms2/basic-wms2.py?" />
</HTTP>
</DCPType>
</Capabilities>
</Request>
<Exception>
<Format>
<INIMAGE />
<BLANK />
</Format>
</Exception>
<Layer>
<Title>Demo Map Server</Title>
<SRS>EPSG:4326</SRS>
<LatLonBoundingBox minx="-180" miny="-90" maxx="180" maxy="90" />
<Layer queryable="0">
<Name>RELIEF</Name>
<Title>Relief (ETOPO/GTOPO)</Title>
<Abstract>Colored relief map with political boundaries and
coastlines</Abstract>
</Layer>
</Layer>
</Capability>
```

```
</WMT_MS_Capabilities>
"""
##
# Name/value utilities
#
## split_args(args)
#
# Takes a CGI string. Turns it into a list of name/value pairs. Names
with
# no value are given a None value (a python special value). All names
# are converted to upper case since WMS arguments are case insensitive
#
def split_args(args):
"split_args : take a CGI string and return a list with name/value pairs"
  canon_args = {}           # Start an empty list
  if args == None:          # Return the empty list if no arg
return canon_args
  arglist = args.split('&')      # Split into list of name=value string
  for arg in arglist:         # Now split each name=value and
    tmp = arg.split('=')      # turn them into sub-lists
    if len(tmp) == 1:        # with name in the first part
      canon_args[tmp[0]] = None  # and value in the second part
else:
canon_args[tmp[0].upper()] = tmp[1]
return canon_args
## send_html_error(req, s, status)
#
# Returns a text/html response to the client with an error message
# packaged inside. The status is raised as an exception which neatly
# bumps us all the way back to the apache server.
#
def send_html_error(req, s, status):
  req.content_type = 'text/html'   # Set the return Content-Type
  req.send_http_header()       # Send the HTTP return header
  req.write('<p>' + s + '</p>')    # Wrap the message in <p></p>
raise apache.SERVER_RETURN, status # return to apache
## handler(req)
#
# The name of this function is dictated by mod_python. This is the entry
# point to the WMS. It is called by apache with the map request.
# A file in the local directory called .htaccess defines some of this.
# (Or it can be configured into the main apache httpd.conf file)
#
def handler(req):
"handler : called when apache gets a map request URI"
# mod_python stuff
  #
request = req.args         # Provides a string with the CGI
```

```
# arguments in it
req.content_type = 'text/plain'   # Useful if we have to send messages
# back to the client. Later we'll
# override it with image/jpeg
# WMS argument processing
# starts here...

canon_args = split_args(req.args)  # This turns the args into a python
# list
# If there are no arguments in the request, exit. The WMS spec does not
# specify what to return here since technically, unless there's at least
# a 'REQUEST' parameter, it's not a WMS request. For now, let's
# use HTTP_BAD_REQUEST and return a message
  #
if len(canon_args) == 0:
send_html_error(req, 'No parameters found', apache.HTTP_BAD_REQUEST)
# Next look at the REQUEST argument.
# If it's not there, this is also not a WMS request...
request = canon_args.get('REQUEST', None);
if request == None:
send_html_error(req, 'No REQUEST parameter found',
apache.HTTP_BAD_REQUEST)
# Here are the 3 choices. In the Capabilities XML we say that the
# layer is not queryable, so we should not be getting a feature_info
# request. If we do, we can say HTTP_BAD_REQUEST... this is consistent
# with the WMS 1.0.0 spec 6.2.9.4 that says an error response must be
# MIME typed.
if request == 'capabilities':
send_capabilities(req, canon_args)
elif request == 'map':
send_map(req, canon_args)
elif request == 'feature_info':
send_html_error(req, 'REQUEST=%s is not implemented:' %
canon_args['REQUEST'], apache.HTTP_BAD_REQUEST)
else:
send_html_error(req,'REQUEST=%s is not a valid WMS request' %
canon_args['REQUEST'], apache.HTTP_BAD_REQUEST)
## version_cmp(v1, v2)
#
# Compare version strings. Works like strcmp.
# Since versions are dotted strings with 1, 2, or 3 components, we first
# check if the strings are actually equal. If not, then we make sure we
have
# three components to compare by adding trailing '0' elements. Then we
# compare the high-order part, the next part, and the next part.
#
# For this WMS we only need to know if they are equal or not equal. This
# WMS does not do version negotiation.
```

```
#
def version_cmp(v1, v2):
# If they are already equal, great
if v1 == v2: return 0
# turn them into lists
L1 = v1.split('.')
L2 = v2.split('.')
# build up things like '1.0' and '1' into '1.0.0'
if len(L1) == 1: L1.append('0')
if len(L1) == 2: L1.append('0')
if len(L2) == 1: L2.append('0')
if len(L2) == 2: L2.append('0')
# now if they are equal, great
if L1 == L2: return 0
if string.atoi(L1[0]) < string.atoi(L2[0]): return -1
if string.atoi(L1[0]) > string.atoi(L2[0]): return 1
if string.atoi(L1[1]) < string.atoi(L2[1]): return -1
if string.atoi(L1[1]) > string.atoi(L2[1]): return 1
if string.atoi(L1[2]) < string.atoi(L2[2]): return -1
if string.atoi(L1[2]) > string.atoi(L2[2]): return 1
## send_capabilities(req, args)
#
# Simply sets the Content-Type to 'text/xml' and returns the
Capabilities
# string that's included at the top of this file.
#
def send_capabilities(req, args):
# This is where version negotiation would go. We'll ignore it for now
# since we only support one version. If the client tries to version
# negotiate, we'll just send our 1.0.0 capabilities back each time.
# Eventually the client will accept this or go away
req.content_type = 'text/xml'
req.send_http_header()
req.write(capabilities)
raise apache.SERVER_RETURN, apache.OK
## Projections
#
# Currently assume a base world image of 3600x1800 with the whole world
# from -180,-90 to 180,90
#
# No inverse is needed since we don't handle feature_info requests.
#
def LonToPix(lon):
return int ((lon * 10) + 1800 + .5)
def LatToPix(lat):
return int ((-lat * 10) + 900 + .5)
## send_map(req, args)
#
```

```
# Checks to see if all the args that it knows about are present and
correct
# If so, send a map.
# Note: 2001.05.07 adoyle - added .upper() to all references to
#    found['FORMAT'] to improve leniency for people who use lowercase
'png'
#    'jpeg' etc by mistake.
#
def send_map(req, args):
formats = {'JPEG' : '| ppmtojpeg',
'PNG' : '| pnmtopng',
'TIFF' : '| pnmtotiff',
'PPM' : ' '}
# These are the required parameters (WMS 1.0.0 Table 6.3)
required = ['LAYERS', 'STYLES', 'SRS', 'BBOX', 'WIDTH', 'HEIGHT',
'FORMAT']
# These are the optional parameters (WMS 1.0.0 Table 6.3)
optional = ['TRANSPARENT', 'BGCOLOR', 'EXCEPTIONS']
# Loop through the list of required args. If any are missing, return
# an error.
# The ones that we start with are the optional ones, set to the default
found = {'BGCOLOR' : '0xFFFFFF',
'TRANSPARENT' : 'FALSE',
'EXCEPTIONS' : 'INIMAGE'}
for param in required:
found[param] = args.get(param, None);
if found[param] == None:
send_html_error(req, 'No ' + param + ' parameter found',
apache.HTTP_BAD_REQUEST)
for param in optional:
found[param] = args.get(param, found[param]);
# Find the 4 values in the BBOX
bbox = found['BBOX'].split(',')
# Turn the BBOX values into pixel values
for i in (0, 1, 2, 3):
bbox[i] = string.atof(bbox[i]
x0 = LonToPix(bbox[0])
y0 = LatToPix(bbox[1])
x1 = LonToPix(bbox[2])
y1 = LatToPix(bbox[3])
# get the width/height values
width = string.atoi(found['WIDTH'])
height = string.atoi(found['HEIGHT'])
error = 0
# Let's do a little checking
if bbox[0] < -180.0 or bbox[0] > 180.0 \
or bbox[1] < -90.0 or bbox[1] > 90.0 \
or bbox[2] < -180.0 or bbox[2] > 180.0 \
or bbox[3] < -90.0 or bbox[3] > 90.0:
```

```
error = 1
message = "BBOX out of range"
# If there's an error, then we have to decide whether to return
# an INIMAGE error (i.e. write an error message on an image) or
# whether to make a blank image. In both cases, inimage or blank, we

# then need to decide whether we're supposed to do transparency and
# whether the image format supports it (only PNG does). Then we
# have to make sure the result has transparency.
if error and found['EXCEPTIONS'] == 'INIMAGE':
# Build the image with the text
cmd = 'ppmmake \#%s %s %s' % (found['BGCOLOR'][2:], width, height) \
       + ' | ' \
•  'ppmlabel -background \#888888 -colour \#000000' \
•  ' -x 5 -y 20 -text \"%s\"' \
% message + formats[found['FORMAT'].upper()]
# decide whether to make it transparent
if found['FORMAT'].upper() == 'PNG' and found['TRANSPARENT'] == 'TRUE':
cmd = cmd + ' -force -transparent \#%s' % (found['BGCOLOR'][2:])
# If we're supposed to return a blank image, just make an image
# with BGCOLOR as the entire image.
elif error and found['EXCEPTIONS'] == 'BLANK':
# Build the image
cmd = 'ppmmake \#%s %s %s' % (found['BGCOLOR'][2:], width, height) \
•  formats[found['FORMAT'].upper()]
# decide whether to make it transparent
if found['FORMAT'].upper() == 'PNG' and found['TRANSPARENT'] == 'TRUE':
cmd = cmd + ' -force -transparent \#%s' % (found['BGCOLOR'][2:])
# If there was no error, then build the command that will return
# a new map that is a rectangle cut from the old map and the scaled
# into the new dimensions.
else:
cmd = "pnmcut -left %s -bottom %s -right %s -top %s < %s" \
% (x0,y0,x1,y1, map) \
       + ' | ' \
•  "pnmscale -xysize %s %s" % (width, height) \
•  "| pnmsmooth" + formats[found['FORMAT'].upper()]
# If you added a debug=1 (or any debug) parameter to the request
# this bit will return some debugging info as text instead of the
# image. This is not advertised in the capabilities because this
# is not meant to be used by WMS clients.
if args.get('DEBUG', None):
req.send_http_header()
req.write('bbox %s ' % bbox)
req.write('WxH=%sx%s ' % (width, height))
req.write('x0,y0=%s,%s ' % (x0,y0))
req.write('x1,y1=%s,%s\n' % (x1,y1))
req.write(' params %s\n' % found)
req.write(cmd + '\n')
req.write(message)
raise apache.SERVER_RETURN, apache.OK
# This executes the command we built above and gathers the output
# of the command for reading as a file
pipe = os.popen(cmd, 'r')
# Set the return Content-Type to 'image/<FORMAT>'
req.content_type = "image/%s" % found['FORMAT'].lower()
  req.send_http_header()       # Send the header
  req.write(pipe.read())       # Send the image
raise apache.SERVER_RETURN, apache.OK # exit to apache
```