



ERDAS APOLLO

Administrator's Guide

Essentials-SDI Edition

August 2011



Copyright © 2011 ERDAS, Inc.

All rights reserved.

Printed in the United States of America.

The information contained in this document is the exclusive property of ERDAS, Inc. This work is protected under United States copyright law and other international copyright treaties and conventions. No part of this work may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying and recording, or by any information storage or retrieval system, except as expressly permitted in writing by ERDAS, Inc. All requests should be sent to the attention of:

Manager, Technical Documentation
ERDAS, Inc.
5051 Peachtree Corners Circle
Suite 100
Norcross, GA 30092-2500 USA.

The information contained in this document is subject to change without notice.

Government Reserved Rights. MrSID technology incorporated in the Software was developed in part through a project at the Los Alamos National Laboratory, funded by the U.S. Government, managed under contract by the University of California (University), and is under exclusive commercial license to LizardTech, Inc. It is used under license from LizardTech. MrSID is protected by U.S. Patent No. 5,710,835. Foreign patents pending. The U.S. Government and the University have reserved rights in MrSID technology, including without limitation: (a) The U.S. Government has a non-exclusive, nontransferable, irrevocable, paid-up license to practice or have practiced throughout the world, for or on behalf of the United States, inventions covered by U.S. Patent No. 5,710,835 and has other rights under 35 U.S.C. § 200-212 and applicable implementing regulations; (b) If LizardTech's rights in the MrSID Technology terminate during the term of this Agreement, you may continue to use the Software. Any provisions of this license which could reasonably be deemed to do so would then protect the University and/or the U.S. Government; and (c) The University has no obligation to furnish any know-how, technical assistance, or technical data to users of MrSID software and makes no warranty or representation as to the validity of U.S. Patent 5,710,835 nor that the MrSID Software will not infringe any patent or other proprietary right. For further information about these provisions, contact LizardTech, 1008 Western Ave., Suite 200, Seattle, WA 98104.

ERDAS, ERDAS IMAGINE, Stereo Analyst, IMAGINE Essentials, IMAGINE Advantage, IMAGINE, Professional, IMAGINE VirtualGIS, Mapcomposer, Viewfinder, and Imagizer are registered trademarks of ERDAS, Inc.

Other companies and products mentioned herein are trademarks or registered trademarks of their respective owners.

Table of Contents

Table of Contents	iii
List of Tables	xiii
List of Figures	xv
Configuration Overview	1
The Services Framework Architecture	1
Framework Components	2
Scalable J2EE Component	2
ERDAS Servlets	2
Connectors and Providers	2
Databases, Flat Files, and Imagery	3
Configuration Files	3
Basic Configuration	3
Servlet Engine Configuration	4
Actual Servlet Level Configuration	4
Data Level Configuration	5
Geographic Information and Transactional Configuration	5
Additional Configuration Steps	6
Service Configuration	7
Configuration Methodology	7
Data services	8
Provider Concepts	8
Configuring a Provider	9
Steps to configure a Provider	11
Sample providers.fac	11
How to Control the Provider Configuration	11
Catalog service	12
Deployment and Administration of the Server	12
Environment Configuration	12
Database Schema Management	14
Security Configuration	14
Logging Configuration	18
Typical Scenarios	21
Publishing Vector Data in WFS	21
Shapefile Provider on top of a Data Directory	21
Create a Vector Provider on top of Oracle Data	23
Create a Transactional Provider over Oracle	24
Create a PostgreSQL/PostGIS Vector Provider	28
Create an ArcSDE Vector Provider	30
Create a Vector Provider on top of GML Data	32
Create Styles on Vector Data	33
Publishing Images in WMS	35
Raster Images	35

Publishing Raster Data in WCS	35
Simple Coverage Services	35
Mosaic and List Coverage Services	36
ArcSDE-Raster	38
Populate, Browse and Query the Catalog	41
Authentication	41
Publish a service	41
Data Discovery	42
Using the CSW endpoint	44
Assembling Services and Combining Data	44
Pyramid WMS	44
Cascading with an OpenGIS WMS Context	45
Chaining Services	45
Proxying a OpenGIS-compliant WMS	45
Proxying a OpenGIS-compliant WFS	46
SLD Portrayal Service for Features and Coverages	47
Producing Smart Maps	47
WMS by Portraying Features	48
Map Dressing Service	48
Advanced Portrayal	49
Add a Legend	49
Sample WFS Requests with Filters	51
Filter by FeatureID	51
Filter Equal to an Alphanumeric Property	52
Filter Equal with Namespaces	53
Filter on Two Alphanumeric Properties	54
Geometry Filter: Operator BBOX	55
Geometry Filter: Operator Intersects with a Given Polygon	56
Geometry Filter: Operator Beyond a Given Point	58
Filter combining Spatial and Non-Spatial Operators	60
Manage Data and Enhance Services	63
Restrict Data	63
Disable Interfaces	63
Hiding Columns	64
Disable Output Formats	65
Add a Copyright or Watermark	65
Add a CRS to WCS GIO Decoder Framework	66
ERDAS IMAGINE Projection Engine	67
Add an EPSG Code	70
Define a CRS	70
Filter in a GetMap	72
Add User Functions	74
Add a Java class Function	74
Add a Datasource Function	76
Set Up a WFS with GML2 Objects	78
Insert Data into the Provider	79
Curves, Surfaces, Rings	80
Measurements, Units of Measure	81
Temporal Properties and Operators	83

Portrayal Capabilities	87
Data Portrayal	87
Rules and Styles	87
Rules, the Portraying Logic	87
Styles, Definition of the Look and Feel	88
Creating Maps	88
Styles Templates Description	88
Creating Styles	90
Languages	90
Deploying Styles	96
Deployment Structure	97
Using the Map Dressing Service	98
Grid	98
North Arrow	100
Scale Bar	100
Image Border	101
Complete Dressing Example	102
Displaying Statistics in a Map	103
Call	103
Output Information	103
Definitions	104
Portrayal Statistics Output Values	104
Producing KML	105
Limitations	107
Fast 2D Rendering	107
Coverage Portrayal	107
Output Formats	109
Overview	109
Image Outputs	109
Graphic Interlaced Format (GIF)	109
Joint Photographic Experts Group (JPEG)	109
Keyhole Markup Language (KML)	110
Scalable Vector Graphics (SVG)	111
GeoTIFF	112
Portable Network Graphic (PNG)	113
X-BMP	114
WBMP	114
Text Outputs	115
Plain Text Output	115
HTML	115
GeoRSS	116
JSON	117
Data Outputs	117
Shapefiles	117
GML 2/3	118
GeoTIFF	118
JPEG2000, ECW, NITF, DTED	118

ERDAS IMG	119
Coordinate Transformations	121
SRS Concepts	121
Add a Custom SRS	121
Projection System Information	122
Modify epsg.plb	126
Create usersref.xml	128
Integrate usersref.xml	129
Modify coordinate_system_category.xml	130
Rebuild and Deploy the Webapps	130
Test the Custom SRS in the Data Manager	131
Test the Custom SRS in the Web Client	131
.....	133
Usage and Syntax of the SRS/CRS Parameter	133
Administration of ERDAS APOLLO	135
Introduction	135
Types of Administration	135
Servlet-Engine Level Configuration Parameters	135
Servlet-Engine Level Security	136
Servlet-Specific Configuration Parameters (providers fac) ...	137
Parameters in the providers fac File	138
Framework Configuration	138
WMS (map) Servlet	141
WFS (vector) Servlet	141
WCS (coverage) Servlets	142
Checks	142
General Checks	142
Connections	144
Logging	144
Debugging	145
Performance Tuning	147
Introduction	147
Tuning the GetMap Request	148
Tuning the Data Extraction	149
Tune the RDBMS configuration	149
Tuning the Database Indexes	151
Tuning the Native Request	152
Tuning Portrayal	153
Tuning the Raster Data Sources	155
Tuning Parameters and Configuration for WCS GIO Decoders	155
global-processmanager.properties:	157
local-processmanager.properties:	158
Tuning the Execution Environment	159

Conclusions	160
Using Apache Ant to Rebuild the Webapps	161
Deploying WAR Files on Supported Servlet Engines	164
JBoss	164
Jakarta Tomcat	164
ERDAS APOLLO Tools and Viewers	165
Service Tester	165
Customizing Service Tester Templates	166
Data Indexer	167
Image Indexing with the Data Manager	167
Coverage Indexer	167
Shapefile RTree Builder	167
Vector Services Utilities	169
Schema Generator	169
From-SQL Generator	170
WFS Loader	173
Pyramid and Mosaic Builder	178
Pyramid Builder	178
WMS Tiler	179
Catalog Web Interface	181
Log In to the Web Application	181
Searching and Browsing Content	181
Advanced Search	182
Publishing content	182
Testing the CSW endpoint	183
Administration options	184
Specifying the Storage Directories for Metadata, Thumbnails, & Pyramids	185
Changing the Storage Location for Metadata Files	185
Changing the Storage Location for Thumbnail Files	186
Changing the Storage Location for Pyramid Files	187
General Server Configuration	191
Install Properties	191
Hiding Clear Text Passwords in Configuration Files	194
Configuration and Customization	199
Internationalization	199
ERDAS APOLLO Web Client Configuration	204
The ERDAS APOLLO Style Editor	211
Exploring Data	211
Getting started	211
Procedure Setting the Connection Time-out	214
Data Sources	217
Layers	231
Map Navigation	238
Views	244

Styling Data	251
Brief Introduction to Styling	251
Managing Styles	252
Scale Range Management	258
Rules Reference Guide	260
"Uniform" Rule	260
Classifications	275
"Uniform Roads" Rule	289
Known Symbol" Rule	295
Feature Numberer" Rule	298
HTML Report" Rule	300
Variable Markers" Rule	305
Patternner" Rule	308
Symbol Roller" Rule	310
Common Elements	312
FAQ/Troubleshooting	317
FAQ	317
Troubleshooting	320
Rebuilding the Webapps	323
Deploying WAR Files on Supported Servlet Engines	326
JBoss	326
Jakarta Tomcat	326
Detailed Parameters of a Provider	327
Lists of Possible Parameters	327
Parameters Common to All Types of Providers	327
Parameters for the Map Framework	330
Parameters for Vector Providers (WFS Servlet)	335
Parameters for the Coverage Framework	339
Provider Types	344
Connectors	344
WFS - or Vector - Connectors	345
Oracle Connector	347
Oracle JDBC Thin Driver	347
Oracle JDBC OCI Driver	347
Differences between Oracle OCI and Thin Driver	348
Oracle RAC	348
PostgreSQL/PostGIS	349
Shapefiles	351
ArcSDE	351
DBF Files	355
Microsoft	356
ODBC data source	356
MS-Access	357

SQLServer 2008	360
GML and GML-T	361
DGN	362
Proxy WFS	364
Simple Framework	364
WMS - or Raster - Connectors	369
Simple Image	369
Image Collection	369
Multiple Images	370
Proxy WMS	370
Map Dressing	371
Pyramid Provider	371
Portray Provider	374
ArcSDE-Raster Provider	376
Context Provider	378
Oracle 10g GeoRaster Provider	382
WCS - or Coverage - Connectors	385
Simple Coverage	385
Indexed Coverages	386
Multi Simple Coverages	389
Hierarchical Coverages	391
Oracle 10g GeoRaster Coverages	392
HDF-EOS Coverages	395
Pyramid Provider	397
GML Application Schema and Mapping to Databases	399
Introduction	399
Key Concepts	400
Application Schema	400
GML Application Schema	401
Feature and Feature Type	402
Mapping	402
Configuration Overview	402
Feature Schema Configuration	404
GML Application Schema Construction	405
The Steps to Construct the Feature Type Schema	405
Feature Mapping	407
Mapping Concepts	407
Mapping Methodology	408
Mapping Tags Description	410
Explicit Mapping Definition Steps	411
SQL Mapping Definition Steps	413
Automatic Mapping Definition Steps	415
Relational (Explicit) Mapping Definition Steps	417
Mapping of Enumerations	429
How to Control Mapping Correctness	434
Moving to GML3	434
ERDAS APOLLO support of GML3	434
GML3 Concepts and Schemas	435
Setting Up a ERDAS WFS Serving GML3	435
Migrating a GML2 WFS to GML3	436

Setting Up a ERDAS WFS Serving GML-SF (Simple Feature)	437
Feature Mapping Tags	438
Mapping Section <MAPPING>	438
Metadata Section <INFO>	444
Capabilities Feature Type Section: <EXPORT>	450
Collection Section: <COLLECTION>	450
Options Section: <OPTION>	451
User Functions Section: <UserFunction>	455
Units Definition Section: <UnitDefinition>	455
Units Association Section: <UnitAssociation>	456
WMS Layer Hierarchy Section: <WMS>	456
Coverage and Image Servers	458
Image Server Concepts	458
Image Provider Types	458
Configuring Individual Coverages/Images	459
Configuring a Mosaic or a list of Coverages/Images	461
Image Layers Index File	461
The Image Data Model	462
The HDR File Organization	463
Layout	466
The World Coordinate File Organization	467
The Color File Organization	470
Header Files Summary Table	470
USGS Metadata	471
Limitations and Constraints	471
Imagery Connectors	472
The GDAL Tool	472
GDAL Installation Notes	476
GDAL Configuration	477
Very Large Coverage Manager	481
Very Large Coverage Management Description	481
Very Large Image Management	484
Temporary Files Can Be Very Big	489
Configuration	489
Limitations	490
Issues	490
Examples	490
Advanced Configuration	492
Metadata URL	492
Templates	492
Storage	493
Metadata Configuration in the WMS and WCS Servlets	493
Metadata Configuration in the WFS Servlet	494
Legend URL	495

The Map Generation Transformer	495
Introduction	495
Using MapGen	496
MapGen Tags and Attributes	498
The <MapGen> Tag	498
Feature Properties (Re)definition	498
scaleMin and scaleMax	500
Filter - The "Where" Tag	500
The "Last" Tag	501
Warning: MapGen and Portrayal Rules	503
Scale Dependent Table	503
Data filtering	503
Advanced Security	504
Coarse-Grained Security	505
Basic ERDAS APOLLO Security	507
Fine-Grained Security	508
Security at the Data Source Level	516
Login Credential Map Example	518
Oracle Proxy Session Example	522
Masking	524
SRS Configuration Parameters	531
Structure	531
STORAGE	532
OPTION	532
INCLUDE	533
Object Definition	533
Object Sharing	534
Unit Definition	534
Spheroid Definition	535
Meridian Definition	535
Datum Definition	536
Geographic System Definition	537
Projected System Definition	537
Other IDs	537
Namespaces	538
Projection Definition	538
Coordinate System Definition	541
Structure of the ESRI Mapping File	542
Installing an Optional Spatial Transformation.	543
ERDAS IMAGINE Projection System Configuration	544
Spheroids and Datums	544
Parametric Datums	544
Surface Datums	545
ERDAS IMAGINE Projection Configuration Files	546
mapprojections.dat	546
epsg.plb	546
spheroid.tab	546
sptable.tab	547
units.dat	547

Understanding Datums, Spheroids, and Projections	548
Seven-Parameter Ellipsoidal Transformation	550
Spheroid Example	551
Surface Datum Types	551

List of Tables

Table 1: Base Configuration Levels	3
Table 2: Projection Entry Translation Table	68
Table 3: Supported SLD Tags	91
Table 4: NeedStat Output Meaning	104
Table 5: Framework Configuration Elements	138
Table 6: Debug Request Parameters	143
Table 7: rds.properties Configuration elements	156
Table 8: global-processmanager.properties Configuration elements	157
Table 9: local-processmanager.properties Configuration elements	158
Table 10: Keywords Operators for Advanced Searches	181
Table 11: Location of Metadata, Pyramid Layer, Thumbnail, and Geoprocess Output Files	185
Table 12: Customizable Parameters in the Install.Properties File	191
Table 13: Graphic Options According to Geometries	257
Table 14: Antialiasing Options	262
Table 15: Uniform - Only On Labels options	264
Table 16: Symbols	265
Table 17: Alignment options	268
Table 18: Target Layer options	269
Table 19: Stroke Width Units	271
Table 20: End Cap Parameters	271
Table 21: Join Parameters	272
Table 22: Dashing Patterns	272
Table 23: Uniform - Sample Styles	275
Table 24: Discrete Classification - Data Types	278
Table 25: Class Populator - Key Ordering	283
Table 26: Range Classification Types	288
Table 27: Description of the Map	289
Table 28: Placement Options	292
Table 29: Uniform Roads - Detail on the Sample Styles	295
Table 30: Known Symbol Shapes	297
Table 31: Known Symbol - Sample Styles	298
Table 32: Symbols	307
Table 33: Parameters Applying to All Providers	328
Table 34: Parameters Applying to Map Providers	330
Table 35: Parameters Applying to Vector Data Providers	335
Table 36: Parameters Applying to Coverage Providers	339
Table 37: Types of Connectors	344
Table 38: WFS Providers Implementation Level	346
Table 39: The LOAD_MODULE Function	366
Table 40: The UPPER Function	366
Table 41: The LOAD_MODULE Function	367
Table 42: The LOAD_FILE Function	367
Table 43: The Mapping Tag	438
Table 44: The SQL Tag	438
Table 45: Sub-Elements of the SQL Tag	439
Table 46: Sub-elements of the Element tag	443

Table 47: The Info Tag	444
Table 48: Sub-Elements of the Info Tag	444
Table 49: The Export Tag	450
Table 50: Sub-Elements of the Export Tag	450
Table 51: The Collection Tag	450
Table 52: The Option Tag	451
Table 53: The UserFunction Tag	455
Table 54: Sub-Elements of the UserFunction Tag	455
Table 55: The UnitDefinition Tag	456
Table 56: The UnitAssociation Tag	456
Table 57: The WMS Tag	457
Table 58: Sub-Elements of the WMS Tag	457
Table 59: Parameter Names and Descriptions	465
Table 60: Layout Table	466
Table 61: HDR File Tags	469
Table 62: Color File Format Table	470
Table 63: Header Files Table	471
Table 64:	472
Table 65: GDAL-based Source Formats by Platform	472
Table 66: Geographic Credentials	510
Table 67: Login Credential Map	517
Table 68: The masking parameters	528
Table 69: Types of mask	530
Table 70: SRS ConfigurationTags	531

List of Figures

Figure 1: Services Framework Architecture	1
Figure 2: Providers.fac Content	9
Figure 3: A BBOX Filter Request	56
Figure 4: A Filter to Intersect with a Polygon	58
Figure 5: A Filter to not be Beyond a Point	60
Figure 6: A Filter to Cross a LineString	62
Figure 7: Projection Entry Diagram	68
Figure 8: Map Dressing Output	103
Figure 9: The Projection Identifiers in MapProjections.Dat	123
Figure 10: The Projection Parameters in MapProjections.dat	124
Figure 11: Example of an Entry in the File epsg.plb	127
Figure 12: The GetMap stream with an Oracle source	147
Figure 13: The GetMap optimizations with an Oracle source	160
Figure 14: Service Tester applet	165
Figure 15: RTree Structure	168
Figure 16: Advanced Search	182
Figure 17: Advanced Operations	183
Figure 18: CSW Panel	183
Figure 19: ERDAS APOLLO Style Editor Main Window	212
Figure 20: Preferences item in the Tools menu	213
Figure 21: Preferences Window.	214
Figure 22: The File Menu	215
Figure 23: Open Project	216
Figure 24: Open Recent Project	216
Figure 25: Data Menu	217
Figure 26: Add Data Source	218
Figure 27: Attach a Newap Feature Server - Step 2	219
Figure 28: Attach a New Feature Server - Step 2	220
Figure 29: Add Shapefiles	221
Figure 30: Add Map Server	222
Figure 31: Attach a New Map Server	222
Figure 32: Attach a Georeferenced Image	223
Figure 33: Coverage Source	224
Figure 34: Portrayal Service URL	225
Figure 35: Secure Connection Window	226
Figure 36: Import Context	227
Figure 37: Data Source Panel	228
Figure 38: Data Source Properties Item	228
Figure 39: Data Source Properties Window	229
Figure 40: Remove Data Source Option.	230
Figure 41: Add Data Menu	230
Figure 42: History List options.	231
Figure 43: Layers Panel	232
Figure 44: Add Layer	232

Figure 45: Layer Properties	233
Figure 46: Remove Layer	233
Figure 47: Ordering Layers	234
Figure 48: Layer Properties Menu Item	235
Figure 49: Max Count in Layer Properties Window	235
Figure 50: Use Box in Layer Properties Window	236
Figure 51: Additional Parameter New Entry Window	237
Figure 52: Layer Statistics with boston_shape	238
Figure 53: Change Scale	239
Figure 54: Fit To Layer	239
Figure 55: Envelope Panel	240
Figure 56: Envelope Menu	240
Figure 57: Overview Panel	241
Figure 58: The Geometry Editor	242
Figure 59: Feature Info	243
Figure 60: List of Features	243
Figure 61: Gazetteer	244
Figure 62: Views	245
Figure 63: Create a New View - Method 1	246
Figure 64: Create a New View - Method 2	246
Figure 65: View Properties	247
Figure 66: Enabling Map Dressing	247
Figure 67: View with Map Dressing Enabled	247
Figure 68: View with Transparent Areas	248
Figure 69: Select a Device	249
Figure 70: New Device	249
Figure 71: Configure Device	250
Figure 72: Export Context	250
Figure 73: The ERDAS APOLLO Style Editor Architecture	252
Figure 74: Create Style Menu	253
Figure 75: Geometry Property Selection	253
Figure 76: Styling Rule Selection	254
Figure 77: Name Selection and Validation	254
Figure 78: Style Editing Dialog	255
Figure 79: Scale View	258
Figure 80: Edit Scale Range Window	259
Figure 81: Changing the Scale	260
Figure 82: Point Style Example	261
Figure 83: Uniform - Graphics Panel (Point Mode)	262
Figure 84: Uniform - Marker Panel	263
Figure 85: Uniform - Select Symbol Window	265
Figure 86: Uniform - Label Panel	267
Figure 87: Uniform - Graphic Panel (Line Mode)	270
Figure 88: Uniform - Graphic Panel (Polygon Mode)	273
Figure 89: Uniform - Label Panel	274
Figure 90: Uniform - Sample Styles	275

Figure 91: Discrete Classification - Classification Panel	277
Figure 92: Discrete Classification - Opacity	279
Figure 93: Discrete Classification - Styles Table	279
Figure 94: Discrete Classification - Styles Table	280
Figure 95: Classes Populator - General Panel	281
Figure 96: Classes Populator - Advanced Panel	282
Figure 97: Range Classification - Classification Panel	284
Figure 98: Classes Populator - General Panel	286
Figure 99: Classes Populator - Advanced Panel	287
Figure 100: Classifications - Sample Result	288
Figure 101: Uniform Roads - Graphic Panel	290
Figure 102: Uniform Roads - Label Panel	291
Figure 103: Uniform Roads - Graphic Panel	293
Figure 104: Uniform Roads - Select Symbol	294
Figure 105: Uniform Roads - Sample Styles	295
Figure 106: Known Symbol - Graphic Panel	296
Figure 107: Known Symbol - Symbol Panel	297
Figure 108: Known Symbol - Sample Styles	298
Figure 109: Feature Numberer - Marker Panel	299
Figure 110: Feature Numberer - Numbering Panel	300
Figure 111: HTML Report - Global Report Panel	301
Figure 112: HTML Report - Feature Fragment Panel	302
Figure 113: HTML Report- Header Result	304
Figure 114: HTML Report- Footer Result	305
Figure 115: Variable Markers - Marker Panel	306
Figure 116: Variable Markers - Sample Style	308
Figure 117: Patterner - Pattern Panel	309
Figure 118: Symbol Roller - Symbol List	310
Figure 119: Symbol Roller - Entry Editor	311
Figure 120: The Color Chooser - Swatches Panel	312
Figure 121: The Color Chooser - HSB Panel	313
Figure 122: The Color Chooser - RGB Panel	314
Figure 123: The Color Chooser - Opacity Panel	315
Figure 124: The Font Selector	316
Figure 125: Internal Data Model versus Exposed Feature Types	400
Figure 126: GML Schema Structure	402
Figure 127: WFS Configuration	404
Figure 128: WFS Mapping	408
Figure 129: Road-Lane UML Diagram	417
Figure 130: Road-Lane Relational Diagram	417
Figure 131: Parcel-Person UML Diagram	423
Figure 132: Parcel-Person Relational Diagram	424
Figure 133: Brussels Orthophotoplan	459
Figure 134: A Set of Images on Brussels	460
Figure 135: Example of a Data Model Organization	463
Figure 136: Bil Bands	464

Figure 137: BIL Layout	467
Figure 138: World File	468
Figure 139: Raster CS Type	470
Figure 140: Tiles order in Landsat hdf4 dataset	476
Figure 141: WCS Process Chain	482
Figure 142: Very Large Coverage Processing	483
Figure 143: WCS/CPS processing/rendering chain.	485
Figure 144: Without rendering control	486
Figure 145: With rendering control	486
Figure 146: Very Large Image Management process	487

Configuration Overview

This chapter gives an overall view of the configuration in the ERDAS APOLLO Server components.

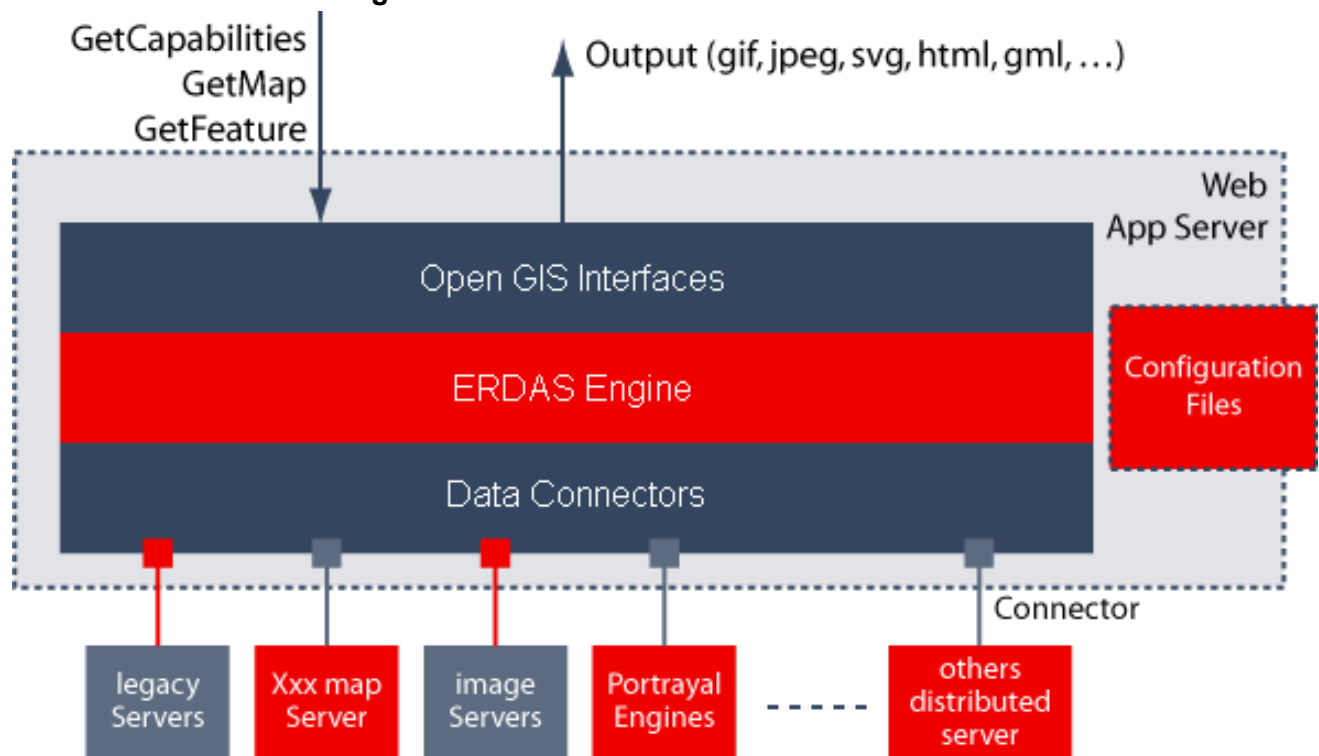
The Services Framework Architecture

The Services Framework Architecture shows how data is manipulated through the application programming interface (API). There are three layers:

- The Open GIS Interfaces
- ERDAS Engine
- Data Connectors

In addition, there are the configuration files that are accessed by the Data Connectors layer.

Figure 1: Services Framework Architecture



The Open GIS Interfaces layer interacts with the client applications and services. At this level, user requests are translated into internal statements and the results are converted into documents or images.

The middle layer, the ERDAS Engine, is the processing layer that contains the ERDAS servlets. ERDAS servlets perform a variety of functions, such as data conversions, calculations and map projections. The servlet's behavior can be fine tuned by modifying a set of configuration files.

The Data Connections layer is the lowest layer. It connects the data server or source to the servlets. It uses the configuration files that provide information permitting access to the data and making the data connectors visible as services.

Framework Components

The components of the framework are:

- Scalable J2EE Components: The J2EE container or Web application server that manages the service, e.g., Tomcat, JBoss and WebLogic
- ERDAS Servlets
- Connectors and Providers: Data accessed via the servlets, i.e., Oracle 9i/10g/11g, PostgreSQL/PostGIS, Shapefiles and Map, Coverage or DEM Sources
- Databases, Flat Files and Imagery: Data connectors - a type of "plug-in" that understands database files accessed through WFS as well as imagery sources published as WMS, WCS or WTS
- Configuration Files: Setting up the services

Scalable J2EE Component

The ERDAS APOLLO Server components are completely written in Java and use several configuration files allowing easy and rapid configuration. These services are highly scalable and the fully reentrant J2EE components can manage as many threads as these services request. This permits the container to manage the threads with no risk of interference.

ERDAS Servlets

The ERDAS Servlets are middleware that provide the processing necessary to perform a number of functions within the application. These functions are data management and conversion and map projections.

Connectors and Providers

Each service connects to a data source - imagery, database or other. The data source connection is optimized through a specific connector that manages the mapping from the Web Service to the data. A connector that is configured to accomplish the task of mapping is called a "provider".

Databases, Flat Files, and Imagery

The database can be an SQL database engine, another geo-engine, a data file, a raster image or a coverage. It is possible to have multiple Web Services accessing the same data. Each type of data source is configured with a specific connector having its own parameters that best matches the internal connector model with the web service's one.

Beyond the connectors predefined in the product, it is possible to develop new connectors. ERDAS APOLLO Solution Toolkit product, usable on top of ERDAS APOLLO Server, provides a "Simple Framework" API in order to plug a map connector into the WFS servlet. It is primarily intended to make it easy to implement a connector over flat files. It also allows to develop custom coverage data decoders and custom coverage metadata decoders.

Configuration Files

The ERDAS Web Services configuration files are used to control the type of services, connection pools, or operating system resources. These are also used to define the global behavior of the system. Most of the configuration modules rely on XML files as this format is widely used by software vendors and fully managed utilizing Java.

Basic Configuration

ERDAS servlets are middleware that must be configured to a specific environment. Each user environment differs; they will connect to unique data sources and may have different Web Application Servers. Additionally, methods and requirements for displaying geoinformation vary from installation to installation. Even though the configuration of ERDAS products may result in unique profiles for each environment, the procedure to configure services is the same.

Regardless of the servlet type, configuration for ERDAS APOLLO Server components consists of four basic levels:

- Servlet Engine Configuration
- Actual Servlet Level Configuration
- Data Level Configuration
- Geographic Information - Transactional Configuration

Table 1: Base Configuration Levels

	WMS	WFS	WCS
Servlet Engine Level	web.xml	web.xml	web.xml

Table 1: Base Configuration Levels (Continued)

	WMS	WFS	WCS
Servlet Level	Providers.fac, World File	Providers.fac, Feature Mapping and Schema	Providers.fac, index
Data Level	Format, Size	RDBMS, JDBC, index	Format, Size
GI/Transactional Level	SRS	SRS, Portrayal, Transactional	SRS, Portrayal
Additional Configuration	Metadata, Dimension/Filter, LegendURL, Copyright, Performance tuning, GML output	Metadata, Dimension/Filter, LegendURL, Copyright, Performance tuning, GML output	Metadata, Performance tuning, Coverage output

Servlet Engine Configuration

A servlet exists in the context of a Web Application, defined in a single XML file called `web.xml`. Configuration at the Servlet Engine level consists of configuring the content of this `web.xml` file. This file is located in the WEB-INF subdirectory of the Web Application and provides the primary configuration information such as the name and location of the configuration files for each servlet. So, for the ERDAS APOLLO Server installation, `web.xml` files can be found in the WEB-INF directory of each of the ERDAS WAR files, i.e. `erdas-apollo.war` or `apollo-client.war`.

For more information about the `web.xml` file including specific steps for set up, please refer to [Administration of ERDAS APOLLO](#) in this guide.

Actual Servlet Level Configuration

All of ERDAS's servlets are configurable according to the desired functionality. The majority of the configuration steps are at this level. Most of ERDAS's servlets use a factory file, called `providers.fac` which defines the link to data sources and servlet properties. These `providers.fac` files are located in the installation directory, under `config/erdas-apollo/providers/<service_type>`, where `<service_type>` can be `map`, `coverage`, `vector`, or `catalog`. It serves two functions:

- To administer the servlets
- To set up access to data sources

The methodology and step-by-step definition of content in the `providers.fac` file is detailed in [Service Configuration](#).

In some cases, such as the WFS, additional steps need to be performed. Additional steps may include defining feature types and matching of these features types with a data structure. Feature types are defined in a XML Schema file and matching this schema with the corresponding data structure is done in a XML "Mapping" file. They are both referenced in the providers.fac file. There are several different types of mapping that are available depending on the nature of the data. The main options available are:

- SQL Mapping for data directly mapped to feature types
- Explicit Mapping for assigning names to feature types different from data
- Automatic Mapping for data model built from feature types definition

Reference [Feature Mapping Tags](#) for mapping different data types.

Data Level Configuration

Once the servlets are configured, the data will need to be formatted and named to make them accessible by the services. ERDAS APOLLO provides several methods for preparing data sources for use with the servlets. Depending on the type of data and the servlet type, i.e. WMS, WFS or WCS, the following can be configured:

- For Map data in a WMS: Data Format and Size.
- For Feature data in a WFS: Database Connection and Tuning, (e.g. JDBC, RDBMS, index files).
- For Coverage data in a WCS: Data Format, Size and Organization. (Possibly indexed by a WFS or a Catalog).

This level of configuration takes place within the data files themselves. Please refer to [Typical Scenarios](#) and [Provider Types](#) for additional details and exact steps for each data type.

Geographic Information and Transactional Configuration

The Geographic Information (GI) and Transaction Configuration level insures that geodata served by ERDAS components are comprehensible to users. This may mean that all data must adhere to certain GI characteristics, such as spatial reference systems and bounding box extents. For feature data, the look and feel of the data can be configured by defining portrayal rules and styles. This configuration takes place at the level of the servlet through an additional parameter given in the providers.fac file. Refer to [Portrayal Capabilities](#) for further information on the steps to take.

Additional Configuration Steps

Data accessed by the WFS may also be *transactional*. This means that the data can be altered in the database through insert, update and delete statements. The WFS can become transactional by defining additional settings in the "Mapping" XML file. Please refer to the chapter [Typical Scenarios](#) on page 21 and the chapter [Advanced Configuration](#) on page 183 for additional details on transactional services.

After completing the four basic levels of servlet configuration, the geodata are ready for publishing on the Web. However, there may be additional publishing needs and further optimization of data access.

ERDAS APOLLO supports advanced configuration allowing:

- Configuration of ISO-compliant metadata to describe the data for other users.
- Application of filters in requests to narrow down the data for efficient analysis purposes.
- Addition of a legend bitmap reference that supports the display of legend icons on the maps.
- Creation of different types of output, including GML documents and shape files.
- Fine-tuning of GML output to allow filtering, renaming of elements, and to hide private data

Refer to the chapter [Typical Scenarios](#) on page 21 and the chapter [Advanced Configuration](#) on page 183 for additional details and detailed steps for the desired configuration.

Service Configuration

This chapter gives a general explanation on how to manually configure each type of data source. Nevertheless, the simple way to administrate services and data sources is the ERDAS APOLLO Data Manager, which is a graphical tool provided in a separate installer along with the ERDAS APOLLO product (see the "Installing ERDAS APOLLO Data Manager" section of the Installation and Configuration Guide for more details).

Configuration Methodology

Configuring a Web Service, i.e., WMS, WFS, WCS, depends on the type of data and services published. The steps listed below cover common data types.

For *vector data*, data is commonly configured to be presented as features.:

- Define an XML schema definition of the feature types to expose. (See [GML Application Schema and Mapping to Databases](#) on page 399.)
- Establish the feature types relational mapping to the datastore, often a SQL mapping (See [Configuration Overview](#) on page 1.)
- For each service, indicate the connectors to be used in the providers file (see the following sections for instructions for each type of data source).

For *raster and coverage data*, the data must be configured, i.e., world file, colors, format, and declared.

- Put the image or coverage files in a set of directories accessible to the servlet.
- Create and put the corresponding World and/or Metadata files in that same directory.
- For each service, indicate the connectors to be used in the providers file (see the following sections for instructions for each type of imagery data)

For any other type of data (proxy, pyramid, specific), the entry in the providers file is the only mandatory step. The other actions are dependent on the type of connector used.

Presenting vector data as maps requires an additional step - portrayal.

Once the configuration tasks are completed, the servlet is automatically restarted to expose the newly created service.

To check if new providers are well defined and visible, call the servlet with the parameters **request=debug&cmd=getlist**, through a URL that will look like: **http://myhost:80/erdas-apollo/vector/debug?request=debug&cmd=getlist**. If the new provider is well defined, its ID will appear in the list.

To check if the data are correctly published, request an OGC capabilities from the service and check that the returned capabilities document is as expected. In the case of a WFS, calling the "DescribeFeatureType" checks the feature type mapping. An example would be: **http://myhost:80/erdas-apollo/vector/ATLANTA_VECTOR?version=1.0.0&service=WFS&request=DescribeFeatureType&typename=roads**.

Data services

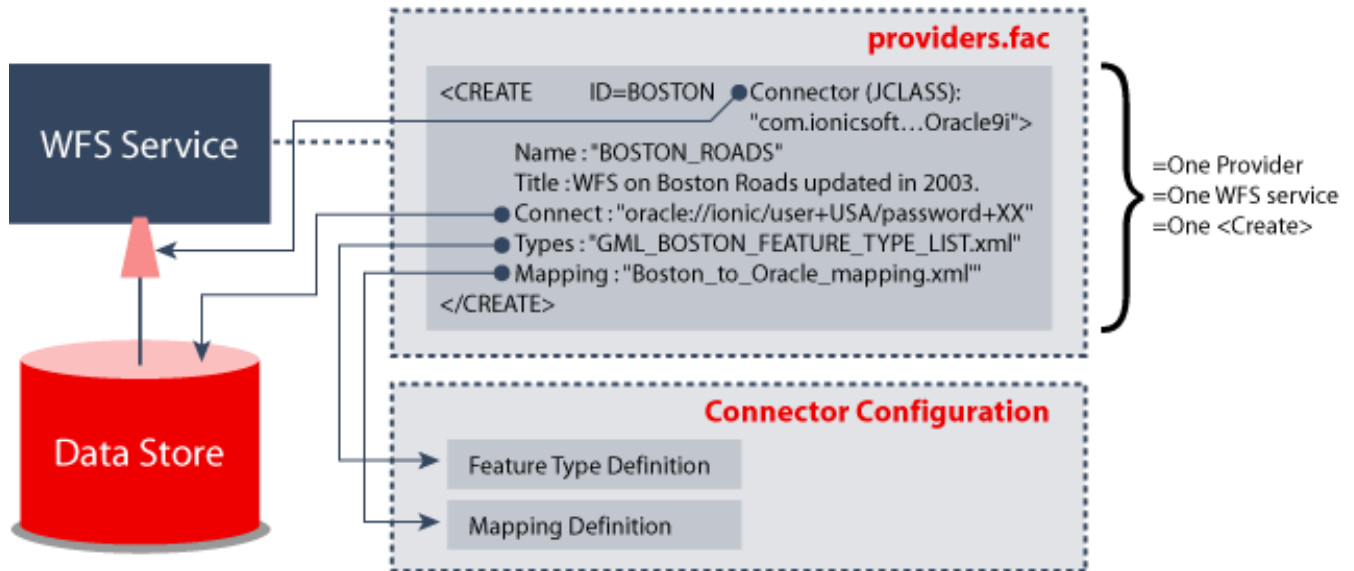
An ERDAS Web service, i.e., WFS, WMS, WCS is configured through a main factory file named providers.fac used by the associated servlet (see [Servlet-Specific Configuration Parameters \(providers fac\)](#)). This XML configuration file is composed of:

- A global configuration section used to configure the framework behavior utilizing the <CONFIGURATION> tag.
- Several WMS/WFS/WCS configuration sections, one per defined service, utilizing multiple <CREATE> tags.

Provider Concepts

A provider describes an instance of a connector. A connector is a Java class that plugs into the servlets to make a link with one type of data such as a large raster image or a shapefile. Multiple instances of a connector can be defined if there are multiple data sources such as multiple images or shapefiles. Each instance is a provider and creates a running Web Service. A section of the "providers.fac" is called <CREATE> and sets how the instance of a given service is linked to a connector and how the data store is accessed.

Figure 2: Providers.fac Content



A provider has the following characteristics:

- One Web Service, WMS, WFS, WCS, etc. is associated with one and only one provider.
- One provider is an instance of a connector.
- One configured connector is a provider.
- One provider is configured in the providers.fac by one tag <CREATE>.

Configuring a Provider



This section describes in detail the physical definition of a provider. Most of the time, it is not necessary to create or edit them manually as the ERDAS APOLLO Data Manager is designed to allow all types of service configurations needed.

The <CREATE> element

The <CREATE> element is used to configure a provider and contains two attributes:

- The ID that uniquely identifies each provider in a framework and

- The JCLASS that indicates the kind of provider class used to connect to the data source.

Each service to be managed is defined and has an associated ID which is referenced by client-side components. This ID will give the name of the service. For example, if the ID is "ATLANTA", then the service could be: "http://www.mysite.com/erdas-apollo/vector/ATLANTA?".

JCLASS indicates the name of the connector's Java class that will be used for this Web service. A list of all the possible connectors is provided in [Provider Types](#) on page 344.

The <PARAM> and <PARAMBLOCK> elements

Each provider is defined by a CREATE element. The Create element has parameters defined in the PARAM and PARAMBLOCK elements. The PARAM and PARAMBLOCK elements have a number of possible names and values. Each provider has a specific set of mandatory and optional PARAM and PARAMBLOCK elements. A "PARAM" element has two attributes: "NAME" and "VALUE", i.e., <PARAM NAME="title" VALUE="Erdas WFS server on ATLANTA"/>. A "PARAMBLOCK" element has only the "NAME" attribute" which maybe optional and embeds one or more <PARAM> and <PARAMBLOCK> sub-elements. The tables in Appendix "Detailed Parameters of a Provider" lists the set of supported PARAM and PARAMBLOCK elements for each type of service.

Not all parameters apply to each type of provider. Some are mandatory and others are optional.

Please refer to [Detailed Parameters of a Provider](#) on page 327 for the complete list of parameters, and to [Provider Types](#) on page 344 for the set of parameters applicable for each type of provider.



URL parameters behavior:

- Relative URLs are relative to the URL of the factory file (providers.fac)
- Object URLs (obj:) are relative to the resource directory of the servlet (i.e., com.ionicsoft.wfs.server.resource for WFS)
- Resource URLs (res:) are relative to the CLASSPATH (i.e., res:///com/erdas/sref/impl/resource/factorysref.xml)
- Others are absolute URLs

Steps to configure a Provider

Sample providers.fac

Always use the ERDAS APOLLO Data Manager to configure a new service provider. Refer to the online help for detailed instructions.

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE FACTORY SYSTEM "factory.dtd" >
<FACTORY>
...
<CREATE ID="ATLANTA_ORA"
  JCLASS="com.erdas.wfs.provider.oracle.OracleProvider">
  <PARAM NAME="title" VALUE="Erdas WFS server on ATLANTA"/>
  <PARAM NAME="connect"
    VALUE="oracle://myhost/user+foo/password+bar/SID+ATLANTA"/>
  <PARAM NAME="types" VALUE="obj:///atlanta_ora.xsd" />
  <PARAM NAME="mapping" VALUE="obj:///atlanta_ora.xml" />
</CREATE>

<CREATE ID="ATLANTA_SHAPE"
  JCLASS="com.erdas.wfs.provider.shapev2.ShapeProvider">
  <PARAM NAME="title" VALUE="Erdas WFS over ATLANTA SHAPE
Files"/>
  <PARAM NAME="path" VALUE="D:/Erdas/data/shapes/atlanta" />
  <PARAM NAME="types" VALUE="obj:///atlanta_shape.xsd" />
  <PARAM NAME="mapping" VALUE="obj:///atlanta_shape.xml" />
</CREATE>

<CONFIGURATION>
  //if framework configuration is also in the same file, it
comes here
  ...
</CONFIGURATION>

</FACTORY>
```

How to Control the Provider Configuration

If there is a syntax error in the providers.fac file, either the service will not start or one or more providers will not be accessible. Invoking the service "debug" pseudo-provider or one of the newly defined provider IDs will provide a direct answer on syntax correctness. Example:
<http://myhost:80/erdas-apollo/vector/debug?request=debug&cmd=getlist>.

Another way to check the syntax of the providers.fac is to use an XML validation tool along with the factory.dtd definition file that defines the allowed tags.

If there is a mistake in the definition of a provider, the effect will vary depending on the mistaken attribute or parameter. As a control, if the service agrees to deliver the provider capabilities document, the information included will give details about the erroneous declarations. If a capabilities document is successfully returned, send a GetMap or a GetFeature request because it will invoke the data server and return maps or data. Even though GetCapabilities, DescribeFeatureType (WFS only) and DescribeCoverage (WCS only) cause the instantiation of the provider and the connection to the data server, they do not send data requests.

Catalog service

Deployment and Administration of the Server

After the ERDAS APOLLO installation process completes successfully, the APOLLO Catalog will be part of the installed web applications.

It is still possible to change the configuration of your APOLLO Catalog afterward. You can easily change the configuration of the used data source (the database), the security definition, or the logging mechanism. The next sections contain the information for doing this.

Environment Configuration

The APOLLO Web Client is part of the global `erdas-apollo.ear` application that was installed in your application server as a result of the install procedure. The `apollo-catalog` is mainly parameterized using a single file, `erdas-apollo.ear/conf/hibernate.properties`.

A build of the application will replace the tokens located in that file with what was provided during the installation. Those values are stored in the `build.properties` which customizes how the build operates. This section mainly discusses how to change some of the tokens without reinstalling the product.

The `hibernate.properties` file contains the following kinds of configuration:

- Database Configuration
- Search Configuration
- Security Configuration

Database-Related Variables

Database-related variables allow you to change the database that the catalog connects to; it can also be used to switch from one RDBMS to another if necessary. The following keys can be changed in the `build.properties` file:

- `hibernate.dialect` defines the database dialect to use. You do not need to change this unless you want to switch from one RDBMS to another. For Oracle, the value is `com.erdas.rsp.hibernate.oracle.OracleDialect` and for PostgreSQL the value is `com.erdas.rsp.hibernate.postgis.PostgisDialect`.
- `hibernate.connection.datasource` defines the J2EE data source used by the catalog. See the description of `apollo-ds.xml` below.
- The data source and related database connection parameters are defined in the `apollo-ds.xml` file, located in at the root of your application server deployment directory.

Search-Related Variables

It is possible to change the storage policy of the indexes used to perform free text searches. This is an advanced use case and ERDAS recommends that you check the official [Hibernate](#) documentation first.

The following properties can be changed directly in the `erdas-apollo.ear/conf/hibernate.properties` file:

- `hibernate.search.default.directory_provider` defines the directory provider managing the index information storage.
- `hibernate.search.default.indexBase` defines the base path where the indexes are stored.



Depending on the provider type used, other variables could be affected or used. Those are explained in the official documentation.

Security-Related Variables

The security is largely explained below, see [Security Configuration](#).

By default, the configuration of the default permissions is defined by the file `erdas-apollo.ear/erdas-apollo.war/WEB-INF/classes/default-permissions-config.xml`. If you want to configure the application so that another configuration file is used instead, you must update the `default-permissions-config.xml` property in the `hibernate.properties` file.

Database Schema Management

The ERDAS APOLLO installation process can automatically generate the required schema for using the ERDAS APOLLO Catalog if you specify during the installation that it should do this. If you have an existing database, you should create a backup before you install a new version of ERDAS APOLLO.

The schema generation script can be invoked on the command line in order to recreate a fresh schema, to update an existing schema, or if `apollo-catalog` has been configured to use another database or schema.

The generator is located in the `tools/schema-generator` directory of the ERDAS APOLLO installation. By default, the `build.properties` contains the database credentials that have been entered at installation time. If a fresh schema needs to be created on another database or schema, you will have to update this file. See [Database-Related Variables](#) for more details.

The management of the schema is handled by an *Apache Ant* build script which provides the following operations:

- *upgrade* (default): upgrades the schema if necessary. If the schema is empty, the necessary tables are created.
- *drop*: removes the `apollo-catalog` tables and their contents.
- *recreate*: is a shortcut that performs the drop operation followed by the create operation.



Make sure to back up your data prior to any upgrade operation.

Any schema management operation will log everything in the `schema-upgrade.log` file located in the same directory.

Security Configuration

To implement the security requirements of the application, the open source framework [Spring Security](#) is used.

The configuration of security is mainly in the `erdas-apollo.ear/erdas-apollo.war/WEB-INF/config/security-config.xml` file. This file defines the basic components of the security of the system. The overall security of the catalog can be set up at different levels:

- Authorizing which methods can be invoked (Method Level Security).
- Authorizing access to individual domain object instances (Object Level Security).

- Authorizing web requests (HTTP Authentication).

Configurations relevant to all of the levels are explained below with examples.

Method-Level Security

Method-based security lets you secure a method which can be executed by only those users who have been granted a particular security role. Different methods of the service bean are configured to be secured. Only certain roles are allowed to execute those methods. This is configured using the method security interceptor.

Method security is enforced using this `MethodSecurityInterceptor`, which secures `MethodInvocations`. Depending on the configuration approach, an interceptor may be specific to a single bean or shared between multiple beans. The interceptor uses a `MethodDefinitionSource` instance to obtain the configuration attributes that apply to a particular method invocation. It is configured in the `security-config.xml` file.

```
<bean id="theMethodSecurityInterceptor"
class="org.springframework.security.intercept.method.aopalliance.MethodSecurityInterceptor">
  <property name="authenticationManager"
ref="authenticationManager" />
  <property name="accessDecisionManager"
ref="accessDecisionManager" />
  <property name="objectDefinitionSource">
    <value>

com.erdas.rsp.babel.service.persistence.PersistenceService.find
*=IS_AUTHENTICATED_ANONYMOUSLY

com.erdas.rsp.babel.service.persistence.PersistenceService.coun
t*=IS_AUTHENTICATED_ANONYMOUSLY

com.erdas.rsp.babel.service.persistence.PersistenceService.pers
ist*=BABEL_PUBLISHER,BABEL_ADMIN

com.erdas.rsp.babel.service.persistence.PersistenceService.dele
te*=BABEL_PUBLISHER,BABEL_ADMIN

com.erdas.rsp.babel.service.persistence.PersistenceService.upda
te*=BABEL_PUBLISHER,BABEL_ADMIN
    </value>
  </property>
</bean>
```

Object Level Security

Object Level security is used to secure a resource by restricting access to only those users who have been granted a particular security role.

This is achieved by securing the domain objects. Domain objects are persisted along with the permissions associated with them. When the object is accessed using a service, the current user's identity and roles are checked and if the principal has READ permission set on the domain object, then the object is returned in the service response.

Generally when a domain object is persisted, either it is persisted with a specific set of permissions or it is persisted with the default permissions. Those default permissions are defined in the `default-permissions-config.xml` file located in the `erdas-apollo.ear/erdas-apollo.war/WEB-INF/classes` directory of the `apollo-catalog` source web application.

This file defines a set of policy and the policy that is active for the whole instance. For instance, if you want to grant all rights to the user who created an object and READ permissions to anyone, you would configure it like this:

```
<Security xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="default-permissions-
  config.xsd">

  <!-- Defines the policy in use -->
  <ActivePolicy name="user"/>

  <Policy name="user">
    <Default>
      <CurrentUser rights="RUD+-"/>
      <Anyone rights="R"/>
    </Default>
  </Policy>
</Security>
```

The syntax of the security file is fairly simple. The active policy must be named as follows:

```
<ActivePolicy name="policy2" />
```

Each policy must be defined. Its definition basically contains a name and a default section that defines the rights that you want to associate to an object created in the system.

```
<Policy name="policy2">
  <Default>
```

```

        <CurrentUser rights="RUD" />
        <CurrentRoles rights="RU" />
        <Role name="role1" rights="RUD" />
    <Default>
</Policy>

```

The rights are simply defined by a letter where R means Read, D means Delete, U means Update, + means Grant Right, and - means Revoke Right.

HTTP Authentication

HTTP authentication and authorization is provided through the use of a web filter, an AuthenticationProvider, and an AuthenticationEntryPoint. The configuration of the web filter, AuthenticationProvider, and AuthenticationEntryPoint are as follows:

- In the `web.xml` file, this application will need a single Spring Security filter in order to use the FilterChainProxy.

```

<filter>
  <filter-name>filterChainProxy</filter-name>
  <filter-class>
    org.springframework.web.filter.DelegatingFilterProxy
  </filter-class>
</filter>
<filter-mapping>
  <filter-name>filterChainProxy</filter-name>
  <url-pattern>/catalog/*</url-pattern>
</filter-mapping>

```

The above declarations will cause every web request to be passed through to the bean called filterChainProxy which will usually be an instance of Spring Security's FilterChainProxy.

- The FilterChainProxy enables web requests to be passed to different filters based on URL patterns. Those delegated filters are managed inside the application context, so they can benefit from dependency injection. The FilterChainProxy bean definition (located in the security-base-config.xml file) looks like the following inside our application context:

```

<bean id="filterChainProxy"
class="org.springframework.security.util.FilterChainProxy">
  <security:filter-chain-map path-type="ant">
    <security:filter-chain
      pattern="/**"
      filters="httpSessionContextIntegrationFilter,logoutFilter,

```

```

authenticationProcessingFilter, securityContextHolderAwareRequestFilter,
anonymousProcessingFilter, basicProcessingFilter, filterInvocationInterceptor"/>
    </security:filter-chain-map>
</bean>

```

The property `filter-chain-map` allows us to define the mapping from URLs to filter chains, using an instance of `filter-chain` containing an ordered list of filters. It is important to note at this stage that a series of filters will be run - in the order specified by the declaration - and each of those filters is actually the ID of another bean in the application context. So, in our case all the beans that are declared under `filterChainProxy` will also appear in the application context, and they will be named `httpSessionContextIntegrationFilter`, `logoutFilter`, and so on.

Logging Configuration

Logging into ERDAS APOLLO is ensured by Log4j, a Java-based logging utility of the Apache Software Foundation. Log4j offers six standard logging levels. From highest (coarsest) to lowest (finest), those levels are:

- **FATAL:** production, fatal application error, application cannot continue. Could be caused if the database is down, for example.
- **ERROR:** production, application error/exception but application can continue. Part of the application is probably not working.
- **WARN:** production, simple application error or unexpected behavior. Application can continue. Can be used in case of bad login attempts or unexpected data during import jobs, for example.
- **INFO:** production optionally. Can be used to print that a configuration is initialized or that a long running import job is starting and ending, for example.
- **DEBUG:** development only, for debugging purpose.
- **TRACE:** development only, can be used to follow the program execution.

The ERDAS APOLLO Catalog uses these different levels of logging to provide feedback on activity. Logging configuration is defined in the `WEB-INF/classes/log4j.properties` file of your ERDAS APOLLO Catalog web application. In JBoss, this configuration may be overridden by the global `jboss-log4j.xml` file located in the `jboss/conf` directory. This file allows you to define the following.

- **Appenders:** control how the logging is output.

- Layout, associated to appenders: control how to format the output.
- Loggers: responsible for handling the majority of log operations.

See this file for any additional information.

Typical Scenarios

This chapter provides some common scenarios for setting up web services for geodata. The examples rely on sample data used in real-world situations.

The first four sections of this chapter are a step-by-step guide for configuring web services over data. The following sections will optimize and enhance the web services.

Publishing Vector Data in WFS

This section describes the steps to set up a web service over vector data. At the end of the sequence, the service will be able to respond to WFS requests (GetCapabilities, DescribeFeatureType, GetFeature) and to display maps through WMS requests (GetCapabilities, GetMap, GetFeatureInfo).

Shapefile Provider on top of a Data Directory

This section provides an example on how to create a shapefile provider on top of a data directory based on the sample data over the city of Atlanta installed with the product (if you chose that option).

```
C:/Erdas/Apollo2011/data/erdas-apollo/shapes/atlanta
```

1. In the first dialog, create a new vector service in the Data Manager, choose **Service Type** as the vector data and **Data Source Type** as the file.
2. In the Shape File Selection panel, select the first radio button **Data located on the server**.
3. In the second dialog, choose **Shape Service** as the Service Type.
4. In the Shape File Selection panel, choose the directory containing the sample Shapefiles on Atlanta: click **Browse** and navigate to data/erdas-apollo/shapes/atlanta".
5. In the **Shape File SRS** field, select "NAD83/Georgia West State Plane (ftUS)" or encode "EPSG:2240".
6. In the Basic Service Properties panel, enter the following values:
 - Name: ATLCITY
 - Title: City of Atlanta
 - Abstract: City of Atlanta, Shapefile Service setup using ERDAS APOLLO
 - Keywords: Atlanta,Buildings

7. In the Additional Service Properties panel, choose the **Autodetect geometry types** option. Keep the other fields as proposed.

The **Index data** option is selected by default. It creates RTree files beside the shapefiles for optimized access. This operation can take several minutes before completing. This action can be disabled and taken later.

8. After the creation is over, in the Edit panel, click **GetCapabilities** to check that the service is properly initialized. An XML document appears and declares a set of feature types: buildings, futurelanduse, roads, places.

Next Steps:

- When a WFS service is defined, it can be opened in the Apollo Web Client and the layers can be presented as WFS Layers.
- Adding Metadata to a WFS service allows you to publish richer information.
- Creating styles on vector data allows you to expose it as WMS layers. See [Create Styles on Vector Data](#) below for guidance.

In the Style Editor, you will notice that the futurelanduse and places do not overlay the other layers (because the overlay has mismatched SRS code and extent values). It is most visible when right-clicking on those layers and as "Frame the view to the layer". The "Envelope" pane confirms that the coordinates for those layers are not in the Georgia West State Plane but in WGS 84.

To fix this, update the configuration of the provider to indicate that the places and futurelanduse feature types use the EPSG:4326 (WGS84) coordinate system.

1. In the Data Manager edit the provider using the online help.
2. Click on the ATLCITY provider you defined before.
3. In the **Resources** tab of the Edit view, click on the Resource named generatedMapping.xml and click Edit.
4. Go to the <Info> section for the two impacted feature types.
5. Change the EPSG code for the SRS and BoundingBox properties, from EPSG:2240 to EPSG:4326. Save the file.
6. In the **Service Info** tab, click on **Restart Service** for the changes to be applied. Restart the Style Editor and check the extents again.



When complete, save your project into

C:/Erdas/Apollo/tools/styleeditor/projects

Create a Vector Provider on top of Oracle Data

This section provides an example of how to create a vector provider on top of Oracle data based on the sample data over the city of Boston, installed with the product (if you chose that option).

This workflow assumes that the ERDAS APOLLO product has been installed and that the services are accessible via the `http://localhost:8080` URL.

Steps to deploy a WFS on top of Oracle data:

1. If the data is not yet in an Oracle database, take the sample file `<APOLLO_HOME>/data/erdas-apollo/db/oracle/boston_ora9.dmp` and import it in the Oracle9i Spatial or higher database. The command could look like: `imp user/pwd@sid File=boston_ora9.dmp GRANTS=N FULL=Y`



Data provided in the Shapefile format can be imported using the Oracle shp2sdo tool.

2. Whatever the data, this step is complete when an Oracle schema is filled with a set of tables, its rows, indexes and possible constraints, views,...
3. Launch the Data Manager and follow the instructions.
4. In the first dialog, choose Vector Data as the **Service type** and Database as the **Data source** type.
5. Choose Oracle as the **Service** type.
6. In the Connection panel, fill the fields (most of the time, host, port, sid, user and password suffice). Click **Test Connection** to verify that the connection to the database succeeds.
7. In the **Default SRS** field, encode the value "EPSG:26986" (it corresponds to Massachusetts State Plane in Meter).
8. In the Basic Service Properties panel, encode the following values:
 - Name:** BOSTON_ORA
 - Title:** City of Boston
 - Abstract:** City of Boston, Oracle Service setup using ERDAS APOLLO
 - Keywords:** Boston, Oracle

Check **Generate types and mappings** before clicking **Finish**.

The service is created and the Service Provider Editor view shows the properties of the new service.

9. But a set of mapping and types files are needed for the WFS to be properly configured. Choose the **Data Source** tab. Beside the **Mapping File** field click **Browse**. Select `config/erdas-apollo/providers/vector/boston_ora.xml`. Then select the **Types Schema** field, click **Browse**. Select `config/erdas-apollo/providers/vector/boston_ora.xsd`.
10. Click **Save** to persist the changes.
11. Click **GetCapabilities** to check that the service is properly initialized. An XML document displays and declares a set of feature types: highways, hydro, land_use, place_names, protectedareas, roads.



If additional Metadata are needed per layer, the mapping file (generatedMapping.xml) should be edited to add such tags in each <Info> section. Such an addition could look like:

```
<wfs:Title>Protected Areas</wfs:Title>

<wfs:Abstract>Polygons of Boston Protected Areas</
wfs:Abstract>

<wfs:Keywords>Boston, Protected, Areas</wfs:Keywords>
```

Next Steps:

- When a WFS service is defined, it can be opened in the ERDAS APOLLO Web Client and the layers can be presented as WFS Layers.
- Adding Metadata to a WFS service allows you to publish richer information.
- Creating styles on vector data exposes it as WMS layers. See [Create Styles on Vector Data](#) below for guidance.

Create a Transactional Provider over Oracle

This section explains how to convert a vector Oracle Provider (WFS interface) onto a transactional service supporting updates of the data. The main path updates the BOSTON_ORA provider defined previously but a set of Notes describe alternatives if a custom data source is used instead.

This workflow assumes that the ERDAS APOLLO product has been installed and that the services are accessible via the `http://localhost:8080` URL.

Steps to define a WFS-T based on an existing WFS over Oracle data:

1. Add the BUSINESS and LOCKTIMEOUT tables to the Oracle schema. To do this, execute successively the SQL script files `<APOLLO_HOME>/data/erdas-apollo/db/oracle/bus_create.sql` and `lock.sql`. The first one creates the BUSINESS table, the associated indexes and insert a small set of records in that table. The second script creates the LOCKTIMEOUT table used to manage the Locking mechanism.



In the alternative of a custom service, we assume one or more of the existing tables will be made transactional. At this stage, you need to check or adapt the table(s) so that one or more columns identifies each row uniquely (whether or not it is configured as a primary key). Moreover, if you want to enable the Locking mechanism, an additional character-type column should be added.

The SQL statement could be: `ALTER TABLE BUSINESS ADD (LOCK_ID VARCHAR2(255 BYTE));`

2. Launch the Data Manager follow the online help to edit an existing service provider.
3. Select the previously created "BOSTON_ORA" vector service. Display the properties of this service in the Service Provider Editor View.
4. Adapt the content of the mapping file. To do so, select the **Data Source** tab and click **Edit** beside the **Mapping File** field.

If the file relates to BOSTON_ORA, uncomment the Mapping and Info sections referencing BUSINESS at the bottom of the file. If converting a custom table to make it transactional, the following changes are needed:

- If the primary key setting is set to `<NoPrimary/>`, replace that line with an actual Primary Key definition. If the primary key column is named "BUS_ID", it could be: `<Primary name="BUS_ID" type="xsd:integer" fid="generated" />`

The `fid="generated"` option is set to ensure uniqueness of the keys: each time a record is inserted, the system will generate the key value.

- If Locking is enabled, add the following line to declare the "LOCK_ID" column as locking flag key: `<Lock nameSQL="LOCK_ID" />`

- Ideally, the BUS_ID and LOCK_ID columns should not be mapped to a feature property. To achieve this, remove the lines (if they were ever created):

```
<Element name="wfs:BUS_ID" nameSQL="BUS_ID"/>
<Element name="wfs:LOCK_ID" nameSQL="LOCK_ID"/>
```

- In the <Info> tag for your table, only the Query operation is enabled. It should be extended to enable transactional-type operations. To do so, add one or more of the following operation types in a comma-separated list: Insert, Update, Delete, Lock. Alternatively, replacing the whole set with just the "*" value enables all operation types.

The end of the mapping file for BOSTON_ORA could look like this:

```
<Mapping>
  <SQL name="wfs:BUSINESS">
    <Table nameSQL="BUSINESS"/>
    <Primary name="BUS_ID" type="xsd:integer" fid="generated"
  />
    <Lock nameSQL="LOCK_ID" />
    <Element name="wfs:NAME" nameSQL="NAME"/>
    <Element name="wfs:TYPE" nameSQL="TYPE"/>
    <Element name="wfs:STREET_NAME" nameSQL="STREET_NAME"/>
    <Element name="wfs:BUILDING_NBR" nameSQL="BUILDING_NBR"/>
    <Element name="wfs:POSTCODE" nameSQL="POSTCODE"/>
    <Element name="wfs:CITY" nameSQL="CITY"/>
    <Element name="wfs:TELEPHONE" nameSQL="TELEPHONE"/>
    <Element name="wfs:TOTAL_EMPLOYEES"
nameSQL="TOTAL_EMPLOYEES"/>
    <Element name="wfs:GEOMETRY" nameSQL="GEOMETRY"/>
  </SQL>
</Mapping>
<!--Info for type wfs:BUSINESS - to enable when table is created
-->
  <Info name="wfs:BUSINESS">
    <Operations>*</Operations>
    <SRS>EPSG:26986</SRS>
    <BoundingBox SRS="EPSG:26986" minx="227317." miny="889948."
maxx="238670." maxy="901300."/>
  </Info>

</xsd:schema>
```

5. Save the updated mapping file.
6. Synchronize the content of the Types Schema file, so that the transactional table is declared with only the mapped properties exposed. To do so, select the **DataSource** tab and click **Edit** beside the **Types Schema** field.

If the file relates to BOSTON_ORA, uncomment the xsd:element and xsd:complexType sections referencing BUSINESS at the bottom of the file. If converting a custom table to make it transactional, make sure the primary key (e.g. BUS_ID) and locking property (e.g. LOCK_ID) only appear in this file if they have been mapped in an <Element> tag in the mapping file. If not, remove those properties from the feature type definition. Those lines look like this:

```
<xsd:element name="BUS_ID" type="xsd:int">
</xsd:element>
<xsd:element name="LOCK_ID" minOccurs="0" nillable="true"
type="xsd:string">
</xsd:element>
```

7. Save the updated types schema file.
8. Click **GetCapabilities** to check that the service is properly updated. An XML document displays. For BOSTON_ORA, that document declares a new feature type: wfs:BUSINESS.

For BUSINESS and for custom transactional tables, the permitted operation set appears behind each FeatureType declaration, in a <Operations> tag. The BUSINESS feature type declaration could look like:

```
<FeatureType xmlns:wfs="http://www.ionicssoft.com/wfs">
  <Name>wfs:BUSINESS</Name>
  <SRS>EPSG:26986</SRS>
  <Operations>
    <Query/>
    <Insert/>
    <Delete/>
    <Update/>
    <Lock/>
  </Operations>
  <LatLongBoundingBox minx="-71.16892567638165"
miny="42.25904339835999" maxx="-71.03057571317116"
maxy="42.361722456564976"/>
</FeatureType>
```

Next Steps:

- Manual checks of transactional behavior can be done with the Service Tester applet (under <http://localhost:8080/erdas-apollo/servicetester>). Sample and template operations are provided in <APOLLO_HOME>/data/erdas-apollo/db/oracle/bus_insert.xml and bus_transac.xml.

Create a PostgreSQL/PostGIS Vector Provider

- When a WFS-T service is defined, it can be opened in the ERDAS APOLLO Web Client as a WFS, and the layers can be managed as WFS Layers: When information is requested on a feature, that information holds links to allow edition, creation, or deletion of features.

This section provides an example on how to create a PostgreSQL/PostGIS vector provider, based on the sample data over the city of Boston, installed with the product (if you chose that option).

This workflow assumes that the ERDAS APOLLO product has been installed and that the services are accessible via the `http://localhost:8080` URL.

Steps to deploy a WFS on top of PostgreSQL/PostGIS data:

1. If the data is not yet in a PostgreSQL database, you must import it from shapefiles. Use the command line to generate an sql script file that will enable you to create the tables and to populate them: `shp2pgsql table.shp schema.table > table.sql`. Do this for every table you want to include in the database.
2. Make the tables and populate them: `cat table.sql | psql dbname`
NOTE: Note that a PostGIS-enabled PostgreSQL database contains the tables `geometry_columns` and `spatial_ref_sys` in the user schema.
3. Whatever the data, this step is complete when a PostgreSQL/PostGIS schema is filled with a set of tables, its rows, indexes and possible constraints, views,... .
4. Launch the Data Manager.
5. In the first panel, choose **Vector Data** as the **service type** and choose **Database** as the **Data Source type**. In the second panel, choose **PostgreSQL Service** as the **service type**.
6. In the Connection panel, fill the fields (most of the time, host, port, database, user and password suffice). Click **Test Connection** to verify that the connection to the database succeeds.
7. In the **Default SRS** field, encode the value "EPSG:26986" (it corresponds to Massachusetts State Plane in Meter).
8. A set of mapping and types files are needed for the WFS to be properly configured. Choose the Data Source tab. Beside the **Mapping File** field, click **Browse**. Select `config/erdas-apollo/providers/vector/boston_pg.xml`.

Then, select the **Types Schema** field, click **Browse**. Select `config/erdas-apollo/providers/vector/boston_pg.xsd`.

9. Click **Save** to persist the changes.
10. Click **GetCapabilities** to check that the service is properly initialized. An XML document appears that declares a set of feature types: highways, hydro, land_use, place_names, protectedareas, roads.



If additional Metadata are needed per layer, the mapping file (generatedMapping.xml) should be edited to add such tags in each <Info> section. Such an addition could look like:

```
<wfs:Title>Protected Areas</wfs:Title>
```

```
<wfs:Abstract>Polygons of Boston Protected  
Areas</wfs:Abstract>
```

```
<wfs:Keywords>Boston, Protected, Areas</wfs:Keywords>
```

Next Steps:

- When a WFS service is defined, it can be opened in the ERDAS APOLLO Web Client and the layers can be presented as WFS Layers.
- Adding Metadata to a WFS service allows you to publish richer information.
- Creating styles on vector data allows to expose it as WMS layers. See [Create Styles on Vector Data](#) for guidance.
- Extending the provider to WFS-T. The procedure is in all details similar to the Transactional Provider over Oracle. Just make sure to use <APOLLO_HOME>/data/erdas-apollo/db/postgresql/bus_create_pg.sql and lock_pg.sql sql scripts over PostgreSQL, and bus_create_pgis.sql and lock_pg.sql over PostGIS.



The key generation option fid (Oracle)/gid (Postgres)="auto" (attribute of the "Primary" property in the mapping file) is not supported for PostgreSQL/PostGIS as this DBMS does not support the unique identifier auto-generation option. Oracle does, based on UUIDs.

Create an ArcSDE Vector Provider

This section explains how to set up a vector Provider (WFS interface) on an ArcSDE database over the city of Boston. Additionally, each Note describes alternatives or additional operations if a custom data source is used instead.

This workflow assumes that the ERDAS APOLLO product has been installed and that the services are accessible via the `http://localhost:8080` URL.

Steps to deploy a WFS on top of an ArcSDE server:

1. Before configuring the service, add the ArcSDE SDK library to the web application. This library is composed of a small set of jar files available in the ArcSDE Java SDK Installation Directory, under the lib folder. They can be named `jsdeXX_sdk.jar`, `jpeXX_sdk.jar`, and possibly `icu4j.jar` where XX is the version of ArcSDE.

Copy those jar files into `<APOLLO_HOME>/webapps/erdas-apollo/profiles/eas/WEB-INF/lib` (for APOLLO Essentials) or into `<APOLLO_HOME>/webapps/erdas-apollo/profiles/eaim/lib`. Rebuild the erdas-apollo webapp by running `ant` from `<APOLLO_HOME>` as described in [Using Apache Ant to Rebuild the Webapps](#) on page 161 and redeploy them into your servlet engine as described in [Deploying WAR Files on Supported Servlet Engines](#) on page 164.



ArcSDE libraries are not included with ERDAS APOLLO.

2. If the data are not yet stored in a database or not yet configured as a "feature class" in an ArcSDE server, start by loading it using one of the methods described in the sample file `<APOLLO_HOME>/data/erdas-apollo/db/arcsde/bus_create_sde.txt`. For this scenario, use the BUSINESS Shapefile included in that same sample data directory.



Data provided in the Shapefile format can be imported using the ArcSDE `shp2sde` tool. The command could be:

```
shp2sde -o create -l business,geometry -f BUSINESS -a  
all -G 26986 -e -p -u <username> [-k ERDAS]
```


Replace "<username>" with the real ArcSDE login name. The value "26986" expresses the projection system. The "-k ERDAS" option is to be set if the default geometry storage method in the ArcSDE server does not correspond to the requirements. For example, if the underlying database is Oracle Spatial and the geometries are to be stored as Oracle SDO_GEOMETRY type, create a new entry in the ArcSDE DBTUNE table setting the "GEOMETRY_STORAGE" property to "SDO_GEOMETRY" and name it ERDAS. The -k parameter in the command refers to that name. The alternative methods to populate data are either using the sdetable and sdelayer commands or creating the SQL table first and then registering it as a feature class. The commands are illustrated in the bus_create_sde.txt file and the SQL scripts are bus_create_ora.sql for ArcSDE/Oracle and bus_create_mssql.sql for ArcSDE/MS-SQL Server.

3. Whatever the data, this step is complete when an ArcSDE database is filled with a set of tables, its rows, indexes and possible constraints, views,...
4. Launch the Data Manager and follow the instructions to create the service, setting "BOSTON_SDE" as service name and "City of Boston" as service Abstract and Title. The wizard creates an incomplete service. You still must edit the provider properties and encode values.
5. In the **Data Source** tab, expand the **Connect String** property and fill the sub-fields (most of the time, host, port, instance, user and password suffice). Click **Test Connection** to verify that the connection to the database succeeds.
6. A set of mapping and types files are needed for the WFS to be properly configured. Choose the Data Source tab. Beside the **Mapping File** field, click **Browse**. Select config/erdas-apollo/providers/vector/bus_sde.xml. Then, select the **Types Schema** field, click **Browse**. Select config/erdas-apollo/providers/vector/bus_sde.xsd.
7. Click **GetCapabilities** to check that the service is properly initialized. An XML document displays and declares a feature type: business.



If additional Metadata are needed per layer, the mapping file (generatedMapping.xml) should be edited to add such tags in each <Info> section. Such an addition could look like:

```
<wfs:Title>Businesses</wfs:Title>
```

```
<wfs:Abstract>Points of Boston  
Businesses</wfs:Abstract>
```

```
<wfs:Keywords>Boston,Business</wfs:Keywords>
```

8. Click **Save** to persist your changes into the actual configuration file.

Next Steps:

- When a WFS service is defined, it can be opened in the ERDAS APOLLO Web Client and the layers can be presented as WFS Layers.
- Adding Metadata to a WFS service allows you to publish richer information.
- Creating styles on vector data allows you to expose it as WMS layers. See [Create Styles on Vector Data](#) for guidance.
- Extending the provider to WFS-T. The procedure is similar to the Transactional Provider over Oracle. Just make sure to use `<APOLLO_HOME>/data/erdas-apollo/db/oracle/lock.sql sql script over ArcSDE/Oracle`, and `<APOLLO_HOME>/data/erdas-apollo/db/arcsde/lock_mssql.sql over ArcSDE/MS-SQL`.

More guidance on setting up an ArcSDE vector data source is given in [Provider Types](#)

Create a Vector Provider on top of GML Data

This section provides an example on how to create a vector provider on top of GML data based on the sample data installed with the product (if you chose that option).

This workflow assumes that the ERDAS APOLLO product has been installed and that the services are accessible via the `http://localhost:8080` URL.

Steps to deploy the ATLANTA WFS on top of GML data:

1. If the data is not yet in a GML file, use the sample file `<APOLLO_HOME>/data/erdas-apollo/gml/atlanta/atlanta21a.gml` as GML file input. If the GML document has to be produced, it can be generated through a GetFeature request on any WFS service.

Along with the GML file, it is necessary to have an XML Schema file holding the feature types definitions used in the GML document. That schema could be referenced at the beginning of the GML file but it can also be obtained through a DescribeFeatureType request on any WFS service. In our scenario, the schema is provided beside the GML file and is named atlanta.xsd. Use it as value for the **Type File** field.

2. In the Basic Service Properties panel, set the following values:

Name: ATLANTA_GML

Title: City of Atlanta

Abstract: City of Atlanta, GML service setup using ERDAS APOLLO

Keywords: GML service, Atlanta, Georgia, Buildings



A Mapping file could also be mentioned. In the case of a GML provider, the gml-to-datasource mapping is trivial, but that file can be used to set additional information such as the data extent, some metadata, The distribution includes such a mapping file, named `generic_sql_mapping.xml` and located beside the providers.fac.

3. Click **GetCapabilities** to check that the service is properly initialized. An XML document displays and declares the feature types buildings and roads.

Next Steps:

- When a WFS service is defined, it can be opened in the ERDAS APOLLO Web Client and the layers can be presented as WFS Layers.
- Adding Metadata to a WFS service allows you to publish richer information.
- Creating styles on vector data allows you to expose it as WMS layers. See [Create Styles on Vector Data](#) for guidance.
- For a GML provider to disable transactions, the **Disable Transaction** property should be set to "true" in the Administration Console.



More advanced information is provided in the "Provider Types" chapter, for the GML and GML-T WFS - or Vector - Connectors.

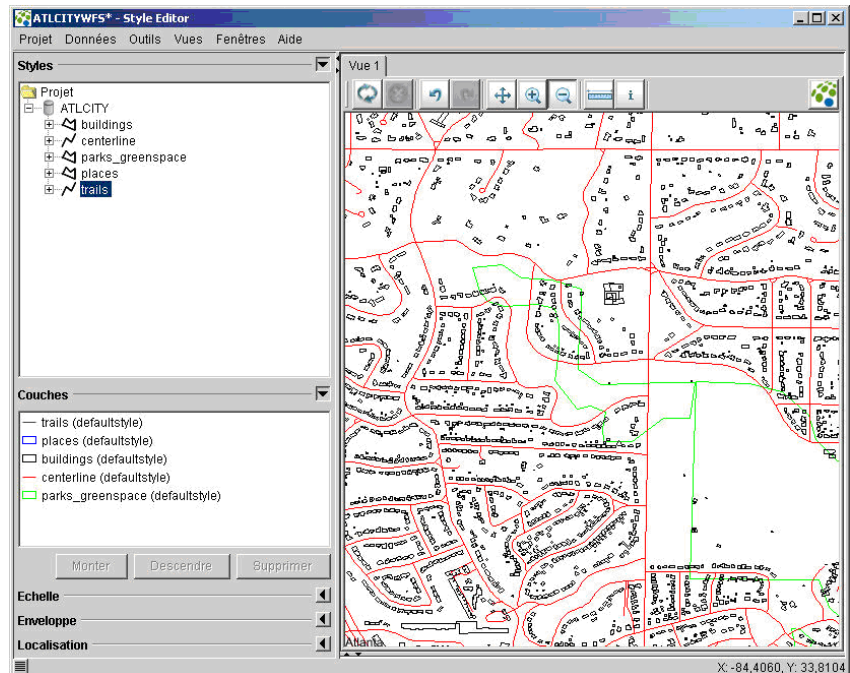
Create Styles on Vector Data

It is possible to use the ERDAS APOLLO Style Editor tool to create a style bundle that can be used to render the vector data. To do that, follow these steps:

1. Start the ERDAS APOLLO Style Editor Application
2. Right-click on **Project**; choose **Add Data Source** then **Web Feature Server** and click **Next**.
3. Input the URL of your service (e.g. <http://localhost:8080/erdas-apollo/vector/ATLCITY>) and click **Add**, once your URL appears in the upper window, click **Next>>**, choose a name and a title.
4. Click **Finish**.

The service and the defined feature types appear in the left pane. (The Finish button is disabled after adding the URL.)

5. Right-click on each feature type name and choose **Add to preview**. The name will display in the Layers panel and the right panel map will fill with graphics.





*It frequently appears that a feature type holds heterogeneous geometries (such as lines and multi-lines), leading to an error message when trying to display them. It can be avoided by enabling the **forgiving** flag for the data source. Right-click on the project name (e.g. "ATLCITY") and choose **Properties** in the list. In the panel, check the **forgiving** box.*

6. To update one or more styles, right-click on the layer name in the Layers view and choose Properties. A panel lets you configure your style.
7. When you are finished with the rendering, deploy the styles using the menu **File -> Styles -> Deploy to Directory...** . Choose the `<APOLLO_HOME>/config/erdas-apollo/rendering` directory and click **Save**. The set of styles will be copied there, allowing you to display nice layers using the WMS GetMap interface.



When finished with the Style Editor, save your project into `<APOLLO_HOME>/tools/styleeditor`

Publishing Images in WMS

Raster Images

Starting with ERDAS APOLLO 2010, Raster Images services are no longer served by the "map" servlet. They are now included in the set of services provided through the "coverage" servlet. Please refer to the next section for the step-by-step sequence of setting up a Raster Images service.

Publishing Raster Data in WCS

Simple Coverage Services

This section describes how to configure a Web Coverage Service Provider to serve datasets and raster images through the WMS and/or WCS interfaces.

This section provides an example based on the sample data over the city of Atlanta, installed with the product (if you chose that option).

This workflow assumes that the ERDAS APOLLO product has been installed and that the services are accessible via the `http://localhost:8080` URL.

Add an Atlanta Tile (ECW)

Data Path: <APOLLO_HOME>/data/erdas-apollo/coverages/
mosaic/atl_tiles_1_1.ecw

1. After login, right-click on the **Rasters** node and choose **Create Service**.
2. In the Service creation wizard panel, select **Raster data** as Service type and **File** as Data source type. Click **Next**.
3. Select **Single** as Service type. Click **Next**.
4. Choose the **Data located on the server** option. Click the **Browse** button beside the **Raster File** field. Choose the tree <APOLLO_HOME>/data/erdas-apollo/coverages/mosaic/atl_tiles_1_1.ecw and click **OK**.
5. As Raster SRS, either encode **EPSG:2240** or click the **Select...** button to choose the **NAD83/Georgia West State Plane (ftUS)** projection. Click **Next**.
6. Specify "Rockdale_Tile" as the **Service Name**, Rockdale as **Title and Abstract**, Atlanta, Rockdale, ECW, 2240 as **Keywords**.
7. In the **Select the main service properties** panel, click **Finish**.
8. After a few seconds, a new item named Rockdale_Tile appears under the **Rasters** tree node. The right pane of the Data Manager displays the properties of the newly created service.
9. Click **Get Capabilities** to see the Capabilities document (if you see some kind of error please go back and check all of your input parameters and retry).



*If the service is only intended to be used through the WMS interface, open the **Miscellaneous** tab in the service properties panel and uncheck the **WCS** box beside the **Enabled Interfaces** property.*

Mosaic and List Coverage Services

Use preconfigured MultiSimple Coverage Service

This section provides an example based on the sample data over the city of Atlanta, installed with the product (if you chose that option).

This workflow assumes that the ERDAS APOLLO product has been installed and that the services are accessible via the `http://localhost:8080` URL.

Add Atlanta 2002 ECW data:

Data Path: `<APOLLO_HOME>/data/erdas-apollo/coverages/mosaic`

EPSG:2240

Background Value: 0

1. After logging in the Data Manager, right-click on the **Rasters** node and choose **Create Service**.
2. In the Service creation wizard panel, select **Raster data** as Service type and **File** as Data source type. Click **Next**.
3. Select **List** as Service type. Click **Next**.
4. Choose the **Data located on the server** option. Click the **Browse** button beside the **Raster Dir** field. Choose the tree `<APOLLO_HOME>/data/erdas-apollo/coverages/mosaic` and click **OK**.
5. As Raster SRS, either encode **EPSG:2240** or click the **Select...** button to choose the **NAD83/Georgia West State Plane (ftUS)** projection. Click **Next**.
6. In the main service properties panel, encode "ATLANTA_LIST_2002" as **Name**, "Atlanta List" as **Title and Abstract**, "Atlanta, 2002, ECW, 2240" as **Keywords**.
7. In the main service properties panel, just click **Finish**. Notice that a checkbox named **Index data** is checked, meaning that the collection of images will automatically be indexed.

After a few seconds, a new item named ATLANTA_LIST_2002 is added under the **Rasters** tree node. The right pane of the Data Manager displays the properties of the newly created service.

8. Click the **Get Capabilities** link to see the Capabilities document (if you see some kind of error please go back and check all of your input parameters and retry).



*If the service is only intended to be used through the WMS interface, open the **Miscellaneous** tab in the service properties panel and uncheck the **WCS** box beside the **Enabled Interfaces** property.*

IndexProvider scenario

This section provides an example of an Index Provider scenario based on the sample data over the city of Atlanta, installed with the product (if you chose that option).

This workflow assumes that the ERDAS APOLLO product has been installed and that the services are accessible via the `http://localhost:8080` URL.

Add Atlanta 2002 ECW data:

Data Path: `<APOLLO_HOME>/data/erdas-apollo/coverages/mosaic`

EPSG:2240

Background Value: 0

1. After losing in the Data Manager, right-click on the **Rasters** node and choose **Create Service**.
2. In the Service creation wizard panel, select **Raster data** as Service type and **File** as Data source type. Click **Next**.
3. Select **Mosaic** as Service type. Click **Next**.
4. Choose the **Data located on the server** option. Click the **Browse** button beside the **Raster Dir** field. Choose the tree `<APOLLO_HOME>/data/erdas-apollo/coverages/mosaic` and click **OK**.
5. As Raster SRS, either encode **EPSG:2240** or click the **Select...** button to choose the **NAD83/Georgia West State Plane (ftUS)** projection. Click **Next**.
6. In the main service properties panel, encode "ATLANTA_INDEX_2002" as **Name**, "Atlanta Index" as **Title and Abstract**, "Atlanta, 2002, ECW, 2240" as **Keywords**.
7. In the main service properties panel, click **Finish**. Notice that a checkbox named **Index data** is checked, meaning that the collection of images will automatically be indexed.

After a few seconds, a new item named ATLANTA_INDEX_2002 is added under the **Rasters** tree node. The right pane of the Data Manager displays the properties of the newly created service.

8. Click **Get Capabilities** to see the Capabilities document (if you see some kind of error please go back and check all of your input parameters and retry).



*If the service is only intended to be used through the WMS interface, open the **Miscellaneous** tab in the service properties panel and uncheck the **WCS** box beside the **Enabled Interfaces** property.*

ArcSDE-Raster

ERDAS APOLLO WMS supports serving raster data stored in ESRI ArcSDE. This scenario describes how to set of a WMS over an ArcSDE-Raster data source.

This scenario assumes:

- An ESRI ArcSDE server is running on a host named arc.sde.com.
- An SDE schema contains a table named BOSTON_SDER with a raster column belonging to the SDE user "sdeusr" with the password "sdepwd".
- The sample raster data on BOSTON are loaded.
- ERDAS APOLLO is installed on another server and the erdas-apollo.war archive is deployed.

Refer to [Provider Types](#) for more configuration information.

Environment Configuration

Before configuring the service, add the ArcSDE SDK library to the web application. This library is composed of a small set of jar files available in the ArcSDE Installation Directory, under the lib folder. They can be named jsdeXX_sdk.jar, jpeXX_sdk.jar, and possibly icu4j.jar where XX is the version of ArcSDE.

Copy those jar files into `<APOLLO_HOME>/webapps/erdas-apollo/profiles/eas/WEB-INF/lib` (for APOLLO Essentials) or into `<APOLLO_HOME>/webapps/erdas-apollo/profiles/eaim/lib` (for APOLLO Advantage/Professional). Rebuild the erdas-apollo webapp by running ant from `<APOLLO_HOME>` as described in [Using Apache Ant to Rebuild the Webapps](#) on page 161 and redeploy them into your servlet engine as described in [Deploying WAR Files on Supported Servlet Engines](#) on page 164.

Provider setup

Steps to deploy a WMS on top of an ArcSDE-Raster server:

This section describes the steps specific to ArcSDE-Raster and provides an example based on the sample data over the city of Boston, installed with the product (if you chose that option).

This workflow assumes that the ERDAS APOLLO product has been installed and that the services are accessible via the `http://localhost:8080` URL.

1. After logging in the Data Manager, right-click on the **Rasters** node and choose **Create Service**.
2. In the Service creation wizard panel, select **Raster data** as Service type and **Database** as Data source type. Click **Next**.
3. Select **ArcSDE (portable) Service** as Service type. Click **Next**.
4. In the main service properties panel, encode the following values:
Name: BOSTON_SDER
Title and Abstract: City of Boston
Keywords: ArcSDE, Boston, Raster
5. Click **Next** then **Finish**.



A provider type named "ArcSDE (native) Provider" also exists but it is deprecated and implies a much heavier environment setup as it uses native libraries (DLLs).

6. After a few seconds, a new item named "BOSTON_SDER" is added under the **Rasters** tree node. The right panel of the Data Manager displays the properties of the newly created service.



The creation wizard is currently limited to a part of the steps needed to create a valid ArcSDE-Raster service. The rest of the work has to be done in the properties panel of the Data Manager for that service, as described below.

7. In the **Data Source** tab of the properties panel, expand the **Connect String** property and fill the sub-fields (most of the time, host, port, instance, user and password suffice).
8. The layers to be published by the service need to be configured. Expand the **layers** property and click on **Add Entry**. Expand the newly displayed node to show its subproperties. Some of those properties need to be filled, such as table and column. The proposed values are:
SRS set to EPSG:26986
Title: SDE Raster Image on Boston
column: image
table: boston_raster
9. Refer to **Provider Types** for more capabilities. When done, click the Save icon to persist your changes into the actual configuration file.
10. Click **GetCapabilities** to check that the service is properly initialized. An XML document displays and declares a layer named boston_BOSTON_RASTER_image_null.

Next steps:

- When a WMS service is defined, it can be opened in the ERDAS APOLLO Web Client and the layers can be presented as WMS Layers.
- If there are several ArcSDE raster tables and several layers are to appear in the service, there are two ways to do it. The first way is to use a pattern as table value (e.g. "uk_%"), that will produce one layer per raster table; the layer name will have the table name. The second way is to explicitly define several "layers" properties by clicking several times on the **Add Entry** link.
- Refer to **Provider Types** for more configuration information.

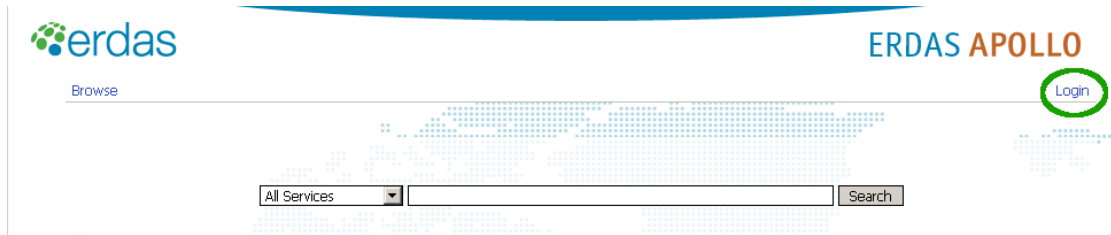
Populate, Browse and Query the Catalog

This section describes typical usage of the catalog through its web interface. An exhaustive description of the Catalog Web Interface is provided in the **Catalog Web Interface** section. [How is this different from the Catalog Web Interface in the WC User's Guide?](#)

Authentication

Although the default configuration of the catalog allows read-only access to part of the content, it is usually necessary to authenticate to get full access and enable specific operations, like publishing, updating or management of the catalog.

1. On the main page of the catalog web interface, click **login**.



2. Then you have two text fields, the first (left to right) is the user name, the second is for the password.
3. Finally, press **enter** or the **ok button**. If the authentication succeeds, the username appears at the upper right of the page, beside the *Logout* action. Otherwise, the message "Invalid login/password" displays.

Publish a service

If the logged in user has the required roles, he will be allowed to publish data in the catalog. By default, roles that are granted these rights are *BABEL_PUBLISHER* and *BABEL_ADMIN*.

1. First, log in with an user having one of the *BABEL_PUBLISHER* or *BABEL_ADMIN* role.
2. On the upper left, use the *Publish* action.



3. The publish operation accepts various types of resources. This section will focus on OGC Service publishing; publishing of other types of resources will be covered in the Catalog Web Interface section.
4. Using the drop-down list, specify the OGC service type to publish (WMS, WFS, WCS). The "W*S" value will cause the publish process to use heuristics to guess the service type from the URL (checking for the presence of a "service" parameter, inspecting the structure of the URL). If a specific value (WMS, WFS, WCS) is selected, it always overrides the service type that may be inferred from the URL.

5. Type in the service URL in the text field. It can be the URL of the GetCapabilities operation, or simply the base URL of the service.
6. Press the *Publish button* or *enter*. The publishing process will start and may take a while, depending of the size of the resources being harvested.
7. When the publish process is done, the interface is redirected to the newly created object.

Data Discovery

This section explains how to discover and browse resources stored in the catalog. Advanced browsing scenarios are covered in the [Catalog Web Interface section](#).



Pressing CTRL+ALT keys while in the Browse panel will display contextual help.

1. Go to the **Browse** panel. This panel is also available for anonymous users.
2. Select the data type of interest in the drop-down list. This drop-down lists the usual data types of interest, i.e. Vector, Map and Coverage resources/services.
3. To search for vector data regarding *road in Atlanta*, select “Vector layers” and type "road* Atlanta" in the search bar. Note that the wildcard (*) is used here to hit the *road* and *roads* words.



ERDAS APOLLO

Browse

Login

The screenshot shows the ERDAS APOLLO Browse panel. At the top, there is a search bar with a dropdown menu set to "Vector Layers" and the search text "road* AND atlanta". Below the search bar, the results are displayed in a list format. The first result is "FEATURE TYPE RAILROAD" with tags [ERDAS, APOLLO, Vector, Cherokee, Oracle, 2006]. The second result is "FEATURE TYPE Roads" with tags [ERDAS, APOLLO, vector, shapefile, Cherokee]. The third result is "FEATURE TYPE Roads" with tags [ERDAS, APOLLO, Vector, Cherokee, Oracle]. The fourth result is "FEATURE TYPE Roads53" with tags [ERDAS, APOLLO, Vector, Cherokee, Oracle]. Each result has a small thumbnail image to its left and a right-pointing arrow icon to its right. The page number "Page 1" and "from a total of 14 records" are visible at the top of the results area.

This will result in a list of matching records (in this case Feature Types), presented with a brief description and thumbnail if available. Clicking on the title of any of those records will browse to the detailed description of that record, which may contain links to other records where relevant (e.g. a FeatureType record will contain a link to its owning WebFeatureService).

At any point while browsing the catalog, icons on the upper right of the browse area provide quicklinks to other representations of the current record(s), in KML, GeoRSS, or for a direct view in GoogleMaps (if the catalog server is publicly available).



As an example of use, the current query on “road AND Atlanta” can be exported to GeoRSS. By doing so, the resulting GeoRSS resource will represent a feed that can alert you any time a new resource matching that query is registered in the catalog.

At the top of the browsing area, paging links are displayed when the number of results is too big for a single page display. They offer an easy way to quickly go through a large result set.



By default, no more than 500 records are counted. If you really need to browse further, or to know the actual total number of results, browse to the last page, as this will force the counting of the whole result set.

Using the CSW endpoint

This section explains how to use the CSW endpoint to discover data.

The CSW endpoint is available out of the box at the URL `http://<serverURL>/erdas-apollo/catalog/csw.jsp`. Requests compliant with the OGC CSW 2.0.2/ebRIM 1.0 Application Profile can be sent to this endpoint.

To easily test those requests, a CSW test page is available in the catalog web interface. This page provides a convenient way to send requests using HTTP POST on the CSW endpoint; it also contains a set of sample CSW requests.

This page can be accessed by logging as admin in the web interface, and then click on the CSW tab. Please see the [Testing the CSW endpoint](#) section for details on how to use the catalog web interface.

Assembling Services and Combining Data

Pyramid WMS

This section describes the Pyramid WMS and Cascading WMS.

A common situation encountered is having multiple resolutions of an image or a set of images. There is a trade-off between performance and the size of the image being served. The request's map scale determines the need to serve a larger, high-resolution image or a smaller, low-resolution image. Raster pyramiding can be used to define a scale range and output resolution for quick access and display of very large images.

The Pyramid Provider connector acts as a proxy provider on top of one or more providers and chooses among the providers for each request depending on the scale. Therefore, it is necessary to configure one provider for each different data sources and display scale plus the pyramid provider itself.

The Pyramid Builder tool automatically creates an optimized pyramid of Geotiff files from a layer of images. Refer to [Provider Types](#) for the steps needed to configure a Pyramid Provider. Refer to Chapter 14 "Tools and Viewers", Section 11 "Pyramid Builder" for the steps needed to create a pyramid with the Pyramid Builder.

Cascading with an OpenGIS WMS Context

A WMS provider can be set up on top of an OGC WMS Context file or a URL that serves Context. Various OGC-WMS compliant tools are available to build a Context file, including the Data Manager and the Web Client.

Refer to [Provider Types](#) for an example of configuring a Context Provider.

Chaining Services

This section explains how to configure services that will chain other existing OGC-compliant WMS, WFS and WCS.

Proxying a OpenGIS-compliant WMS

To proxy an existing OGC-compliant WMS, use the ERDAS APOLLO Data Manager to define and configure a WMS Proxy provider. The distribution predefines such a service named PROXYDEMIS. To define your own use the following steps:



This workflow is no longer performed through the Administration Console.

1. Open the Data Manager

2. Connect to your running ERDAS APOLLO Server
3. In the **Explorer** tab, open the tree
4. On the **Maps & Proxies** tree node, right-click and select **Create Service**
5. Leave the **Service type** field set to **Maps & Proxies**, and select the **Advanced Data source type**
6. Click **Next** and choose the service type **Proxy Service**
7. Click **Next** and enter the basic information for the proxy service (e.g., PROXYDEMIS)
8. Click **Finish**
9. Your new proxy service has been created. Go to the **Explorer** tab, and select it in the tree. Right-click on it, and select **Edit Provider**
10. In the **Data Source** properties tab, enter the URL of the WMS service that is being proxied
11. Save your modifications, and reload the service

Other parameters can be configured to enhance the proxied service, including:

- LIMITEDSIZE
- LIMITEDCOLOR
- LIMITEDTRANSPARENCY
- HIDDEN_MAP_FORMAT
- HIDDEN_INFO_FORMAT
- USER
- PASSWORD

For a complete definition of the parameters, see [Detailed Parameters of a Provider](#).

Proxying a OpenGIS-compliant WFS

To proxy an existing OGC-compliant WFS, use the ERDAS APOLLO Data Manager to define and configure a WFS Proxy provider. The distribution predefines such a service named PROXYWORLD. To define your own use the following steps:



This workflow is no longer performed through the Administration Console.

1. Open the Data Manager
2. Connect to your running ERDAS APOLLO Server
3. In the **Explorer** tab, open the tree
4. On the **Vectors** tree node, right-click and select **Create Service**
5. Leave the **Service type** field set to **Vector Data**, and select the **Advanced Data** source type
6. Click **Next** and select the **Proxy Service** service type
7. Click **Next** and enter the basic information for the proxy service (e.g., PROXYWORLD)
8. Click **Next**, then **Finish**
9. Your new proxy service has been created. Go to the **Explorer** tab, and select it in the tree. Right-click on it, and select **Edit Provider**
10. In the **Data Source** properties tab, enter the URL of the WFS service that is being proxied
11. Save your modifications, and reload the service

Other parameters can be configured to enhance the proxied service, including:

- TITLE
- ABSTRACT
- KEYWORDS
- CONTACT
- USER
- PASSWORD

For a complete definition of the parameters, see [Detailed Parameters of a Provider](#).

SLD Portrayal Service for Features and Coverages

The SLD Portray Provider does not hold any data; it simply forwards requests to a WFS or a WCS, ingests a collection of features or coverages, and portrays them using the SLD document. The SLD document is submitted along with the GetMap request and produces an image in the format requested. A Portray Provider is preconfigured in the ERDAS APOLLO distribution and is accessible at: <http://localhost:8080/erdas-apollo/map/PORTRAY>.

SLD Portray Service requires an SLD document. See [Portrayal Capabilities](#), section "Languages" for the whole set of supported tags.

To set up a new Portray Provider or to modify the pre-configured one refer to [Provider Types](#), the "Portray Provider" section.

Producing Smart Maps

This section presents ways to improve the quality of maps produced using WMS.

WMS by Portraying Features

This scenario assumes a WFS is available and responds successfully to GetCapabilities, DescribeFeatureType and GetFeature requests. To display maps, instead of obtaining GML documents describing features, create portrayal styles that instruct the WFS how to transform the requested features.

To create styles, follow these steps:

1. Set the root directory where the WMS servlet will search for styles. Open the `providers.fac` file and set the DIR attribute of the `<STYLE>` element in the `<CONFIGURATION>` section to a path on the system where styles will be stored. The default is `<APOLLO_HOME>/config/erdas-apollo/rendering`.
2. Start the ERDAS APOLLO Style Editor. Add a WFS Data Source by entering the service URL. For example: `http://localhost:8080/erdas-apollo/vector/ATLANTA_VECTOR`. The various feature types defined on that server will be displayed. Use the ERDAS APOLLO Style Editor to create styles for each of the feature types that will be displayed in map requests. See [The ERDAS APOLLO Style Editor](#) for details on exploring and styling data.
3. When finished, select File > Styles > Create Bundle. This will create a `.gar` file. Save this file in the styles root directory of the WFS (Step 1).
4. To view the stylized features, discover the service as WMS in the ERDAS APOLLO Web Client and select one of the newly-styled layers for addition in the map.

Map Dressing Service

The Map Dressing Service adds a grid, scale bar, border or north arrow. Use the ERDAS APOLLO Style Editor to build portrayal styles for a WFS that will contain these presentation elements or output them from a distinct service independent of the existing data sources. The Map Dressing service behaves in exactly the same way as an OGC-compliant WMS. The only difference is that the Map Dressing service has no data and builds the map on-the-fly for each request using a configuration file.

The ERDAS APOLLO distribution provides a pre-configured Map Dressing provider that is configured to respond to requests at <http://localhost:8080/erdas-apollo/map/MAPDRESSING>.

Read the content of the WMS capabilities that is returned from a GetCapabilities request from that service, to determine what layers and styles are provided. See [Using the Map Dressing Service](#) for a more detailed description of the parameters to include in a request. That chapter also explains how to adapt the configuration of a Map Dressing Service to add a custom north arrow or define new styles with predefined values for some of the parameters.

Advanced Portrayal

Classical portrayal aims at rendering features or coverages to produce realistic maps. To improve map branding or apply more complex processing, use the Advanced ERDAS APOLLO Portrayal Engine.

Advanced portrayal configuration includes:

- Legend Display
- Choice of Symbols from a large, extensible symbol library
- General Range and Discrete Classifications
- Road-Oriented Range and Discrete Classifications
- Pattern for area rendering
- Symbol Roller to display one or more symbols along a line
- Variable Markers and Numbers for point display
- And much more

The ERDAS APOLLO Style Editor allows interactive configuration to create rules that will produce professional cartographic output. These custom rules can be added to the style library and reused.

Add a Legend

1. Open the Style Editor.
2. Select **Data** -> **Add Data Source** and the Attach a New Feature Server dialog opens.
3. Click **Next** to accept the default of Web Feature Server (HTTP).
4. Specify a vector layer to use such as `http://yourserver:8080/erdas-apollo/vector/ATLANTA_VECTOR`.
5. Click **Add** then **Next**. The dialog asks for the Name and title for the new datasource.
6. You do not need to specify a name or title. Click **Next** to continue.
7. If you layer is secure, enter your login and password and click **Next**. Otherwise, click **Finish**. Your new layer appears in the Styles pane.
8. Click the plus sign in the Styles pane to expand the ATLANTA_VECTOR layer.
9. Right-click on **futurelanduse** and select **Create Style**. The Add Style dialog opens.
10. Click **GEOMETRY (type:Polygon)** -> **Next**. The dialog prompts you to select a source for the new style.
11. Click **Create new style** -> **Next**. The dialog prompts you for the type of style.
12. Click the drop-down list and select **Discrete Classification**.
13. Click **Next** and the dialog prompts you for a name for the new style.
14. Enter a name and then click **Properties**. The Style Properties dialog opens.
15. Click the **Classification** tab.
16. Click the drop-down for **Property** and select LANDUSE. Make sure the Type is **Literal**.
17. Click **Populate** and the Populate Style List dialog opens.
18. Change the Stroke Color and Fill Color to suit your needs.
19. Click **OK** and the Style Properties dialog opens.
20. Click **Apply**, then **OK**

21. Click **Apply** (don't see this button) and then **OK** and return to the Style Editor.

Add the Legend

22. In the Styles pane, right-click ATLANTA-VECTOR and select **Dressing Style Properties** from the drop-down menu. The Style Properties dialog opens.
23. Click the **Legend** tab -> **Enabled** -> **Apply** -> **OK** and the dialog closes.
24. In the Styles pane, expand the futurelanduse layer and note that the style layer you just entered displays.
25. Right-click on the style name and click **Add to Preview** from the drop-down list. The Style you set up in Style Properties is applied to the layer and the legend displays.

Select File and Deploy to Folder

26. Select **File** from the menu bar then **Styles -> Deploy to Directory**.
27. Navigate to APOLLO_HOME\config\erdas-apollo\rendering and click **Save**.
28. Click **Yes** to overwrite the files.

Start the Data Manager

29. Start the Data Manager and log in.
30. Expand the vector, right-click on ATLANTA_VECTOR and select **Edit Provider** from the drop-down list. Notice that the ATLANTA_VECTOR tab appears next to Explorer.
31. Click **Flush service cache** and then restart the service.

Start the Style Editor

32. Open the Style Editor.
33. Select **Data -> Add Map Source -> Web Map Server -> Next**.
34. Enter `http://yourserver:8080/erdas-apollo/vector/ATLANTA_VECTOR` in the New field.
35. Click **Add -> Finish**. The new raster layer appears in the Style pane.
36. Expand Vector Data over Atlanta, futurelanduse and the style you created appears.

37. Right-click on the style and select **Add to Preview** to see the style layer in the view.

Sample WFS Requests with Filters

To output GML from a WFS, build and run GetFeature requests, as specified in the OGC WFS 1.0.0 Implementation Specification using "Filter" expressions as specified in the OGC Filter Encoding 1.0.0 Implementation Specification. Note that the Filter Encoding syntax is used for other types of OGC-compliant requests such as performing transactions or locks on a WFS or using SLD in WMS.

This section describes how to build WFS requests using filters. The examples below are based on the BOSTON_ORA database defined earlier. Following each scenario is a GetFeature request with a different type of Filter and an explanation of the content of that request.

Filter by FeatureID

The Boston County Tax Office has examined the city maps that are being published on the Internet using WFS and has discovered that a few of the rivers are misrepresented. The Office informs the data publisher about which river names, IDs and related place names need to be reviewed. The data publisher can use a FeatureID matching request to extract the properties of these river features as follows:

```
<?xml version="1.0" encoding="UTF-8" ?>
<ogcwfs:GetFeature maxFeatures="20"
  xmlns:ogc="http://www.opengis.net/ogc"
  xmlns:ogcwfs="http://www.opengis.net/wfs"
  version="1.0.0"
  service="WFS" >
  <ogcwfs:Query typeName="hydro">
    <ogc:PropertyName>HD_ID</ogc:PropertyName>
    <ogc:PropertyName>GEOMETRY</ogc:PropertyName>
    <ogc:Filter>
      <ogc:FeatureId fid="hydro.337" />
    </ogc:Filter>
  </ogcwfs:Query>
  <ogcwfs:Query typeName="place_names">
    <ogc:PropertyName>PLACES_ID</ogc:PropertyName>
    <ogc:PropertyName>NAME</ogc:PropertyName>
    <ogc:PropertyName>GEOMETRY</ogc:PropertyName>
    <ogc:Filter>
      <ogc:FeatureId fid="place_names.18" />
    </ogc:Filter>
  </ogcwfs:Query>
</ogcwfs:GetFeature>
```

Filter Equal to an Alphanumeric Property

In this GetFeature request, the data publisher issues a query that addresses the "hydro" and the "place_names" feature types where the output is restricted by explicitly providing column names inside <ogc:PropertyName> element tags. In the "hydro" query, the data publisher adds an <ogc:Filter> to output a single feature with the feature Identifier (fid) equal to "hydro.3930". Similarly, the "place_names" output is restricted to the feature with fid equal to "place_names.3135". The output will be a single "hydro" feature and "place_names" feature.

The Boston Tax Office has also asked the data publisher to provide demographic analysis for the Mattapan neighborhood. To do this, the data publisher would build a request on a single feature type that is filtered on an alphanumeric property using the OGC comparison filter for equality - PropertyIsEqualTo.

```
<?xml version="1.0" encoding="UTF-8" ?>
<ogcwfs:GetFeature maxFeatures="20"
  xmlns:ogc="http://www.opengis.net/ogc"
  xmlns:ogcwfs="http://www.opengis.net/wfs"
  version="1.0.0"
  service="WFS" >
  <ogcwfs:Query typeName="place_names">
    <ogc:Filter>
      <ogc:PropertyIsEqualTo>
        <ogc:PropertyName>NAME</ogc:PropertyName>
        <ogc:Literal>MATTAPAN</ogc:Literal>
      </ogc:PropertyIsEqualTo>
    </ogc:Filter>
  </ogcwfs:Query>
</ogcwfs:GetFeature>
```

In the request, the data publisher uses the "place_names" feature type. The <ogc:PropertyIsEqualTo> Filter receives two arguments:

PropertyName: This contains the name of the feature type property to filter against. This can be written either as a string ("NAME") prefixed with the feature_type name ("place_names.NAME") or prefixed with the namespace ("wfs:NAME"). Refer to next example for details on creating a fully-qualified filter property. **Literal:** This is used to compare against the property name.

For more information on the filter operator names and arguments, refer to the "ERDAS APOLLO Server Concepts Guide."

Filter Equal with Namespaces

Feature schema managed by the WFS are typically more complex than the sample data (BOSTON_ORA) environment and could contain a hierarchy of feature types and properties whose definitions are spread over several schema documents. Avoid ambiguity with well defined feature type names or properties. For example, add the namespace prefixes to the feature name types and properties using the same request from the previous example. Each namespace prefix corresponds to a schema document removing the ambiguity related to features with similar names, but in a different hierarchy.

```
<?xml version="1.0" encoding="UTF-8" ?>
<ogcwfs:GetFeature maxFeatures="20"
  xmlns:ogc="http://www.opengis.net/ogc"
  xmlns:ogcwfs="http://www.opengis.net/wfs"
  xmlns:wfs="http://www.ionicssoft.com/wfs"
  version="1.0.0"
  service="WFS" >
  <ogcwfs:Query typeName="wfs:place_names">
    <ogc:Filter>
      <ogc:PropertyIsEqualTo>
        <ogc:PropertyName>wfs:place_names.NAME</ogc:PropertyName>
        <ogc:Literal>MATTAPAN</ogc:Literal>
      </ogc:PropertyIsEqualTo>
    </ogc:Filter>
  </ogcwfs:Query>
</ogcwfs:GetFeature>
```

In this request, the namespace "http://www.ionicssoft.com/wfs" is defined and assigned the prefix "wfs" in the <ogcwfs:GetFeature> attribute. This namespace corresponds to the highest level feature type schema of this WFS (see boston_ora.xsd).

In the remainder of the request, each reference to a feature type is prefixed with "wfs:" and each feature property prefixed with the feature type name. This allows the data publisher to have more than one feature type with the same property name and more than one schema with the same feature type name.

Filter on Two Alphanumeric Properties

The data publisher wants to enable a query in the WFS so the public can locate parks based on specific criteria. In this example, the query is to select parks that are protected areas and larger than 100,000 square meters. The Filter request is on a single feature type "protectedareas" where the SITE_NAME must end with "PARK", and the AREA must be larger than 100,000 square meters. To retrieve the sites that match *both* criteria, the filters are combined with the AND logical operator.

```
<?xml version="1.0" encoding="UTF-8" ?>
<ogcwfs:GetFeature maxFeatures="20"
  xmlns:ogc="http://www.opengis.net/ogc"
  xmlns:ogcwfs="http://www.opengis.net/wfs"
```



```

    version="1.0.0"
    service="WFS" >
<ogcwf:Query typeName="protectedareas">
  <ogc:PropertyName>AREA</ogc:PropertyName>
  <ogc:PropertyName>COUNTY_COD</ogc:PropertyName>
  <ogc:PropertyName>SITE_NAME</ogc:PropertyName>
  <ogc:PropertyName>GEOMETRY</ogc:PropertyName>
  <ogc:Filter>
    <And>
      <ogc:PropertyIsLike>
        <ogc:PropertyName>SITE_NAME</ogc:PropertyName>
        <ogc:Literal>%PARK</ogc:Literal>
      </ogc:PropertyIsLike>
      <ogc:PropertyIsGreaterThan>
        <ogc:PropertyName>AREA</ogc:PropertyName>
        <ogc:Literal>100000</ogc:Literal>
      </ogc:PropertyIsGreaterThan>
    </And>
  </ogc:Filter>
</ogcwf:Query>
</ogcwf:GetFeature>

```

To query for the "protectedareas" feature type, the request specifies the "typeName" attribute in the <ogc:Query> element. Adding <ogc:PropertyName> elements just after the <ogcwf:Query> restricts the output properties to the AREA, COUNTY_CODE, SITE_NAME and GEOMETRY.

The <ogc:Filter> block starts with the <AND> logical operator with two arguments.

The first comparison uses the "PropertyIsLike" operator that makes a pattern comparison. The query is for protected areas whose SITE_NAME ends with "PARK."

The second operator, "PropertyIsGreaterThan," allows comparison with numeric values. The query is for protected areas where AREA is greater than 100,000 square meters.

In the BOSTON_ORA sample database, four feature types meet the first criteria and two of those have an area of more than 100,000 square meters: FRANKLIN PARK and DORCHESTER PARK.

Geometry Filter: Operator BBOX

The Boston Tax Office asked the data publisher to extract information in a given area (around Mattapan) to examine the existing infrastructure. The data publisher knows that Mattapan has a rectangular boundary with specific coordinates and formulates a request to query for highways and place names within the bounding box, or spatial extent, of the area.

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

```

<ogc:wfs:GetFeature maxFeatures="20"
  xmlns:ogc="http://www.opengis.net/ogc"
  xmlns:ogcwfs="http://www.opengis.net/wfs"
  version="1.0.0"
  service="WFS" >
<ogc:wfs:Query typeName="place_names">
  <ogc:PropertyName>NAME</ogc:PropertyName>
  <ogc:PropertyName>COUNTY</ogc:PropertyName>
  <ogc:PropertyName>GEOMETRY</ogc:PropertyName>
  <ogc:Filter>
    <ogc:BBOX>
      <ogc:PropertyName>GEOMETRY</ogc:PropertyName>
      <gml:Box>
        <gml:coordinates>233000,890000
235000,892000</gml:coordinates>
      </gml:Box>
    </ogc:BBOX>
  </ogc:Filter>
</ogc:wfs:Query>
<ogc:wfs:Query typeName="highways">
  <ogc:PropertyName>RT_NUMBER</ogc:PropertyName>
  <ogc:PropertyName>GEOMETRY</ogc:PropertyName>
  <ogc:Filter>
    <ogc:BBOX>
      <ogc:PropertyName>GEOMETRY</ogc:PropertyName>
      <gml:Box>
        <gml:coordinates>233000,890000
235000,892000</gml:coordinates>
      </gml:Box>
    </ogc:BBOX>
  </ogc:Filter>
</ogc:wfs:Query>
</ogc:wfs:GetFeature>

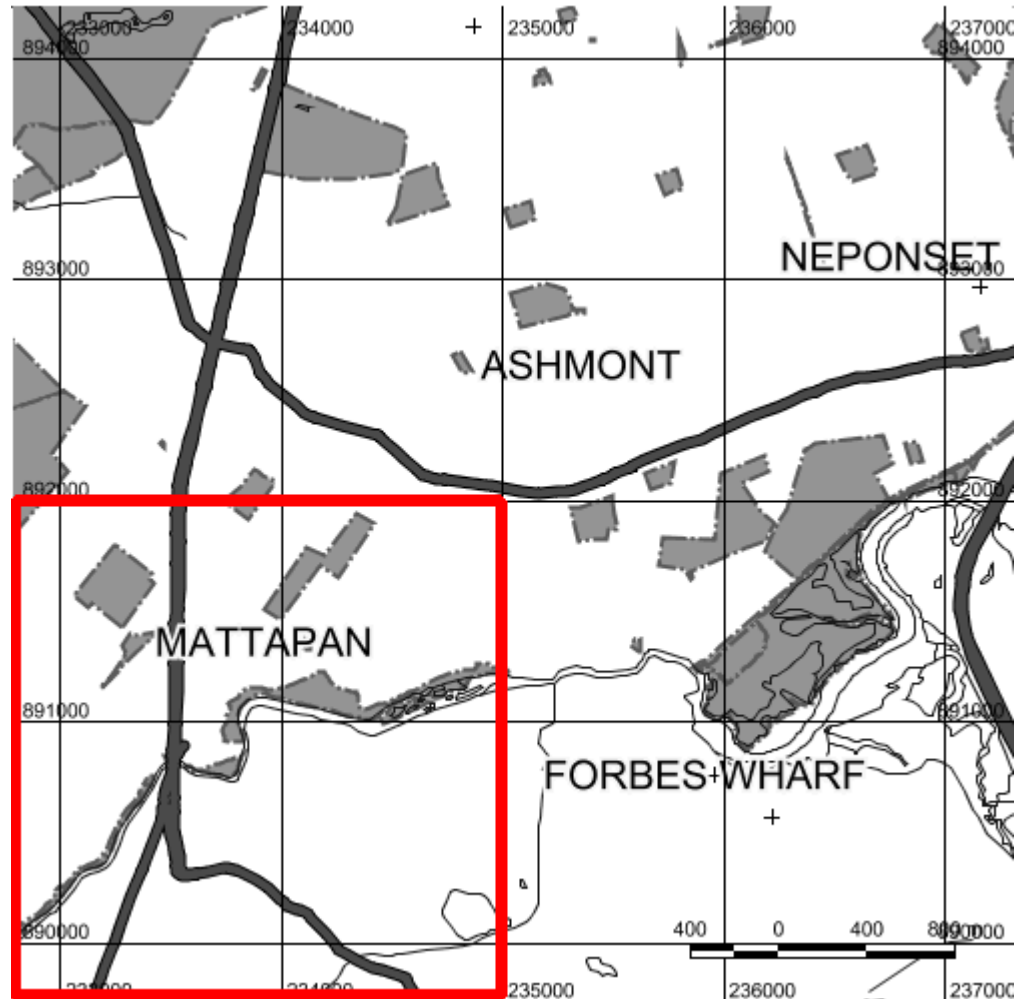
```

This request applies to the "place_names" and "highways" feature types and the output properties are restricted using the <ogc:PropertyName> element.

For each feature type, the <ogc:Filter> block uses the "ogc:BBOX" element to specify the query's spatial operator. This operator is intended to restrict the feature extraction to the given Bounding Box. The first argument must be a geometric property name ("GEOMETRY" in the example) and the second argument must be a <gml:Box> element that provides an extraction rectangle defined by the coordinates for the lower-left and upper-right corners.

This request returns a GML document composed of a place_name, "MATTAPAN", with six highway sections.

Figure 3: A BBOX Filter Request



**Geometry Filter:
Operator Intersects with
a Given Polygon**

The Boston Environmental Office receives a message that commercial ground transportation around Boston has been re-routed and that several of the highways are experiencing increased traffic volume. There is concern that this could pose a threat to some of the environmentally protected areas that intersect the highways. The Office has asked the data publisher to extract those protected areas crossed by major highways.

The data publisher creates a request on a single feature type, highways, with a filter on the geometric property, a LineString, to extract the features intersecting a given polygon.

```
<?xml version="1.0" encoding="UTF-8" ?>  
<ogcwf:GetFeature maxFeatures="20"  
  xmlns:ogc="http://www.opengis.net/ogc"  
  xmlns:ogcwf="http://www.opengis.net/wfs"  
  version="1.0.0"  
  service="WFS" >
```

```

<ogcwf:Query typeName="protectedareas">
  <ogc:PropertyName>SITE_NAME</ogc:PropertyName>
  <ogc:PropertyName>COUNTY</ogc:PropertyName>
  <ogc:PropertyName>GEOMETRY</ogc:PropertyName>
  <ogc:Filter>
    <ogc:Intersects>
      <ogc:PropertyName>GEOMETRY</ogc:PropertyName>
      <gml:Polygon srsName="EPSG:26986">
        <gml:outerBoundaryIs>
          <gml:LinearRing srsName="EPSG:4326">
            <gml:coordinates>233200,891700 233700,891600 234050,892400
234100,893600 233600,893700 233300,892900
233200,891700</gml:coordinates>
          </gml:LinearRing>
        </gml:outerBoundaryIs>
      </gml:Polygon>
    </ogc:Intersects>
  </ogc:Filter>
</ogcwf:Query>
</ogcwf:GetFeature>

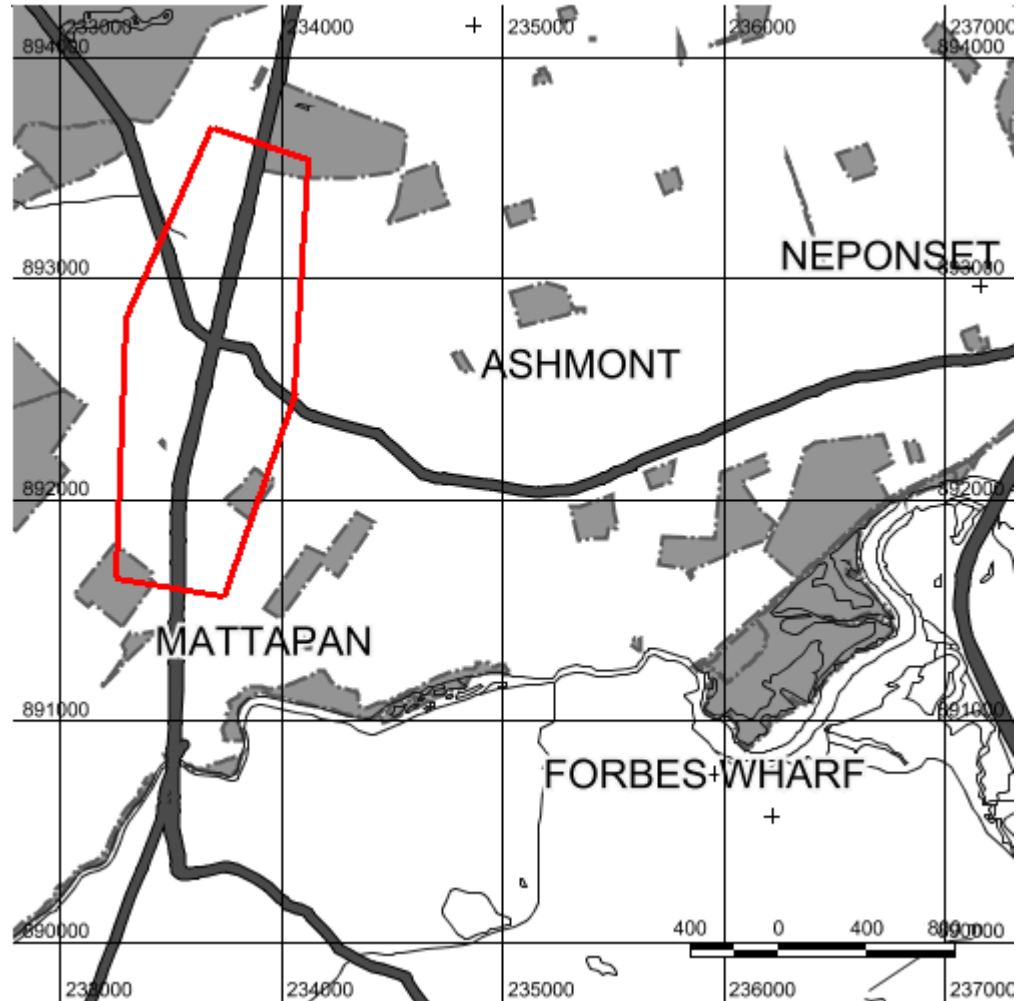
```

This request applies to the "protectedareas" feature type. The SITE_NAME, COUNTY and GEOMETRY properties from the feature type will be extracted.

The query finds the intersection (ogc:Intersects) of the protected areas GEOMETRY and a polygon (gml:Polygon) that surrounds the area of interest. The image below shows the geographic extent of the request.

This request produces a GML document composed of four protected areas features.

Figure 4: A Filter to Intersect with a Polygon



**Geometry Filter:
Operator Beyond a Given
Point**

The data publisher's boss would like to spend the upcoming weekend taking a walk and a swim somewhere in Boston county. He asked the data publisher to find a location in the county that is close to a body of water and the office location.

The data publisher requests the feature type that includes rivers and lakes and applies a filter to the geometric property to extract the features which are not beyond a defined distance from the office.

```
<?xml version="1.0" encoding="UTF-8" ?>
<ogcwf:GetFeature maxFeatures="20"
  xmlns:ogc="http://www.opengis.net/ogc"
  xmlns:ogcwfs="http://www.opengis.net/wfs"
  version="1.0.0"
  service="WFS" >
  <ogcwfs:Query typeName="hydro">
    <ogc:PropertyName>*</ogc:PropertyName>
    <ogc:Filter>
      <ogc:Not>
```

```
<ogc:Beyond>
  <ogc:PropertyName>GEOMETRY</ogc:PropertyName>
  <gml:Point srsNAME="EPSG:26986">
    <gml:coordinates>234500,890000</gml:coordinates>
  </gml:Point>
  <ogc:Distance>500</ogc:Distance>
</ogc:Beyond>
</ogc:Not>
</ogc:Filter>
</ogcwfs:Query>
</ogcwfs:GetFeature>
```

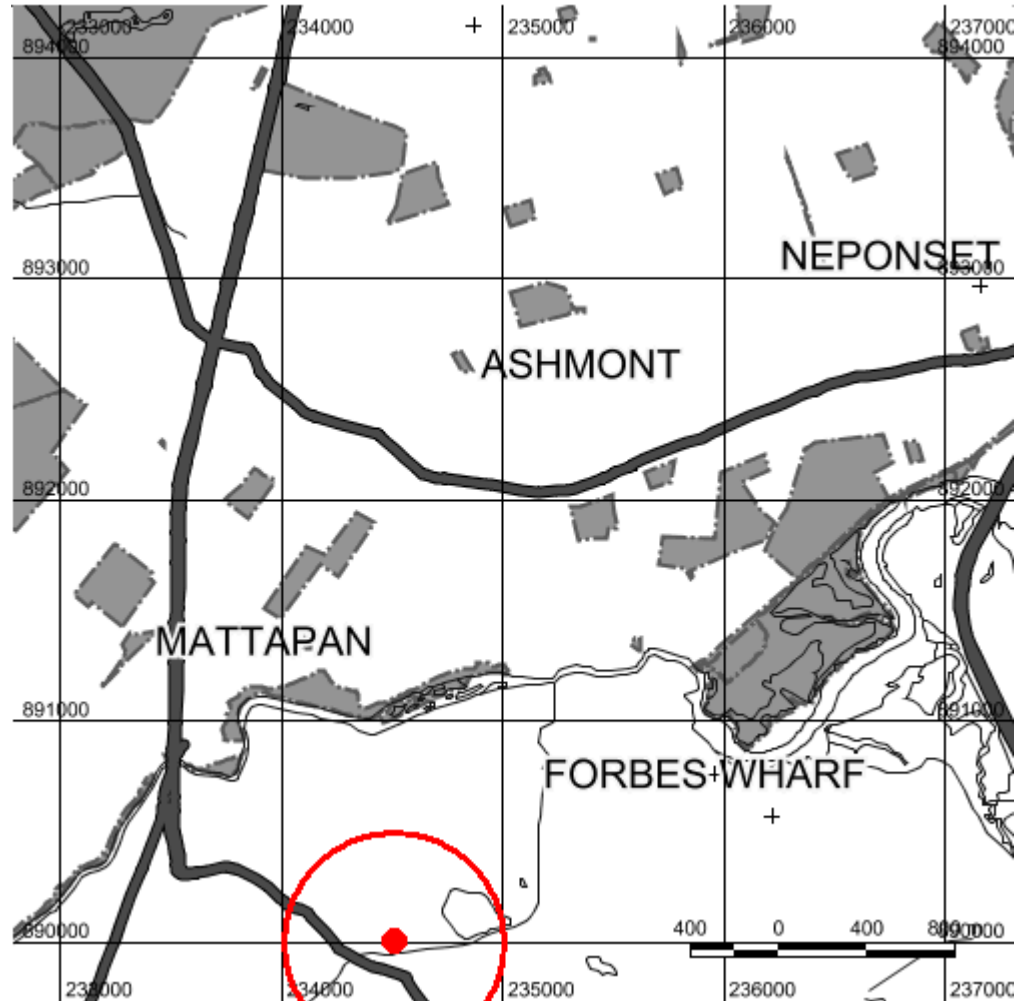
The request applies to the "hydro" feature type. The data publisher can obtain all hydro properties by providing a wild card ("*") in the <ogc:PropertyName> element.

The search consists of locating water bodies that are not beyond (operators ogc:NOT and ogc:BEYOND) 500 meters from a given point. The <ogc:NOT> operator is unary and takes a single argument. The <ogc:BEYOND> operator requires three arguments: the feature geometric property (GEOMETRY), a geometry for the spatial operation (<gml:Point> geometry), and a distance (ogc:Distance). The image below shows the starting point and the area that matches the distance parameter.



The current OGC WFS and Filter Encoding specifications do not support spatial "Joins". Otherwise, the office location geometry could have been used instead of a gml:Point geometry.

Figure 5: A Filter to not be Beyond a Point



Filter combining Spatial and Non-Spatial Operators

There has been an accident in Boston involving a truck carrying hazardous materials. The data publisher has been asked to locate the roads that intersect the accident and prioritize road closures based on road classification.

A Filter is applied to the "roads" feature type using the "PropertyIsBetween" operator. The "Crosses" operator is applied to locate all the roads that are crossed by the spill.

```
<?xml version="1.0" encoding="UTF-8" ?>
<ogcwf:GetFeature maxFeatures="20"
  xmlns:ogc="http://www.opengis.net/ogc"
  xmlns:ogcwf="http://www.opengis.net/wfs"
  version="1.0.0"
  service="WFS" >
  <ogcwf:Query typeName="roads">
    <ogc:PropertyName>STREETNAME</ogc:PropertyName>
    <ogc:PropertyName>CLASS</ogc:PropertyName>
    <ogc:PropertyName>GEOMETRY</ogc:PropertyName>
```

```

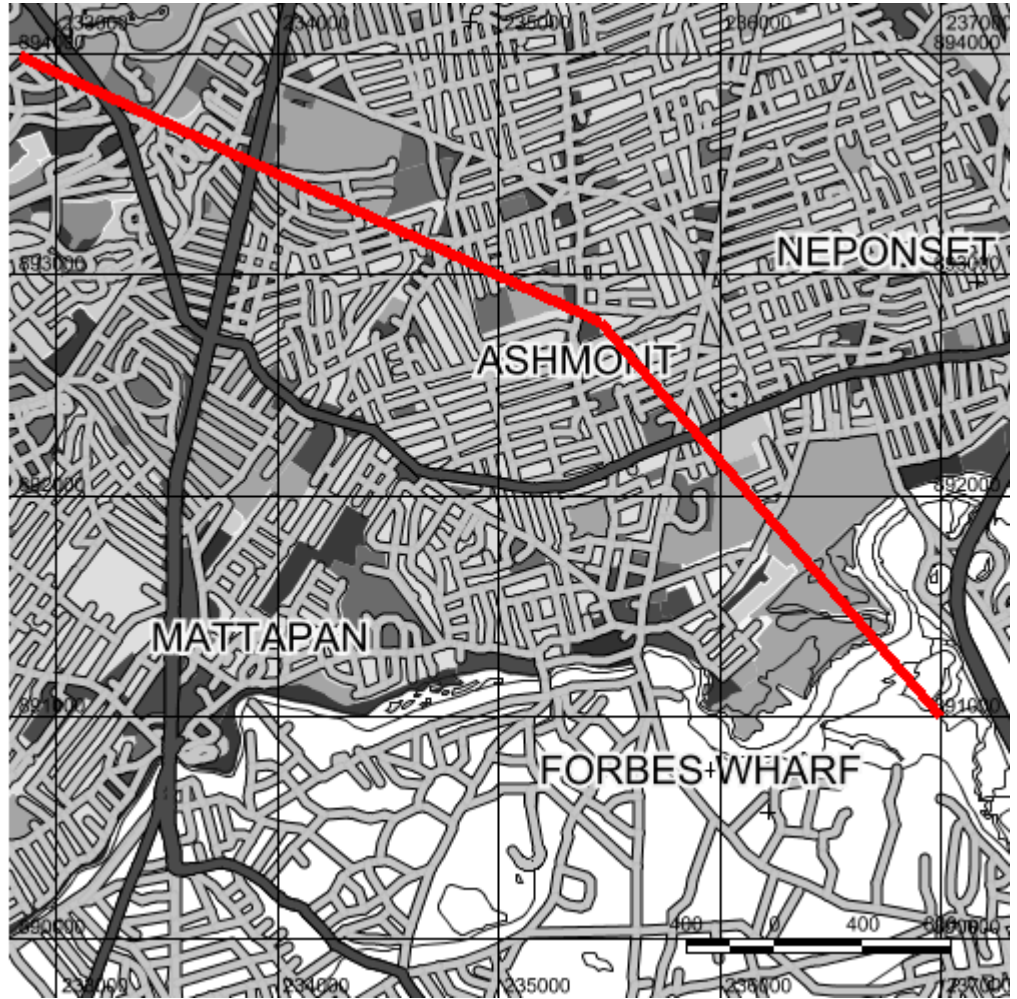
<ogc:Filter>
  <ogc:And>
    <ogc:PropertyIsBetween>
      <ogc:PropertyName>CLASS</ogc:PropertyName>
      <ogc:LowerBoundary>
        <ogc:Literal>2</ogc:Literal>
      </ogc:LowerBoundary>
      <ogc:UpperBoundary>
        <ogc:Literal>3</ogc:Literal>
      </ogc:UpperBoundary>
    </ogc:PropertyIsBetween>
    <ogc:Crosses>
      <ogc:PropertyName>GEOMETRY</ogc:PropertyName>
      <gml:LineString srsName="EPSG:26986">
        <gml:coordinates>232900,894000 235500,892750
        237000,891000</gml:coordinates>
      </gml:LineString>
    </ogc:Crosses>
  </ogc:And>
</ogc:Filter>
</ogcwfs:Query>
</ogcwfs:GetFeature>

```

This request applies to the "roads" feature type. The STREETNAME, the road CLASS and the GEOMETRY of the roads will be extracted.

The search consists of applying the AND operator to two filters. The first filter restricts the roads to Class 2 and 3. The second filter uses the "Crosses" spatial operator. The parameters for <ogc:Crosses> are the feature property name (GEOMETRY) and the geometry to compare against. The result is a LineString representing the Class 2 and 3 roads impacted by the spill.

Figure 6: A Filter to Cross a LineString



Manage Data and Enhance Services

This Chapter provides additional steps for enhancing and managing data and services.

- Restrict the data
- Add a copyright to protect data
- Create advanced filtering in GetMap requests
- Create a custom SRS
- Add functions for more processing by your WFS

Restrict Data

Data often has a commercial and/or legal value and, therefore, protecting data is a major concern for data providers. This section describes a set of configurations that provide restrictions on published data.

- Disabling some of the request types that can be sent to the service
- Hiding columns to show only a subset of the underlying data
- Disabling output formats to prevent the actual data from being extracted

Disable Interfaces

Service Providers over vector data (Shapefile, Oracle) automatically support the OGC-WMS and OGC-WFS interfaces. This means that a user can request maps as well as features in GML or Shapefile.

A data provider may want to allow map output but not deliver the vector data as features, or, conversely, provide access to data but restrict the ability to create a map.

Those restrictions can be activated by setting the "Enabled Interfaces" field on the "Miscellaneous" tab of the Edit Provider feature of ERDAS APOLLO Data Manager. Supported parameter values are "wms", "wfs" or both. This disables the associated set of request types: for "wms", the WMS GetCapabilities, GetMap and GetFeatureInfo and for "wfs", the WFS GetCapabilities, DescribeFeatureType, GetFeature, LockFeature and Transaction.

WFS Operations:

In the mapping file of the WFS Provider, the <Operations> tag in the <Info> section of each feature type contains the list of supported WFS operations. The value "*" enables all operations, query and transactions. Operation values are Query, Insert, Update, Delete and Native. Below is an example for "wfs:roads" from an Oracle provider mapping file.

```
<!--Info for type wfs:roads-->
<Info name="wfs:roads">
<Operations>Query,Insert</Operations>
<SRS>EPSG:26986</SRS>
<BoundingBox SRS="EPSG:26986" minx="227317.38" miny="889948.26"
maxx="238669.29" maxy="901300.18"/>
</Info>
```

WMS Operations:

By default, a WFS Provider supports the three basic WMS request types including GetFeatureInfo. To restrict the allowed request types, set the <Queryable> tag to "false" in the <Info> tag of the feature type. This sets the "queryable" attribute to "0" in the WMS capabilities document and GetFeatureInfo requests on that feature type are denied.

Example: <Queryable>>false</Queryable>

Hiding Columns

When a vector provider's mapping file is created using SQL mapping or the FromSQLGenerator tool, the mapping between the feature type attributes and the table columns is often one-to-one. However, it is possible to hide some of the columns to prevent disclosure of useless or crucial information or produce lighter results.

To achieve this, create explicit mapping and schema files:

- In the Mapping file, remove the corresponding <Element> and <Geometry> lines for the mapping of any columns that should be hidden.
- In the Schema file, remove the declaration of the properties that are no longer mapped with a column name.

Submit a DescribeFeatureType request to see that the removed properties are no longer visible. A GetFeature request will confirm that the corresponding column values are hidden.

Disable Output Formats

When the provider is configured, the servlet will automatically publish, in the WMS Capabilities document a set of formats in which the maps or feature information can be requested. This set of formats can vary depending on the underlying data type: raster, vector, or coverage.

There may be a requirement to remove some of the formats and reject requests asking for those formats. This can be done easily by setting the "Hidden Map Formats" and "Hidden Info Formats" fields in the "Styling Info" tab page of the provider definition. The values are a comma-separated list of format names. For raster formats, the possible values are GIF, JPEG, PNG, SVG, TIFF, WBMP and XBMP. These correspond respectively to the mime-types image/gif, image/jpeg, image/png, image/svg+xml, image/tiff and GeoTIFF, image/vnd.wap.wbmp and image/x-bmp. For information formats, the values are GML, HTML, TXT and XML, and correspond respectively to the mime-types application/vnd.ogc.gml, text/html, text/plain and text/xml.



If all the possible formats are disabled, the service capabilities document will become invalid against the DTD or schema. In addition, the client applications could behave strangely when querying the service.

Add a Copyright or Watermark

When maps are produced by the service, the owner of the service could expect the intellectual property of that map to be preserved. Several ways exist, in ERDAS APOLLO, to burn a text or an image into a map or GML document.

Access the ERDAS APOLLO Data Manager, highlight the Service Provider in the Explorer view, right-click, then select **Edit Provider** -> **Security Parameters** and enter the text in the Copyright field. The text you enter will appear in the upper left corner of the image if you request a map, or in the header of your GML document if you extract features. For this solution, you can only use text and you cannot change the location or the appearance of the text.

If you are managing vector data layers and want a raster image to be added to the output, use the map dressing function of the ERDAS APOLLO Style Editor and follow these steps.

1. Navigate to <Apollo_Home>\config\erdas-apollo\rendering\mapdressing\collection\northarrow and copy/paste the arrow folder.
2. Rename the copied folder to "copyright".

3. Edit the SVG.prop in it by changing the symbolName value with the one you have chosen.
4. Navigate to <Apollo_Home>\tools\styleeditor\data\Symbols and retrieve the symbol you have chosen to be added as watermark to the context.
5. Paste the file in <Apollo_Home>\config\erdas-apollo\rendering\mapdressing\symbol\lib1.
6. Add the MAPDRESSING provider in the ERDAS APOLLO Style Editor (<http://localhost/erdas-apollo/map/MAPDRESSING>).
7. To make your styles persistent and available to other clients, deploy them on the server. (**File -> Styles -> Deploy to Directory** and choose the <Apollo_Home>\config\erdas-apollo\rendering folder).
8. Add the vector and raster layers on which you want to see the watermark.
9. Export your changes as a context (File -> Export Context...).
10. Edit the mappresentation_layers.xml (located in <APOLLO_HOME>\APOLLOversion_pro_gm\config\erdas-apollo\providers\map) and add the following line under the "<layer name="northarrow"" tag:


```
<style name="copyright" title="Copyright watermark" />
```
11. Look for the watermark when adding this context in the webclient.

Add a CRS to WCS GIO Decoder Framework

ERDAS APOLLO Server is currently composed of different components that use their own projection engines to support different spatial reference systems. It uses an OGC standard XML file-based projection engine and the WCS GIO decoder plug-in, which provides image decoding capability using the ERDAS IMAGINE raster engine, is based on ERDAS IMAGINE (EPRJ) projection engine.



As of APOLLO 10.0, the GIO decoders are not available on Linux platforms.

Datasets that contain EPRJ-based projection representation and decoded via WCS GIO decoders are translated to ERDAS APOLLO projection representation. The bridge between these engines is the European Petroleum Survey Group (EPSG) code, which is the standard for identifying a CRS. Using EPSG code has the advantage because they describe CRS unambiguously and let you define your own by extensions.

The difference between the engines come into play in the following cases:

- **Add an EPSG Code:** The system is not preconfigured with a valid EPSG code that you can use.
- **Define a CRS:** You have a coordinate system with all its parameters that is not defined in EPSG, and you want to extend the system to recognize this new coordinate system.

ERDAS IMAGINE Projection Engine

EPRJ is a very mature projection engine that supports quite a number of CRSs. EPSG support was recently added to it using a translation library that converts EPSG codes to their equivalent EPRJ CRS definitions and vice-versa. This translation from EPRJ to EPSG is used whenever any data or metadata is requested by WCS GIO decoders for all the raster formats they support in ERDAS APOLLO.

To extend the system review the following ERDAS IMAGINE projection configuration files.

- mapprojection.dat
- epsg.plb
- spheroid.tab
- units.dat
- sptable.tab

All of these files are located in this directory:

<APOLLO_HOME>\native\raster\etc\projections



For an explanation of these files, see [ERDAS IMAGINE Projection System Configuration](#) on page 237.

Administrators need to especially understand the epsg.plb file. It is used as the translation table from EPSG to EPRJ.

Projection Entry File Details

The actual projection entry from the epsg.plb file looks like this:

```
"NAD83 / Wisconsin Transverse Mercator (3070)" {  
  INTERNAL 9 "GRS 1980" "NAD83" 0  
  2:9.99600000000000004E-001&nbsp; 4:-  
  1.5707963267948966E+000
```


Table 2: Projection Entry Translation Table

#	Name	Description
6	Datum	Your datum name (again from spheroid.tab) goes here (including quotes).
7	Zone number	This is applicable only to the UTM and State Plane projections and is specified in sptable.tab. For everything else, this should be zero.
8	Projection Specific Parameters	The number of parameters varies with the projection and should match with the number of parameters in mapprojections.dat for that particular projection.
9	Units	This should be one of the entries in units.dat (including quotes).



Please note the following:

- *The projection specific parameters (#8 in the above table) have varying degrees of tolerance when EPRJ compares the values to find the right projection. You must provide enough digits of precision after the decimal point to have a low tolerance.*
- *For spheroids:*
 - *The major axis, minor axis, and radius parameters have a tolerance of 1.0e-05.*
 - *The remaining parameters have a tolerance of 1.0e-09.*
- *For all datum parameters, the tolerance is 1.0e-09.*
- *For all EXTERNAL projections and INTERNAL projections (#3 and #4 in the above table) greater than 3:*
 - *Any angular value has a tolerance of 1.0e-12.*
 - *Any linear distance has a tolerance of 1.0e-04.*
 - *Any scale factor has a tolerance of 1.0e-10.*
 - *Any integer value like NADCODE, StatePlane zone, and UTM zone for instance, the tolerance is 0.1.*
- *When looking at parameters that represent origin, keep in mind that EPRJ's internal angular representation is expressed in radians, not decimal degrees.*

ERDAS APOLLO Server defines coordinate reference systems in a series of XML files. By default, ERDAS APOLLO ships with a significant number of coordinate reference systems including most of the nondeprecated EPSG codes. However, it allows you to define additional coordinate reference systems. The structure and content of XML files have been extensively documented in the SRS Coordinate Transformations chapter.

Add an EPSG Code

The CRS to add exists in both ERDAS APOLLO Server and EPRJ, but not in an EPSG-to-EPRJ translation module. This might be because only a subset of EPRJ is present in the ERDAS IMAGINE EPSG translation library.

The steps for adding EPSG code support are:

1. Open the epsg.plb file using any text editor.
2. Add the EPSG code you want supported as shown in Projection Entry File Details above.
3. Save the file and restart the server.

You can now crawl data with this new projection.

NOTE: The dataset might fail to register even after setting the correct EPSG translation because the projection parameters inside the dataset don't exactly match the tolerances of comparison for any one parameter. In that case, please follow the steps below for Defining a New CRS without Step 3. "Add the CRS/projection information to ERDAS APOLLO Projection Engine".

Define a CRS

Defining a new CRS from scratch is more complicated. The administrator should have a good understanding of the following:

- How to define a CRS and its parameters.
- Future additions to the EPSG dataset, user-defined codes should be above the EPSG integer code limit of 32,767).
- How to add CRS entries to ERDAS APOLLO Platform system.

For more information, please read carefully the information in the SRS Coordinate Transformations chapter. Defining a new CRS requires you to know all the parameters that define a CRS and make changes to the ERDAS APOLLO projection engine.



If you are changing any of the existing .xml files, it is strongly suggested that you first back up the original files and keep them in a safe place.

To add a new CRS to the ERDAS APOLLO Projection Engine, follow these steps:

1. Stop the JBoss application server.
2. Pick a user-defined EPSG code to use (should be > 32,767) if the CRS doesn't have any EPSG code.
3. Add the CRS/projection information to the ERDAS APOLLO projection engine, as follows:
 - a. Create an XML file to include the new user-defined CRS per the instructions listed in SRS Configuration Parameters and save it as usersref.xml.
 - b. Copy the XML file to:
\$JBOSS_HOME\server\default\deploy\erdas-apollo.ear\lib\cots-srs-xxx.jar\com\ionicsoft\sref\impl\resource
 - c. Create the folders if they do not already exist. An example of usersref.xml is:

```
<?xml version="1.0" encoding="utf-8" ?>
<SREF>
  <PROJCS ID="26767" NAME="NAD27 / Georgia West">
    <UNIT ID="9003" />
    <GEOCS ID="4267" />
    <PROJECTION NAME="Transverse Mercator">
      <PARAMETER NAME="central_meridian" VALUE="-
        84.16666666666666"/>
      <PARAMETER NAME="false_easting"
        VALUE="500000"/>
      <PARAMETER NAME="false_northing"
        VALUE="0.0"/>
      <PARAMETER NAME="latitude_of_origin"
        VALUE="29.999999999999996"/>
      <PARAMETER NAME="scale_factor"
        VALUE="0.9999"/>
    </PROJECTION>
  </PROJCS>
</SREF>
```

4. Restart the JBoss application server.
5. Create an aggregate with the new EPSG code defined as its CRS.
6. Start a crawler job with this aggregate as the root.

NOTE: All new datasets found by this crawler should now have this new EPSG code as their CRS. If cataloging fails, make sure that the above steps are followed correctly and review the changes you made to determine whether you specified all the parameters that are needed.



If the GIO decoders are the ones configured for this file extension (in the decoder.txt file), then this solution assumes that they can recognize the image as referenced but cannot create an EPSG code. If not recognized as a referenced image by GIO, please change the decoder to the GDAL and try again.

Filter in a GetMap

Filtering functionality is currently supported in the Web Feature Service (See [The Web Feature Service \(WFS\)](#)). Additional functionality has been added by ERDAS to allow the use of power filtering to define what data will be extracted in a request in the context of WMS requests. ERDAS servlets that publish vector data support WMS GetMap and WMS GetFeatureInfo requests with an additional "FILTER=<value>" parameter, where <value> is the XML syntax for the OGC Filter Encoding 1.0.0 specification. The exact set of Filter operators and functions available is described in the OGC Filter Encoding 1.0.0 specification. The subset of the specification supported by a given provider is published in its WFS capabilities document. By extension, a WMS GetMap or GetFeatureInfo request built on that same service can use those filters.

Example of Filter Operations Declared in the WFS Capabilities

```
...
<Filter_Capabilities xmlns="http://www.opengis.net/ogc">
  <Spatial_Capabilities>
    <BBOX/>
    <Equals/>
    <Disjoint/>
    <Intersect/>
    <Touches/>
    <Crosses/>
    <Within/>
    <Contains/>
    <Overlaps/>
    <Beyond/>
  </Spatial_Capabilities>
  <Scalar_Capabilities>
    <Logical_Operators/>
    <Comparison_Operators>
      <Simple_Comparisons/>
      <Like/>
      <Between/>
      <NullCheck/>
    </Comparison_Operators>
    <Arithmetic_Operators>
```

```

<Simple_Arithmetic/>
<Functions>
  <Function_Names>
    <Function_Name nArgs="1">Upper</Function_Name>
    <Function_Name nArgs="1">Lower</Function_Name>
    <Function_Name nArgs="3">Distance</Function_Name>
    <Function_Name nArgs="1">Score</Function_Name>
  </Function_Names>
</Functions>
</Arithmetic_Operators>
</Scalar_Capabilities>
</Filter_Capabilities>
...

```

For example, using the "roads" feature type defined in the BOSTON_SHAPE WFS, a WMS request for a set of road features where the STREET_NAME contains "Avenue" could contain the following filter definition:

```

<ogc:Filter>
  <ogc:PropertyIsLike>
    <ogc:PropertyName>STREET_NAME</ogc:PropertyName>
    <ogc:Literal>Avenue</ogc:Literal>
  </ogc:PropertyIsLike>
</ogc:Filter>

```

A WMS GetMap request using the filter would look like the example below. Note that the column-like syntax used below is to make it readable. The actual syntax to use in a GetMap request would be on a single line, with '&' as a separator between parameters.

```

http://localhost:8080/erdas-apollo/vector/BOSTON_SHAPE?
VERSION=1.1.1
REQUEST=GetMap
WIDTH=400
HEIGHT=400
SRS=EPSG:26986
BBOX=233000.,890000. 235000.,893000.
LAYERS=roads
STYLES=default
FORMAT=image/png
BGCOLOR=0xFFFFFFFF
TRANSPARENT=TRUE
EXCEPTIONS=application/vnd.ogc.se_xml
FILTER=<ogc:Filter><ogc:PropertyIsLike><ogc:PropertyName>STREET
_NAME</ogc:PropertyName>
<ogc:Literal>River</ogc:Literal></ogc:PropertyIsLike></ogc:Filter>
er>

```



For simple filters, the WMS standardized "Dimension" mechanism is recommended. (See OGC WMS 1.3.0 specification.)



For complex filters, as the FILTER=<value> mechanism is proprietary to ERDAS servlets, it is recommended to use the SLD-based behaviors, as described in the Portrayal Configuration chapter.

Add User Functions

You can extend the processing available in ERDAS Vector services by adding functions called "User Functions". There are two types of functions that can be added to a WFS.

- A Java class that will apply post-processing on the feature set extracted from the data source
- A tag that will publish a datasource procedure or function so that it can be explicitly requested by the user. Generally, it is used for Oracle-based providers in which the WFS calls an Oracle PL/SQL function at the data extraction stage of the query

Add a Java class Function

The steps needed to add a Java class for post-processing are detailed below. They are based on a sample function named SummaryFunction which role is to truncate a text field when the length of the field is greater than a given threshold.



The ERDAS APOLLO distribution provides a couple of Java functions (package com.ionicsoft.wfs.function). The first one is called GeneralizeFunction and is able to generalize a given geometry (to reduce the number of coordinates). The second one is called UpperFunction and is able to convert a String property in uppercase.



Java functions are currently applicable only to provider types exposing a ResultSet-type structure (Oracle, PostgreSQL, SQL Server, ArcSDE). The AST-tree providers (such as the Shapefile provider) do not support Java functions.

1. The first thing to do is to develop a Java class according to the following guidelines:

- If the function does not require parameters, provide the default constructor

```
public SummaryFunction()
{
    m_length=10;
}
```

- If the function requires parameters, it is possible to provide constructors of the form <constructor>(type1 P1, type2 P2....). An example could be:

```
public AnotherFunction(String target, int length)
{
    this.m_target=target;
    this.m_length=length;
}
```

- Create a set of "evaluate" methods. Each evaluate methods must be of the form: ResultType evaluate (type1 P1, type2 P2...), where ResultType can be Object, a simple type or a GeometryType. The parameters types must be a simple type, a simple object type or a geometry type. A simple type is a int, double ... and a simple object type is Integer, Double, String, Date,

```
2
public String evaluate(String target)
{
    return evaluate(target,m_length);
}
2
public String evaluate(String target, int length)
{
    if(target.length()>length)
    {
        return target.substring(0,length)+"...";
    }
    else
    {
        return target;
    }
}
```

2. Once you have coded your function, you have to compile the class and copy the generated class file in the classpath of the WFS service (i.e. by adding the class in the <APOLLO_HOME>/webapps/erdas-apollo/webapp/WEB-INF/classes directory) and rebuild the webapp with the ant command in the <APOLLO_HOME>/webapps/erdas-apollo.

3. The last step is to declare the function in the providers.fac file and the end of the Configuration tag:

```
<CONFIGURATION>

.....

<REGFUNC ID="Generalize"
JCLASS="com.ionicsoft.wfs.function.GeneralizeFunction" />
<REGFUNC ID="Summary"
JCLASS="com.ionicsoft.test.wfs.functions.SummaryFunction">
<PARAM NAME="length" VALUE="5" />
</REGFUNC>

</CONFIGURATION>
```

4. The Java functions can be used in GetFeature requests, but only in the set of output PropertyName tags. They cannot be used in the Filter clause, as they do apply to the feature set after it is extracted from the data source. Moreover, they do not appear in the WFS capabilities document.

Add a Datasource Function

A datasource function is the second way to add processing capabilities to a WFS.

1. In case of an Oracle provider, the example consists in adding an Oracle PL/SQL function to the WFS. The function has the same purpose as the SummaryFunction Java class. Its PL/SQL equivalent can be:

```
create or replace function summarize(target varchar2)
return varchar2 is
begin
  if length(target)>5
  then
    return substr(target,0,5)||'...';
  else
    return target;
  end if;
end;
/
```

2. A "UserFunction" tag has to be added in the mapping file in order to map the PL/SQL function with a WFS Function name (it will appear in the WFS capabilities), and to describe the parameters and return type. For the "summarize" procedure, the function is declared as taking a String as single argument and returns a String. The mapping tags can be:

```
<UserFunction name="Summarize" nameSQL="summarize">
  <Parameter type="string" />
  <Return type="string" />
</UserFunction>
<Mapping>
```

```
<SQL name="wfs:roads">
...
```



This tags add a "Summarize" function to the WFS functions list available in the WFS capabilities:

```
<Functions>
  <Function_Names>
    <Function_Name nArgs="1">Upper</Function_Name>
    <Function_Name nArgs="1">Lower</Function_Name>
    <Function_Name nArgs="3">Distance</Function_Name>
    <Function_Name nArgs="1">Score</Function_Name>
    <Function_Name nArgs="1">Summarize</Function_Name>
  </Function_Names>
</Functions>
```

3. Now you can perform the following WFS query:

```
<?xml version="1.0" encoding="UTF-8" ?>
<ogcwfs:GetFeature maxFeatures="20"
  xmlns:ogc="http://www.opengis.net/ogc"
  xmlns:ogcwfs="http://www.opengis.net/wfs"
  version="1.0.0" service="WFS" >
  <ogcwfs:Query typeName="roads">
    <ogc:PropertyName>STREETNAME
    <ogc:Function name="Summarize">
      <ogc:PropertyName>STREETNAME</ogc:PropertyName>
    </ogc:Function>
  </ogc:PropertyName>
</ogcwfs:Query>
</ogcwfs:GetFeature>
```

The resulting FeatureCollection contains truncated property values:

```
<wfs:STREETNAME>MARIET...</wfs:STREETNAME>
```

Instead of:

```
<wfs:STREETNAME>MARIETTA BLVD</wfs:STREETNAME>
```


You can also performs filtering using the user functions:

```
<?xml version="1.0" encoding="UTF-8" ?>
<ogcwfes:GetFeature maxFeatures="20"
  xmlns:ogc="http://www.opengis.net/ogc"
  xmlns:ogcwfes="http://www.opengis.net/wfs"
  version="1.0.0"
  service="WFS" >
<ogcwfes:Query typeName="roads">
  <ogc:Filter>
    <ogc:PropertyIsEqualTo>
      <ogc:Function name="Summarize">
        <ogc:PropertyName>STREETNAME</ogc:PropertyName>
      </ogc:Function>
      <ogc:Literal>MARIET...</ogc:Literal>
    </ogc:PropertyIsEqualTo>
  </ogc:Filter>
</ogcwfes:Query>
</ogcwfes:GetFeature>
```



The WFS follows two rules in order to find User Functions:

- It first searches a Java function declared in REGFUNC tags, based on the parameter count and names.
- Then it searches for a data source function based on the name, in UserFunction tags in the mapping file.

The Java functions do not appear in the WFS capabilities and cannot be use in the <Filter> part of a GetFeature request. The data source functions appear in the capabilities and can serve in a Filter.

Set Up a WFS with GML2 Objects

This section describes the general way of setting up a Web Feature Server based on the GML 2.1 application schema. It automatically exposes the data stored in Oracle as a WFS 1.1.0, and it opens the door to defining schemas using one or more of the new GML2 types. ERDAS APOLLO supports some of those types and this section explains how to set up and query such a WFS for the new GML2 geometries, Measurements and Units, and for Temporal operations.

You must install ERDAS APOLLO and services must be accessible via the `http://myhost:80` URL.

Steps to build an Oracle WFS on top of those sample GML2 data:

1. Initialize the Oracle data set using the files in
<APOLLO_HOME>/data/erdas-apollo/db/oracle/satellite.
2. Run the `createoracle.sql` SQL script to create the table and indexes.
3. Launch the ERDAS APOLLO Data Manager and log in with the default admin/apollo123 username and password.
4. Right-click on the **vectors** node and choose **Create service**.
5. In the Service Creation Wizard panel, select **Vector data** as the service type and **Database** as the data source type. Click **Next**.
6. Select **Oracle service** as the service type. Click **Next**.
7. Choose the same database connection parameters as those used in [step 1](#). host, port, user, password, SID,... . The default SRS value of "EPSG:4326" is the right one. Click **Next**.
8. In the **Select the main service properties** panel, enter "GML3EXT" as **service name** and "GML3 Satellite" as **Abstract** and **Title**, then click **Next**.
9. In the next panel, uncheck the **Generate Types and Mapping** box. After a few seconds, a new WFS service called GML3EXT is added under the Vector tree node. Also, an error message displays because the service does not yet have the appropriate settings; this is normal.
10. Right click on the node name and choose **Edit Provider**. The right pane of the Data Manager displays the properties of the newly created service.
11. Select the **Data Source** tab. For the Mapping File property, enter satellite.xml.
12. In the **Types Schema** field, enter "satellite.xsd". As those two files exist and are beside the main configuration file, they will be used. Save your changes, and answer Yes to "Do you want to save your modification to a catalog?".
13. Click the **Get Capabilities** link to see the Capabilities document (if you see some kind of error please go back and check all of your input parameters and retry).

Insert Data into the Provider

As soon as the provider is up and running, you can use the WFS Loader tool to insert data into the provider. More detail on that tool is given in [ERDAS APOLLO Tools and Viewers](#). Follow these steps to insert the data using that tool.

1. Copy the whole set of xml files found in `<APOLLO_HOME>/data/erdas-apollo/db/oracle/satellite` into the `<APOLLO_HOME>/tools/ows` directory.
2. Execute `runwfsloader ./gml3.xml` for the script to execute. It will produce a `gml3log0` file which contains the log of your operation. If that file does not contain any "Exception" word and if the table contains 140 records, the inserts succeeded. If not, check the messages in the `gml3log0` file.

The service is now ready to serve the various GML3 objects. The following sub-sections describe how to use them.

Curves, Surfaces, Rings

Using the previously created provider (GML3EXT), it is now possible to use the GML application schema for GML3 geometries, and to query them using spatial operators.

First, you can notice that the schema publishes, for the "satellite" feature type, a "GEOMETRY" property with type "gml:GeometryAssociationType". This type is a generic one, because the inserted geometries are of various types. Executing a GetFeature on the service or viewing the various `<geometry>-request.xml` files used to insert them shows the variety of geometry types inserted, namely:

- ARC SEGMENT, having the GML3 structure: `Curve/segments/Arc` (with 3 coordinates)
- MULTIPLEARCSEGMENT, having the GML3 structure: `Curve/segments/Arc` (one or more, with three coordinates)
- POLYGON, having the GML3 structure: `Surface/patches/PolygonPatch/exterior(or interior)/LinearRing` (one or more)
- TRIANGLE, having the GML3 structure: `Surface/patches/Triangle/exterior/LinearRing` (with four points)
- RECTANGLE, having the GML3 structure: `Surface/patches/Rectangle/exterior/LinearRing` (with five points)
- RING, having the GML3 structure: `Ring/curveMember` (one or more)/`Curve/segments/Arc` (with three coordinates)
- CIRCLERING, having the GML3 structure: `Ring/curveMember/Curve/segments/Circle` (with three coordinates)

To restrict the set of geometry types accepted by your feature type, change its declared type from `gml:GeometryAssociationType` to one, more explicit, among: `gml:CurvePropertyType`, `gml:SurfacePropertyType` and `gml:RingPropertyType`.

You can also use GML3 geometries in your GetFeature requests, like in the example below.

```
<?xml version="1.0" encoding="UTF-8" ?>
<ogcwf:GetFeature maxFeatures="200"
  xmlns:ogc="http://www.opengis.net/ogc"
  xmlns:ogcwf="http://www.opengis.net/wfs"
  xmlns:wfs="http://www.ionicssoft.com/wfs"
  xmlns:gml="http://www.opengis.net/gml"
  version="1.1.0"
  service="WFS" >
  <ogcwf:Query typeName="wfs:satellite">
    <ogc:Filter>
      <ogc:Intersects>
        <ogc:PropertyName>GEOMETRY</ogc:PropertyName>
        <gml:Surface>
          <gml:patches>
            <gml:PolygonPatch>
              <gml:exterior>
                <gml:LinearRing>
                  <gml:pos>-165.0 -33.0</gml:pos>
                  <gml:pos>-127.0 -55.0</gml:pos>
                  <gml:pos>172.0 62.0</gml:pos>
                  <gml:pos>-165.0 -33.0</gml:pos>
                </gml:LinearRing>
              </gml:exterior>
            </gml:PolygonPatch>
          </gml:patches>
        </gml:Surface>
      </ogc:Intersects>
    </ogc:Filter>
  </ogcwf:Query>
</ogcwf:GetFeature>
```

Measurements, Units of Measure

Using the same provider (GML3EXT) as in the previous example, we can manage units of measures and measurements.

In the example, the "TEMPERATURE" property is of `gml:MeasureType` type. It has a value and a "uom" attribute (Unit Of Measure). The unit value is taken from one or more dictionaries of units. ERDAS APOLLO WFS comes with a set of predefined units, namely Angles in Degrees (deg), Angles in Grads (grad), Angles in Radians (rad), Distance in Kilometers (Km), in Meters (m), in Centimeter (cm), in Millimeter (mm) and in Inches (in). If you want to define additional units, either new Basic ones or others based on an existing one, you need to create an XML document to hold that definition, and reference that document in your mapping file.

Besides the WFS providers.fac available in the distribution, you will find a units.xml file, which declares the Celsius (symbol "Cel") temperature unit as a Base unit, and the Fahrenheit (symbol "Far") temperature unit in relation with the Celsius one. The file is a valid dictionary in the sense of the GML3 Units Dictionary specification (chapter 16 of OGC GML 3.1.0 Specification).

In order for your TEMPERATURE property type to be linked with those unit definitions, you first need to add, in the satellite.xml mapping file, a reference to the units XML document. It is done by adding the line:

```
<UnitDefinition>units.xml</UnitDefinition>
```

Then, you declare the association between your feature type, its property and the default unit. It is done either by adding a "measure" attribute to the <Element>, or by adding the line:

```
<UnitAssociation type="wfs:satellite" name="TEMPERATURE"
measure="Cel"/>
```



In the mapping file the association has to be declared AFTER the unit and the property are known. We recommend the <UnitAssociation> element to be set at the end of the mapping file.

Having done so, each time a request or an insert is done for that property, the system will check the "uom" given in the query, and will possibly convert the value to the default one if different.

For a GetFeature, the GML output will display the TEMPERATURE property as:

```
<wfs:TEMPERATURE uom="Cel">67.17497699418969</wfs:TEMPERATURE>
```

For an Insert transaction or a GetFeature using that property in a filter, the uom given will be taken into account. We will build an example request to obtain the count of features for which the Celsius temperature is greater than 0. We can write it easily, and the response will give 120 features. But as the internal unit is Celsius, it does not illustrate unit conversion. So, we will change the request so that the requested temperature is given as 32 degrees Fahrenheit, which corresponds to 0 degree Celsius. The query is shown below, and you can verify that the response is 120, like before.

```
<?xml version="1.0" encoding="UTF-8" ?>
<ogcwfs:GetFeature maxFeatures="1000"
  xmlns:ogc="http://www.opengis.net/ogc"
  xmlns:ogcwfs="http://www.opengis.net/wfs"
  version="1.1.0" service="WFS"
```

```

        resultType="hits" >
<ogcwf:Query typeName="satellite">
  <ogc:PropertyName>TEMPERATURE</ogc:PropertyName>
  <ogc:Filter>
    <ogc:PropertyIsGreaterThan>
      <ogc:PropertyName>TEMPERATURE</ogc:PropertyName>
      <ogc:Literal uom="Far">32</ogc:Literal>
    </ogc:PropertyIsGreaterThan>
  </ogc:Filter>
</ogcwf:Query>
</ogcwf:GetFeature>

```

Temporal Properties and Operators

Using the same provider (GML3EXT) as in the previous example, we can manage temporal properties and operators.

The example feature type contains several properties, intended to illustrate both the `gml:TimeInstant` and the `gml:TimePeriod` types defined in GML3. The `LAUNCHDATE` property is a `TimeInstant`, and `ACTIVITY` is a `TimePeriod`. They have been created and populated during the initialization phase. The `LAUNCHDATE` property does not need any particular configuration except defining the proper property type in the schema, i.e. `gml:TimeInstantPropertyType`.

For the `ACTIVITY` property, you also need to define it in the schema, as `gml:TimePeriodPropertyType`. You also need to map the property with a couple of columns in the database, one for the begin time position and one for the end time position, both being a `DATE` column type. This is done in the mapping file by configuring it like below:

```

<Element name="ACTIVITY" nameSQL="BEGINACTIVITY"
nameSQLCol2="ENDACTIVITY" />

```

To help you query your WFS, ERDAS APOLLO implemented the operators as defined in the OGC Change Request document "Filter Encoding Change Request CR 05-093 dated 12/10/2005": Before, After, Begins, Ends, During, TEquals, TContains, TOverlaps, Meets, OverlappedBy, MetBy, BegunBy, EndedBy.



The CR 05-093 change request is a proposal, not a voted specification. The set of operator names is subject to change without prior notice. ERDAS supports those operators as part of ERDAS APOLLO 2009, but is likely to align to changes and deprecate those names.

The request below searches for the satellites launched after end 2006.

```

<?xml version="1.0" encoding="UTF-8" ?>
<ogcwfs:GetFeature maxFeatures="20"
  xmlns:ogc="http://www.opengis.net/ogc"
  xmlns:ogcwfs="http://www.opengis.net/wfs"
  xmlns:iwfs="http://www.ionicsoft.com/wfs"
  version="1.1.0"
  service="WFS" >
  <ogcwfs:Query typeName="iwfs:satellite">
    <ogc:Filter>
      <ogc:After>
        <ogc:PropertyName>iwfs:LAUNCHDATE</ogc:PropertyName>
        <gml:TimeInstant>
          <gml:timePosition>2006-12-
31T00:00:00Z</gml:timePosition>
        </gml:TimeInstant>
      </ogc:After>
    </ogc:Filter>
  </ogcwfs:Query>
</ogcwfs:GetFeature>

```

The following example searches for satellites which activity will start on 8th November 2006 and end before 9th November 2007.

```

<?xml version="1.0" encoding="UTF-8" ?>
<ogcwfs:GetFeature maxFeatures="20"
  xmlns:ogc="http://www.opengis.net/ogc"
  xmlns:ogcwfs="http://www.opengis.net/wfs"
  xmlns:iwfs="http://www.ionicsoft.com/wfs"
  version="1.1.0"
  service="WFS" >
  <ogcwfs:Query typeName="iwfs:satellite">
    <ogc:Filter>
      <ogc:Begins>
        <ogc:PropertyName>iwfs:ACTIVITY</ogc:PropertyName>
        <gml:TimePeriod>
          <gml:beginPosition>2006-11-08T08:26:12Z</gml:beginPosition>
          <gml:endPosition>2007-11-09</gml:endPosition>
        </gml:TimePeriod>
      </ogc:Begins>
    </ogc:Filter>
  </ogcwfs:Query>
</ogcwfs:GetFeature>

```

Supported Constructs

TimeInstant currently only supports the ISO 8601 frame, plus the indeterminate positions "now" and "unknown". The calendar era name is not used (the default Gregorian calendar is used).

The TimePeriod type contains five subproperties: begin, beginPosition, end, endPosition, timeLength. The following combinations of subproperties are valid:

- begin, end, timeLength - timeLength is ignored
- begin, endPosition, timeLength - timeLength is ignored
- begin, timeLength -
- beginPosition, end, timeLength - timeLength is ignored
- beginPosition, endPosition, timeLength - timeLength is ignored
- beginPosition, timeLength -

Portrayal Capabilities

Data Portrayal

Portrayal is the use of rules to display and convert data such as GML, coverages or custom data sources, JDBC result sets, COM objects, into an image or formatted text document. For Web Application developers using OGC interfaces, data is accessed through a WFS or WCS. The ERDAS APOLLO Portrayal Engine transforms a collection of features or coverages into the required output format. Output formats can be vector format (SVG), image formats (GIF, JPEG, PNG, WBMP and, GeoTIFF) or even textual formats (text, HTML, XML and, PDF). The ERDAS APOLLO Portrayal Engine uses server-side rules to portray information. These rules can be expressed in several languages: Property, SLD and Java.

The current release of ERDAS APOLLO includes the ERDAS APOLLO Style Editor tool. ERDAS APOLLO Style Editor provides the ability to style data using ERDAS predefined rules. ERDAS APOLLO Style Editor can be used to portray data using these predefined styles and rules or using custom designed ones. Custom rules can be written using the Portrayal API.

Rules and Styles

Portrayal rules and styles are two distinct concepts. Each entity provides a different level of service but both are needed to portray data. Rules are pieces of program code that provide a specific way to portray data using a classification scheme, i.e., classes representing a numeric attribute. Styles are text files that contain parameters defining how to portray a dataset. For example, a Style will define which field the ERDAS APOLLO Portrayal Engine will classify and what fill color and stroke width to use.

Rules, the Portraying Logic

Rules define the behavior to be used when portraying any kind of compatible feature or coverage collection. They are written once and used as many times as requested on as many different data sets as required.

A rule resides in a Java class written using the ERDAS API and is dedicated to render a defined kind of data, feature and coverage collections. A rule may also use a property or SLD file which is called a *style*. The Java code may be either generic to allow the rule to be used with as many feature or coverage types as possible or written to portray a specific feature or coverage collection in a more efficient way. The ERDAS APOLLO product provides a set of generic rules to be used with any kind of features. More specific rules may be written using the developer version of this product, ERDAS APOLLO Solution Toolkit™. Developing new portrayal rules requires advanced Java knowledge.

The provided rules have been extensively tested and optimized to provide open and powerful portraying of any feature collection. Rules also support advanced logic such as generalization, classification, and new feature creation to render an updated feature collection.

Styles, Definition of the Look and Feel

Styles are collections of parameters that are used by a rule to render a specific set of data in a predefined way. They are different for each set of data and are only used by the specified data set.

A style defines set parameters to portray data using a selected rule and the properties for use in labeling and classifying and, to select a geometry to render, colors for fill, what stroke to use and what band to display. Styles are tied to the data being styled as well as to the rule it uses. Styles do not include any kind of logic and do not handle performance issues. These issues are addressed during the rules development process. The styling process only focuses on portraying the data.

To simplify the process of building styles, ERDAS APOLLO provides a tool called ERDAS APOLLO Style Editor which can define, preview and deploy styles. See [Output Formats](#) and [The ERDAS APOLLO Style Editor](#) for more information.

Creating Maps

ERDAS APOLLO Style Editor is a Java-based GUI tool that can be used to create, edit, preview and deploy styles. It contains a range of styling functionality that allows the user to style data quickly and easily. The ERDAS APOLLO Style Editor is tightly bound to the prebuilt style templates.

Styles Templates Description

To facilitate the styling of data, ERDAS APOLLO™ includes a set of prebuilt rules referred to as Style Templates. The style templates provided with the ERDAS APOLLO Style Editor may fulfill most rendering requirements, including classification and generalization. However, more templates can be built using ERDAS APOLLO Solution Toolkit™. Please contact [ERDAS Support](#) if more information about building new templates is needed.

The style templates available in ERDAS APOLLO™ include the following:

- **Uniform:** This template applies a simple style to every feature. The stroke, fill and symbol can be configured for the entire feature collection and any property of the feature can be used for labeling.
- **Known Symbol:** This template applies a fast-to-render marker from a fixed, predefined set to the centroid of each feature. A property of the feature can be used for labeling.

- **Uniform Roads:** This is a style template dedicated to the display and portrayal of various types of roads. The ERDAS APOLLO Style Editor allows custom configuration of the outline, fill color, and center line to line or polyline geometry. Road can be labeled with any property of the feature.
- **Range Classification:** This style is used to classify numeric data, ranked data (to show a progression of values), or to represent percentages.
- **Discrete Classification:** This style template should be used to symbolize categorical data. Data values where the symbol for one value is no more or less prominent than the symbols for another value. It also handles lines and polygons stroke and fill color variations as well as line/outline width.
- **Range Road Classification:** A style is attached to a range of values of a property for the classification of nondiscrete data. Values must be numeric.
- **Discrete Road Classification:** This style template is used to render roads with a discrete classification that affects outline and centerline colors.
- **HTML Report Fragment:** This style template allows the rendering of a feature collection into an HTML fragment. A subtitle can be added that will appear in the output for this feature type.
- **Variable Markers:** This style template marks features with scaled and optionally rotated symbols. The size and orientation are determined using one of the properties of the feature.
- **Patternner:** This styling rule fills polygons with patterned backgrounds.
- **Feature Numberer:** This styling rule marks the features that are the nearest from the map center with sequential numbers.
- **Symbol Roller:** This styling rule renders linear geometries by stamping a list of symbols along the curve in a cyclic manner.
- **Coverage Style:** This is the only template applying to coverages. It permits choosing the channels, the colormap and the contrast operation.

Creating Styles

Two styling languages are available in this version of the ERDAS APOLLO™, Property and SLD. ERDAS APOLLO Solution Toolkit™ allows users to plug custom styling rules into the ERDAS APOLLO Portrayal Engine and/or create their own styling mechanisms.

Languages

Property

The "Property" language is a simple, key/value-based styling language that defines both the rule to load in the ERDAS APOLLO Portrayal Engine and the parameters to apply during the portrayal of the feature collection.

Styles based on the Property language are typically created and modified using ERDAS APOLLO Style Editor. Since the set of available properties depends on the rule chosen, ERDAS APOLLO Style Editor™ offers a user-friendly, self-documenting way of setting the corresponding values.

This language cannot be used to create styles that apply to coverages. That is handled by the SLD language.

SLD

The OpenGIS Styled Layer Descriptor (SLD) is a powerful XML-based styling language. While the complete SLD mechanism is intended for WMS requests, the ERDAS APOLLO Portrayal Engine contained in this product is able to style specific feature types and coverages using a subset of the SLD tags.

For more information about SLD, see the *Styled Layer Descriptor* document in the [Implementation Specifications](#) section of the OpenGIS website. This document contains a complete description of the SLD 1.0 language and the graphical results produced by the various styling constructs.

Currently Supported SLD Tags

The following list is a reference for style developers. Depending on whether the SLD document is provided in a GetMap request to a vector data server (APOLLO vector servlet), a SLD Portray provider addressing a WFS, a SLD Portray provider addressing a WCS (also named a Coverage Portray service), or a Coverage Server, this list of supported SLD tags varies. Moreover, the following list marks the tags that are new to ERDAS APOLLO.

Notation:

- "10.0": label used for tags new in ERDAS APOLLO 10.0
- "VP": for tags supported in the vector providers
- "PP": for tags supported in the SLD Portray provider on top of a WFS
- "CP": for tags supported by the SLD Portray provider on top of a WCS or in a Web Coverage Server
- "ignored": the element does not produce an error and thus, has no apparent behavior.

Table 3: Supported SLD Tags

Tag Name	Parent Element	Supporting version/provider
NamedLayer	StyledLayerDescriptor	VP
UserLayer	StyledLayerDescriptor	PP
Name, Title, Abstract	StyledLayerDescriptor	ignored
@version	StyledLayerDescriptor	defaults to "1.0.0"
Name	NamedLayer	VP
LayerFeatureConstraints	NamedLayer	ignored
NamedStyle	NamedLayer	VP
UserStyle	NamedLayer	VP
Name	NamedStyle	VP,PP
Name	UserLayer	PP,CP
RemoteOWS	UserLayer	PP,CP
LayerFeatureConstraints	UserLayer	PP
LayerCoverageConstraints	UserLayer	CP
UserStyle	UserLayer	PP,CP
Service	RemoteOWS	"WFS" or "WCS"
OnlineResource	RemoteOWS	PP - "&" must be written "&" (i.e. xml encoding)
FeatureTypeConstraint	LayerFeatureConstraints	PP
CoverageConstraint	LayerCoverageConstraints	CP
FeatureTypeName	FeatureTypeConstraint	PP
Filter	FeatureTypeConstraint	PP

Table 3: Supported SLD Tags (Continued)

Tag Name	Parent Element	Supporting version/provider
Extent	FeatureTypeConstraint	ignored
CoverageName	CoverageConstraint	CP
Extent	CoverageConstraint	CP (to be deprecated. Use CoverageExtent instead)
CoverageExtent	CoverageConstraint	CP
TimePeriod	Extent/CoverageExtent	CP
RangeAxis	Extent/CoverageExtent	CP
Name	UserStyle	ignored: mapped with the STYLES parameter of the GetMap request
Title, Abstract, IsDefault	UserStyle	ignored
FeatureTypeStyle	UserStyle	Rendered sequentially (last on top)
CoverageStyle	UserStyle	CP
Name, Title, Abstract	FeatureTypeStyle	ignored
FeatureTypeName	FeatureTypeStyle	Useless if the UserLayer has 1! feature type
SemanticTypeIdentifier	FeatureTypeStyle	ignored
Rule	FeatureTypeStyle	Rendered sequentially (last on top)
Rule	CoverageStyle	CP
Name	Rule	VP
Title, Abstract	Rule	ignored
LegendGraphic	Rule	ignored
Filter	Rule	1.0: parameters are supported and can be expressions
ElseFilter	Rule	VP,PP
MinScaleDenominator, MaxScaleDenominator	Rule	1.0
RasterSymbolizer	Rule	CP
*Symbolizer (except Raster)	Rule	VP,PP
Geometry	*Symbolizer	VP,PP - can be omitted if only one in the feature type.
Stroke	LineSymbolizer	VP,PP
PropertyName	Geometry	VP,PP

Table 3: Supported SLD Tags (Continued)

Tag Name	Parent Element	Supporting version/provider
CssParameter	Stroke	VP,PP
ParameterValueType	CssParameter	VP,PP
expression	ParameterValueType	VP,PP
Graphic	GraphicFill	1.0
Fill	PolygonSymbolizer	VP,PP
Stroke	PolygonSymbolizer	VP,PP
GraphicFill	Fill	1.0
CssParameter	Fill	VP,PP
Graphic	PointSymbolizer	VP,PP
ExternalGraphic	Graphic	1.0
Mark	Graphic	VP,PP
Opacity	Graphic	CP
Size	Graphic	1.0
Rotation	Graphic	1.0
OnlineResource	ExternalGraphic	1.0
Format	ExternalGraphic	1.0
WellKnownName	Mark	VP,PP
Fill	Mark	VP,PP
Stroke	Mark	VP,PP
Label	TextSymbolizer	VP,PP
Font	TextSymbolizer	VP,PP
LabelPlacement	TextSymbolizer	VP,PP
Halo	TextSymbolizer	ignored
Fill	TextSymbolizer	VP,PP
CssParameter	Font	VP,PP
PointPlacement	LabelPlacement	at the centroid of the geometry
LinePlacement	LabelPlacement	ignored
AnchorPoint	PointPlacement	ignored
Displacement	PointPlacement	VP,PP
Rotation	PointPlacement	VP,PP

Table 3: Supported SLD Tags (Continued)

Tag Name	Parent Element	Supporting version/provider
DisplacementX, DisplacementY	Displacement	VP,PP
ChannelSelection	RasterSymbolizer	CP
ContrastEnhancement	RasterSymbolizer	CP
ColorMap	RasterSymbolizer	CP
ShadedRelief	RasterSymbolizer	CP
ReliefFactor	ShadedRelief	CP
RedChannel, Green, Blue, Gray	ChannelSelection	CP
SourceChannelName	RedChannel, Green, Blue, Gray	CP
Normalize	ContrastEnhancement	CP
Histogram	ContrastEnhancement	CP
ColorMapEntry	ColorMap	CP
@color	ColorMapEntry	CP
@quantity	ColorMapEntry	CP

Notes:

- The NamedStyle is processed as a style name.
- The Filter in a Rule element is ignored if it applies to a feature property that does not exist. It is important to ensure the proper handling of syntax errors in property names.
- Globally, the Title and Abstract tags have no effect, as they only make sense when SLD content is published by a server.
- Default SLD version is "1.0.0".
- The LAYERS parameter must be mentioned in the GetMap request to an ERDAS WMS even when the SLD tag is used. Among the layers found in the SLD, only those found in the LAYERS parameter are rendered. This restriction does not apply to the Portray Provider or to the CPS.
- Unknown CssParameters do not produce an error message. A CssParameter can be specified multiple times overriding the values previously set.

- In Text Symbolizers, text anti-aliasing is enabled by default.

Parsing:

The SLD parsing is lenient. If the file is not a valid XML document, the parser will detect an error. If it does not conform to the SLD DTD, the parser will try to extract the most valuable information and continue processing.

Example:Sample SLD Style

```
<StyledLayerDescriptor version="1.0.0"
  xmlns:ogc="http://www.opengis.net/ogc" >
  <NamedLayer>
    <Name>ESA_FIRE</Name>
    <UserStyle>
      <Name>MyStyle</Name>
      <FeatureTypeStyle>
        <FeatureTypeName>ESA_FIRE</FeatureTypeName>
        <Rule>
          <PointSymbolizer>
            <Geometry>
              <ogc:PropertyName>Geometry</ogc:PropertyName>
            </Geometry>
            <Graphic>
              <Mark>
                <WellKnownName>square</WellKnownName>
                <Fill>
                  <CssParameter name="fill">#ffff00</CssParameter>
                  <CssParameter name="fill-opacity">1.0</CssParameter>
                </Fill>
                <Stroke>
                  <CssParameter name="stroke">#0000ff</CssParameter>
                  <CssParameter name="stroke-width">2.0px</CssParameter>
                </Stroke>
              </Mark>
              <Size>20.0</Size>
            </Graphic>
          </PointSymbolizer>
          <TextSymbolizer>
            <Geometry>
              <ogc:PropertyName>Geometry</ogc:PropertyName>
            </Geometry>
            <Label>
              <ogc:Add>
                <ogc:PropertyName>LONG</ogc:PropertyName>
                <ogc:Literal>10</ogc:Literal>
              </ogc:Add>
            </Label>
            <Font>
              <CssParameter name="font-family">Arial</CssParameter>
              <CssParameter name="font-size">12.0</CssParameter>
            </Font>
            <LabelPlacement>
              <PointPlacement>
                <Displacement>
```

```

        <DisplacementX>0</DisplacementX>
        <DisplacementY>0</DisplacementY>
    </Displacement>
    </PointPlacement>
    </LabelPlacement>
    </TextSymbolizer>
</Rule>
</FeatureTypeStyle>
</UserStyle>
</NamedLayer>
</StyledLayerDescriptor>

```

Deploying Styles

The ERDAS APOLLO Portrayal Engine locates the styles to use for portraying features or coverages in a directory hierarchy whose root is defined in the `providers.fac` file.

Example of Portrayal Root Directory Setting

```

<CONFIGURATION>
...
  <STYLE DIR="C:/Erdas/ApolloServer/config/erdas-
  apollo/rendering/" />
</CONFIGURATION>

```

For ease of manipulation, the contents of that directory hierarchy can be packaged in a Grid Archive (GAR).

The styles generated by the ERDAS APOLLO Style Editor tool are packaged in archives that should then be copied in the Style directory. The default installation directory setting is `<APOLLO_HOME>/config/erdas-apollo/rendering` for vector and coverage styling.

If an appropriate style was not found in that directory hierarchy, the ERDAS APOLLO Portrayal Engine will successively look for one in a global library directory and then in each of the locations specified in the servlet-engine `CLASSPATH` variable. This cascading behavior permits the sharing of style libraries between multiple applications or providers.

Provider-Specific Styles

If WFS/WMS provider-specific styles are defined, an additional level of directories with names corresponding to the provider can be created under the root of the hierarchy, for example, <APOLLO_HOME>/config/erdas-apollo/rendering/atlanta_vector, where ATLANTA_VECTOR is the provider name. If such a directory exists, the corresponding provider will search there first for styles corresponding to the retrieved features. This is the easiest way to ensure that no feature type name conflict will arise between multiple providers.

Note that in the current ERDAS APOLLO release, provider-specific styles cannot be created for WCS/WMS.

Deployment Structure

Styles

In each style directory, styles are searched in a path structure composed by successively appending the lowercased `collection` keyword, the feature type or coverage offering name and then the style name. The style filenames are composed of a target format, `SVG` for SVG and raster formats and `HTML` for HTML, etc., and an extension corresponding to the language used `.prop` for property styles and `.sld` for SLD fragments.

For example, a WMS request for the portrayal of the `Buildings` feature type/layer with the `outline` style in PNG format will either use the `collection/buildings/outline/SVG.prop` or the `collection/buildings/outline/SVG.sld` style.



A particular style, named "default", is used if no style is mentioned in the request. The directory name for that style must be named "defaultstyle" if it applies to features but not for coverages. For example, the "default" style for rendering the "Buildings" layer in PNG is `collection/buildings/defaultstyle/SVG.prop`.

Symbols

When a style contains a reference to a symbol, the path searched is composed by successively appending the `symbol` keyword, the symbol library name and the symbol filename.

The ERDAS APOLLO Portrayal Engine libraries can contain symbols in a variety of raster formats, GIF, PNG, etc. as well as SVG and TrueType.

An example of a Property style referencing a TrueType font used as symbol is included in the distribution, and can be invoked through the ATLANTA_VECTOR WFS provider, using the place_names layer with the "truetype" style. The font set is symbol/lib/hanshand.ttf .

Using the Map Dressing Service

The Map Dressing Service is used for placing common map production elements, i.e., north arrow, scale bar, grid, is included in the product and appears as a provider in the WMS servlet. This service is preconfigured at installation and is automatically activated. The service can be invoked using the following URL:

http://localhost:8080/erdas-apollo/map/MAPDRESSING?service=WMS&version=1.1.1&request=GetCapabilities

This request provides a set of available layers, predefined styles for each layer, and additional "Dimensions" that allow for flexibility in the output. Below is a description of the styles and dimensions for each layer of the Map Dressing Service. For more detail on how to configure this service, please refer to [Provider Types](#).

Grid

The grid is a map element used for registering the positions of data to uniform intervals. A grid is based on defined subdivision levels of the page units, i.e., inches, centimeters.

Two styles are defined that determine the units in which the grid is displayed and labeled.

- `currentsrs` - Displays the grid in the SRS unit of the request
- `wgs84` - Displays the grid in WGS84 (EPSG:4326) that is the default SRS defined by OGC standards.

Additional parameters can be included in a request to customize the appearance of the grid line and the properties of the labels. By default, the interval of the grid is automatically calculated using the units of the SRS, to best fit the display window. However, the parameter "gridstep" can be added to a WMS request to customize this measure.

- `gridstep` - Step of the grid, in the request SRS unit

By default, the grid is composed of a set of horizontal and vertical lines crossing the entire map. This can be modified to display small crosses or "crosshairs" at the intersection of those lines. This is done with the optional parameter "gridcrosses" that is set to "false" by default.

- `gridcrosses` - Display small crosses instead of lines

The grid lines have a predefined stroke (black) and width (0.5 pixels). Using one or both of the "gridlinecolor" and "gridlinewidth" parameters in a request allows for custom values for those properties.

- `gridlinecolor` - Color of the grid using RGB values or a color name i.e., blue
- `gridlinewidth` - Width of the grid in pixels

The grid labels are texts displayed on the grid lines. The properties of those labels can be modified using additional parameters: label font name, font style, font size, color, and a halo around the text.

- `labelfontface` - Font name to use. The default is "Helvetica"
- `labelfontstyle` - Style to use to display the label, normal, italic, bold and bolditalic
- `labelfontsize` - The size of the text
- `labelfontcolor` - The stroke of the text
- `labelhalo` - Set to "true" for halo, "False" is the default



An additional property "fullGrid", can be used to display the labels at each intersection of the grid. This parameter cannot be used in a request. The property must be set in the portrayal style using "fullGrid = true" or "fullGrid = false".

An example of a GetMap request with Grid parameters:

```
http://localhost:8080/erdas-apollo/map/MAPDRESSING?
VERSION=1.1.1
REQUEST=GetMap
WIDTH=400
HEIGHT=400
SRS=EPSG:4326
BBOX=10,10,90,90
LAYERS=grid
STYLES=currentsrs
FORMAT=image/png
BGCOLOR=0xFFFFFFFF
TRANSPARENT=TRUE
EXCEPTIONS=application/vnd.ogc.se_xml
DIM_GRIDSTEP=4
DIM_GRIDLINECOLOR=blue
DIM_GRIDLINEWIDTH=5
DIM_LABELHALO=FALSE
DIM_LABELFONTFACE=Arial
DIM_LABELFONTSTYLE=bold
```

```
DIM_LABELFONTSIZE=14
DIM_LABLEFONTCOLOR=black
```

North Arrow

A North Arrow is used to show map orientation. In ERDAS APOLLO™, two north arrow styles are defined:

- `round` - Produces a simple black and grey arrow with an "N" on top
- `arrow` - Produces a blue and black wheel, with the 4 cardinal points.

The positional placement and size of the North Arrow can be specified using:

- `arrowxposition` - Horizontal offset of the arrow, starting at bottom left
- `arrowyposition` - Vertical offset of the arrow, starting at bottom left
- `arrowsize` - The size of the arrow, in pixels. By default, it has the original image size

An example of a GetMap request with a North Arrow:

```
http://localhost:8080/erdas-apollo/map/MAPADDRESSING?
VERSION=1.1.1
REQUEST=GetMap
WIDTH=400
HEIGHT=400
SRS=EPSG:4326
BBOX=10,10,90,90
LAYERS=northarrow
STYLES=arrow
FORMAT=image/png
BGCOLOR=0xFFFFFFFF
TRANSPARENT=TRUE
EXCEPTIONS=application/vnd.ogc.se_xml
DIM_ARROWXPOSITION=30
DIM_ARROWYPOSITION=40
DIM_ARROWSIZE=50
```

Scale Bar

Map scale is the relationship between the dimensions of a map and the dimensions of the Earth. It is usually expressed as a ratio between a distance on the map and a distance on the Earth, for example, 1:1000. The scale ratio 1:1000 means that one map unit represents 1000 of the same units on the Earth's surface. ERDAS APOLLO™ provides two ways to represent scales:

- `km` - Displays the scalebar in kilometers or meters
- `miles` - Displays the scalebar in miles or yards

- The "look" parameter allows two ways to render the scale bar:
- `look` - Displays the scalebar either as a colored rectangle with the scale value in it, value "simple" or as a checkerboard-like bar, value "carto". A third value is "straight" for a 0-starting value, and constant increment.

Four additional parameters provide further customization of the scalebar:

- `scalebackgroundcolor` and `scaleforegroundcolor` - Set the background and/or foreground colors of the bar. Values are given in RGB, such as (255, 255, 127).
- `scalexposition` and `scaleyposition` - Set where the bar will appear. It is expressed in pixels.
- `scalewidth` and `scaleheight` - Pixel sizing of the scalebar.
- `scalelabelfontface`, `scalelabelfontsize` and `scalelabelfontstyle` - Sets the font, size and style of the scalebar label.

An example of a GetMap request with a Scale Bar:

```
http://localhost:8080/erdas-apollo/map/MAPADDRESSING?
VERSION=1.1.1
REQUEST=GetMap
WIDTH=400
HEIGHT=400
SRS=EPSG:4326
BBOX=10,10,90,90
LAYERS=scalebar
STYLES=km
FORMAT=image/png
BGCOLOR=0xFFFFFFFF
TRANSPARENT=TRUE
EXCEPTIONS=application/vnd.ogc.se_xml
DIM_LOOK=carto
DIM_SCALEXPOSITION=200
DIM_SCALEYPOSITION=10
DIM_SCALEBACKGROUNDCOLOR=255,255,0
DIM_SCALEFOREGROUNDOLOR=255,0,0
```

Image Border

An image border is a line that surrounds the map image. ERDAS APOLLO™ provides several options for deciding how to add a border to the map.

Determine the width of the border by using the following parameters:

- `thin` - thin image border that produces a one pixel dark blue border.

- `thick` - thick image border that produces a two pixel dark blue border.

Change the color and width of the border with the following parameters:

- `bordercolor` - change the stroke of the border line. Values are in RGB.
- `borderwidth` - user-defined thickness expressed in pixels.

An example of a GetMap request with an Image Border:

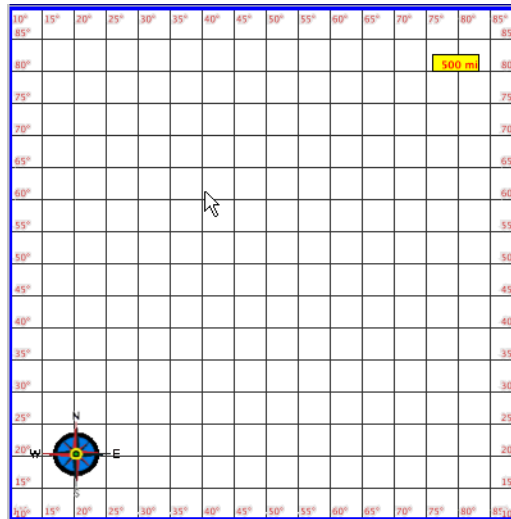
```
http://localhost:8080/erdas-apollo/map/MAPDRESSING?
VERSION=1.1.1
REQUEST=GetMap
WIDTH=400
HEIGHT=400
SRS=EPSG:4326
BBOX=10,10,90,90
LAYERS=border
STYLES=thick
FORMAT=image/png
BGCOLOR=0xFFFFFFFF
TRANSPARENT=FALSE
EXCEPTIONS=application/vnd.ogc.se_xml
DIM_BORDERCOLOR=255,0,0
DIM_BORDERWIDTH=4
```

Complete Dressing Example

An example of a GetMap request with the available Map Dressing parameters:

```
http://localhost:8080/erdas-apollo/map/MAPDRESSING?
VERSION=1.1.1
REQUEST=GetMap
WIDTH=400
HEIGHT=400
SRS=EPSG:4326
BBOX=10,10,90,90
LAYERS=grid,northarrow,scalebar,border
STYLES=wgs84,round,miles,thin
FORMAT=image/png
BGCOLOR=0xFFFFFFFF
TRANSPARENT=FALSE
EXCEPTIONS=application/vnd.ogc.se_xml
DIM_GRIDSTEP=5
DIM_ARROWXPOSITION=50
DIM_ARROWYPOSITION=50
DIM_SCALEXPOSITION=350
DIM_SCALEYPOSITION=350
DIM_SCALEBACKGROUNDCOLOR=255,255,0
DIM_SCALEFOREGROUNDOLOR=255,0,0
DIM_BORDERCOLOR=0,0,255
DIM_BORDERWIDTH=3
```

Figure 8: Map Dressing Output



Displaying Statistics in a Map

Displaying statistics in a map provides the ability to determine exactly what types of geometry and how much geometry is portrayed in the style/rule issued to the ERDAS APOLLO Portrayal Engine. Display statistics can be used to analyze the service performance. Issue this request to the portrayal engine to return statistical information on the amount and complexity of the geometry requested.

Call

Use the WMS GetMap request and append the "NEEDSTAT=TRUE" option.

Output Information

Statistics are only obtained when requesting a raster format. SVG and GML2 output do not contain portrayal statistics. The response is left-aligned text in the image returned:

Example:Response to NEEDSTAT=TRUE

A sample result:

```
Total Geometry:n
Total Group:n
Total Shape:n
Geometry Types:
  Type Unknown(0) : n
  Type Point(1) : n
  Type Line(2) : n
  Type Ring(3) : n
  Type Polygon(4) : n
  Type MultiPoint(5) : n
  Type MultiLine(6) : n
```

```

Type MultiPolygon(7) : n
Total Point:n
Max Point:n
Mean Point:n
Total Color:n
World Size:f x f
Pixel Size:n x n
StandardScale:f

```

Definitions

Geometry is defined in the feature type and includes point, line, polygon, ring, multipoint, multiline, or multipolygon. See the OGC feature types specification.

A shape is a geometric property defined by the Java AWT interface. The shape is described by a PathIterator object, that can express the outline of the shape as well as a rule for determining how the outline divides the two-dimensional plane into interior and exterior points. Each shape object provides callbacks to get the bounding box of the geometry, determine whether points or rectangles lie partly or entirely within the interior of the shape, and retrieve a PathIterator object that describes the trajectory path of the shape outline.

The ERDAS APOLLO Portrayal Engine uses the SVG concept of group to apply the same set of common property values to a set of geometries.

Portrayal Statistics Output Values

Table 4: NeedStat Output Meaning

Line	Description
Total Geometry	The number of graphic geometries that are portrayed
Total Group	Number of SVG groups returned.
Total Shape	The total number of Java AWT shapes that are rendered
Geometry Types	Total amounts are shown for different geometry types, numbered 0 to 7
Total Point:	The total number of points - including points in geometry types other than point (line, polygons,etc)
Max Point	The maximum number of points in a geometry
Mean Point	The mean number of points for the geometries returned
Total Color	The total number of colors requested in the map (based on an RGBA scale)

Table 4: NeedStat Output Meaning (Continued)

Line	Description
World Size	The width and height of the map returned. Units are expressed in a specified unit of measure based on the coordinate transform system.
Pixel Size	Size in pixels of the output image (width and height).
StandardScale	StandardScale is an SLD property, giving the denominator of the map scale.**

**Refer to the SLD 1.0 specification for additional information on standardscale.

Producing KML

KML is a file format used to display geographic data in an Earth browser, such as Google Earth or Google Maps.

Outputting KML with ERDAS APOLLO is possible in various ways, for different types of KML contents:

Without changing anything in your portrayal configuration, you can request maps (with the GetMap request) in KML out of a vector service (WFS), by providing the proper FORMAT parameter value: `FORMAT=application/vnd.google-earth.kml+xml` (replace the "+" sign with "%2B" if in a URL). You also need to be sure that the SRS used in EPSG:4326. Sample request:

```
http://localhost:8080/erdas-  
apollo/vector/ATLANTA_VECTOR?VERSION=1.1.1&REQUEST=GetMap&SRS=EPSG%3A4326&  
BBOX=-71.07503,42.264893,-  
71.06302,42.273808&WIDTH=500&HEIGHT=500&  
LAYERS=protectedareas,hydro,roads,highways,place_names&STYLES=,  
,, &  
FORMAT=application/vnd.google-earth.kml%2Bxml&  
BGCOLOR=0xffffffff&TRANSPARENT=FALSE&EXCEPTIONS=application/vnd.o  
gc.se_xml
```

The ERDAS APOLLO Portrayal Engine, after building the graphic objects that need to be rendered, produces a KML (XML) output instead of a SVG-like structure. But as this stage of the process has no more information about what entities (features) are to be rendered, the produced KML is semantically poor and only holds graphic information. This is a basic KML output that can be produced with minimal configuration efforts and that can fulfill a set of use cases.

Some limitations:

- only the basic styling information is found in the KML document: predefined point icon, line stroke, polygon stroke and fill, polygon opacity.
- Some of the portrayal rules do not produce any output in KML: Predefined Symbols for lines and polygons.
- Some rules produce reduced output: Entity Numbering rules for lines and polygons only produce placemarks
- The GetFeatureInfo request type, with INFO_FORMAT=application/vnd.google-earth.kml+xml, will also produce a KML document that can be opened into your Earth Browser. It will only contain the objects that we found by the request.
- For smarter maps production in KML, you need to write a specific Java rule that receives, as input, the feature set to render and the set of portrayal parameters. You can use the ERDAS KML Helper API to easily generate the KML elements that fit your needs.
- In a near future, you will no more need to write such a rule if you requirements are not too specific. ERDAS is writing a set of such rules to help you start up.
- For KML output out of the raster and coverage servlets, you will not get more than images. They can be either referenced as a GetMap URL in the KML document, or embedded as a raster image in a KMZ archive. For this last case, the format to use in the request is application/vnd.google-earth.kmz. When the output is KML from GetMap, ERDAS APOLLO 2011 leverages the Google Super-overlays technology so that further GetMap requests are executed with small tiles.
- In the future, KML will also be written as a direct conversion of vector data, i.e. as output of a GetFeature request.

The ERDAS APOLLO Catalog is also able to output KML. It outputs the various selected catalog objects, with their title, abstract, description, object sub-tree if any, and footprint if any. If the object is a KML-enabled vector layer, it will contain a network link to a GetMap request to that service. If the object is an KML-enabled image layer, the output is a network link to a GetMap request producing Google Super-overlays (a technology allowing tiling).

Limitations

Fast 2D Rendering

The normal behavior of the portrayal engine is to build an SVG-like tree of graphical elements before converting them into an image or a real SVG file.

But ERDAS APOLLO also includes an alternative portrayal method, which directly uses the Sun Java2D library. This method uses low-level functions to draw points, lines, polygons and several other shapes. It is several times faster than the normal portrayal method but it does not apply in all cases. Some of the situations where it does not provide a proper result are:

- If coordinate transform is needed;
- When using area patterns, line dashes and SVG symbols;
- When requesting SVG as output format.

Coverage Portrayal

In the current Release of ERDAS APOLLO, the styling capabilities are not complete. The restrictions, which are likely to disappear in future releases are:

- No provider-specific style is interpreted by the portray engine.
- The ERDAS APOLLO Style Editor does not produce the GAR to allow deployment on the server. The hierarchy must be built manually.
- The SLD language is the only one currently supported. Property styles are not read.

Output Formats

Overview

ERDAS APOLLO™ supports the output of data into various file formats for further use or for data sharing purposes. The product allows output in a variety of formats including images, text and HTML. In order to output data, the output format must be included in the HTTP request for a map, feature or coverage.

Image Outputs

The ERDAS APOLLO product supports a variety of different image outputs. All image outputs can be initiated from a GetMap request on either a raster or vector WMS.

Graphic Interlaced Format (GIF)

GIF is the most common format used on the Internet and is best for simple graphics, i.e., line art and simple images with large blocks with a few colors. GIF files are good for representing graphics, as opposed to JPEG or other image format types, because the file size is small and of a better quality. A GIF file can handle only 256 colors which makes it inappropriate for photo images. GIFs work well for images like company logos or screen shots. These images should be reduced to 16 colors, if possible, and saved as a GIF.

Copy and paste the example provided below in the Service Tester for a GetMap request in GIF format.

```
http://localhost/erdas-apollo/coverage/ATLANTA_IMAGE?  
VERSION=1.1.1  
REQUEST=GetMap  
WIDTH=400  
HEIGHT=400  
SRS=EPSG:4269  
BBOX=-84.5318499,33.59473822,-84.22546959,33.9180625  
LAYERS=40379914  
STYLES=default  
FORMAT=image/gif  
BGCOLOR=0xFFFFFFFF  
TRANSPARENT=FALSE  
EXCEPTIONS=application/vnd.ogc.se_xml
```

Joint Photographic Experts Group (JPEG)

The Joint Photographic Experts Group (JPEG) is an organization that sets standards for graphic file formats. JPEG is a compressed format, with some loss of quality due to compression. JPEGs are best for photos because the file size is small and there is no limit to the number of colors used. Other file extensions used are .jpg, .jpeg, and .jpe.

ERDAS APOLLO supports a QUALITY parameter in the GetMap request that sets the compression ratio between 0 and 100, 0 being the maximum compression (lowest quality).

Copy and paste the example provided below in the Service Tester for a GetMap request in JPEG format.

```
http://localhost/erdas-apollo/coverage/ATLANTA_IMAGE?  
VERSION=1.1.1  
REQUEST=GetMap  
WIDTH=400  
HEIGHT=400  
SRS=EPSG:4269  
BBOX=-84.5318499,33.59473822,-84.22546959,33.9180625  
LAYERS=40379914  
STYLES=default  
FORMAT=image/jpeg  
BGCOLOR=0xFFFFFFFF  
TRANSPARENT=FALSE  
EXCEPTIONS=application/vnd.ogc.se_xml  
QUALITY=70
```

Keyhole Markup Language (KML)

KML is a file format used to display geographic data in an Earth browser, such as Google Earth, Google Maps, and Google Maps for mobile. KML uses a tag-based structure with nested elements and attributes and is based on the XML standard.

KMZ files are KML files, sometimes along with raster images, the whole being compressed using ZIP compression technology.

ERDAS APOLLO™ provides several ways to produce KML and KMZ documents, depending on the nature of the data you want to output.

- For raster data (the Coverage servlet), you can request KML to obtain a light document containing the URL to a GetMap request with a raster output format. You can also request KMZ, in which case the zip contains a light KML document and an image which is the output of a GetMap in PNG.
- For vector data sets (the WFS servlet), you can only request KML, KMZ being dedicated to raster data sets. As soon as the portrayal styles have been created in order to enable the WMS interfaces out of your servlet, a GetMap request will convert the data into graphics, like for the raster formats, and then convert it to KML. This means that few data attributes will be available, due to the late stage of this conversion.

To benefit from the whole power of the KML output, a specific Java rule can be written, compiled and uploaded on the server. Such a rule can be easily written thanks to a light Java API and a set of helper classes - see ERDAS APOLLO Solution Toolkit for more detail on this API.

With ERDAS APOLLO™, the KML and KMZ formats are published in the Capabilities document for the WMS interface, either as their format names "KML" or "KMZ" or as their mime-types "application/vnd.google-earth.kml+xml" or "application/vnd.google-earth.kmz", depending on the version of the WMS specification.

Scalable Vector Graphics (SVG)

SVG is an XML grammar used for modeling graphics. It differs from the GIF and JPEG in that it uses graphic objects rather than individual points. SVG is also a scalable format. This means that a graphic can be rendered at differing resolutions.

ERDAS APOLLO™ supports three implementations of SVG.

- The default method of implementing SVG output is to issue a format=image/svg+xml (or format = SVG in older versions of the WMS spec) and the application will return an XML document that can be read by any W3C compliant viewer.



This option may produce outputs not readable in an Adobe SVG reader. The Adobe SVG Reader 3.0 is not completely compliant with the W3C specification and does not support base64 encoded content using the "data" protocol for SVG images. This basically means that Adobe will not support a data URL for embedded SVG image files, but will support embedded raster symbols.

If the output contains SVG embedded symbols or pattern fills, it cannot be viewed in the 3.0 version of Adobe. Convert the embedded symbols into pure raster format, GIF or PNG, since Adobe will support embedded symbols in this format or use other ERDAS implementations that support non-compliant SVG viewers.

- The work-around to non-compliant SVG viewers is SVG output without embedded symbols. Simply edit the feature mapping file and add the "DontEmbedSVG" option. The procedure is outlined below:
 1. Locate the feature mapping file. The default location is the providers.fac file directory config/erdas-apollo/providers/vector. Open it in a text editor.
 2. Locate the "Option" section normally found before the beginning of the <Mapping> elements.
 3. Add a new Option listing as follows:

```
<Option>  
  <DontEmbedSVG>true</DontEmbedSVG>  
</Option>
```

4. Save the file and if necessary re-start the servlet.

This option will now use HTTP references for linked or embedded symbols instead of the data URL. This means that all embedded symbols will now appear as HTTP references that the client must download to bring into the desired output.



Ensure that all embedded symbol files are downloaded and stored in the same directory or subdirectory as the main SVG document.



The Java rule developer must insure that the rule created relies on the Web servlet container that has this option in the providers.fac file.

- ERDAS has also created a new mime type format for SVG that returns a zipped document that contains the main SVG document and all embedded or linked files. This option supports local SVG output. To use this option, use the parameter `format=image/svg+zip` or `format = SVG_ZIP` in older versions of WMS and the application will return a zipped file. Unzip the file and place the main SVG document and it's corresponding files into the same directory. Any W3C compliant SVG viewer can read the SVG file.



If the document contains embedded SVG symbols, the output will not work in Adobe 3.0. Either convert the embedded symbols to pure raster format or ensure that the SVG document contains relative HTTP reference links. Links to embedded symbols using a data URL will not work in Adobe.

Due to Adobe SVG Viewer limitations, text rendered with a Halo will not display a complete image in Adobe SVG Viewer. Also, any style or rule producing one of the SVG codes Adobe mentions as non-supported will produce an unreadable file. See Adobe limitations at <http://www.adobe.com/svg/indepth/releasenotes.html>.

GeoTIFF

"GeoTIFF" refers to TIFF files which have geographic or cartographic data embedded as tags within the TIFF file. The geographic data, mainly the SRS and the extent in the header file, can be used to position the image in the correct location and geometry on the screen of a geographic information display.

ERDAS offers full support of the GeoTIFF image format. Several data providers are committed to delivering imagery in GeoTIFF format including SPOT Image Corp, Trifid (representing LandSat data), Space Imaging, US Geological Survey, and the New York Department of Transportation. In addition, the United Kingdom Military Survey has announced it is testing the format for their products.

The following request shows how to return a GeoTIFF image from an ERDAS WMS:

```
http://localhost:8080/erdas-apollo/map/ATLANTA_IMGIDX?
VERSION=1.1.1
REQUEST=GetMap
WIDTH=400
HEIGHT=400
SRS=EPSG:26986
BBOX=225000,886000,237000,902000
LAYERS=ATLANTA_IMGIDX
STYLES=default
FORMAT=image/tiff
BGCOLOR=0xFFFFFFFF
TRANSPARENT=FALSE
EXCEPTIONS=application/vnd.ogc.se_xml
```

Portable Network Graphic (PNG)

PNG is a file format for image compression that, in time, is expected to replace the Graphics Interchange Format (GIF) that is widely used on the Internet. The PNG format was expressly developed to be patent-free. A PNG file is compressed in "lossless" fashion meaning all image information is restored when the file is decompressed during viewing. PNG includes the following upgrades from the GIF format:

- Degree of opacity (transparency)
- Interlacing
- Gamma correction
- True color or GIF palettes

The following request will return output in PNG format:

```
http://localhost/erdas-apollo/coverage/ATLANTA_IMAGE?
VERSION=1.1.1
REQUEST=GetMap
WIDTH=400
HEIGHT=400
SRS=EPSG:4269
BBOX=-84.5318499,33.59473822,-84.22546959,33.9180625
LAYERS=40379914
STYLES=default
FORMAT=image/png
```

```
BGCOLOR=0xFFFFFFFF
TRANSPARENT=FALSE
EXCEPTIONS=application/vnd.ogc.se_xml
QUALITY=30
```

ERDAS APOLLO™ supports PNG output in 8- or 24-bits. To do so, add the "QUALITY" parameter to a request, with a value between 0 and 50 for 8-bits PNG and between 51 and 100 for 24-bits PNG.

Configure any provider to produce PNG of a specific quality.

- For a WMS provider, raster or proxy-WMS, add a PARAMBLOCK tag named "quality" that holds a "PNG" parameter:

```
<PARAMBLOCK NAME="quality">
  <PARAM NAME="PNG" VALUE="30" />
</PARAMBLOCK>
```

- For a WFS provider, the configuration is in the mapping file under the <Option> tag. Create a <Generate8BitsPNG> tag containing "true" or "false" as in the following example:

```
<Option>
  ...
  <Generate8BitsPNG>true</Generate8BitsPNG>
  ...
</Option>
```

X-BMP

X-BMP is the default Windows BMP format. Use the PNG field on the Styling Info tab to return an X-BMP image format.

WBMP

A Wireless Bitmap (WBMP) is a graphic image format for sending Web content to handheld wireless devices. The format is defined in the Wireless Application Protocol (WAP), Wireless Application Environment (WAE) Specification. For Web content that is directed to handheld phones or personal digital assistants (PDA) that have Web access, use the Wireless Markup Language (WML) to encode the page and its text. An image converted from a GIF, TIFF, or other graphic format can be included in the form of a WBMP file. The initial WAP WAE specification supports only WBMP type 0 that is a compression image in monochrome. As the bandwidth for wireless transmission increases, richer images will be supported.

The following request returns a WBMP image format:

```
http://localhost/erdas-apollo/coverage/ATLANTA_IMAGE?
VERSION=1.1.1
REQUEST=GetMap
WIDTH=400
HEIGHT=400
SRS=EPSG:4269
BBOX=-84.5318499,33.59473822,-84.22546959,33.9180625
LAYERS=40379914
STYLES=default
FORMAT=image/vnd.wap.wbmp
BGCOLOR=0xFFFFFFFF
TRANSPARENT=FALSE
EXCEPTIONS=application/vnd.ogc.se_xml
```

Text Outputs

Plain Text Output

ERDAS APOLLO™ supports the following types of text output.

To produce plain text output from an ERDAS WMS or WFS, add the `INFO_FORMAT` parameter to the `GetFeatureInfo` request. This will return either plain text or Comma Delimited Tabs (CSV). The content depends on the connector type. The following is a `GetFeatureInfo` request that returns text output.

```
http://localhost/erdas-apollo/coverage/ATLANTA_IMAGE?
VERSION=1.1.1
REQUEST=GetFeatureInfo
WIDTH=400
HEIGHT=400
SRS=EPSG:4269
BBOX=-84.5318499,33.59473822,-84.22546959,33.9180625
LAYERS=40379914
STYLES=default
FORMAT=image/gif
BGCOLOR=0xFFFFFFFF
TRANSPARENT=FALSE
EXCEPTIONS=application/vnd.ogc.se_xml
QUERY_LAYERS=40379914
INFO_FORMAT=text/plain
X=200
Y=220
```

HTML

The `INFO_FORMAT` parameter of the `GetFeatureInfo` request will also return output in HTML format. For vector data, the output will be a simple but smart HTML page that contains a header, body and footer text, a logo and title. The output will also contain a custom style sheet (CSS) that allows flexible configuration.

HTML output is accessible from the ERDAS APOLLO Style Editor™. ERDAS APOLLO Style Editor™ creates styles for HTML output that can be used in the ERDAS APOLLO Portrayal Engine. Refer to the "ERDAS APOLLO Style Editor User Guide" for additional information on how to access this functionality.

Following is a GetFeatureInfo request that returns HTML output.

```
http://localhost:8080/erdas-apollo/vector/ATLANTA_VECTOR?
VERSION=1.1.1
REQUEST=GetFeatureInfo
WIDTH=400
HEIGHT=400
SRS=EPSG:4269
BBOX=-84.5318499,33.59473822,-84.22546959,33.9180625
LAYERS=40379914
STYLES=default
FORMAT=image/gif
BGCOLOR=0xFFFFFFFF
TRANSPARENT=FALSE
EXCEPTIONS=application/vnd.ogc.se_xml
QUERY_LAYERS=40379914
INFO_FORMAT=text/html
X=200
Y=220
```

GeoRSS

RSS (Rich Site Summary) is an XML format for delivering regularly changing web content. Many news-related sites, weblogs and other online publishers syndicate their content as an RSS Feed to whoever wants it.

GeoRSS is an emerging standard for encoding location as part of a RSS feed (see <http://www.georss.org> for in-progress work on GeoRSS).

ERDAS APOLLO™ supports GeoRSS as output of GetFeature requests on vector data sets (the WFS servlet). It produces RSS 2.0 documents, with both GeoRSS Simple and GeoRSS GML outputs for the geometries. The sample output below shows a GeoRSS output by ERDAS APOLLO.

```
<?xml version='1.0' encoding='utf-8' ?>
<rss version="2.0" xmlns:georss="http://www.georss.org/georss">
  <channel>
    <title>LocalName</title>
    <link>http://www.erdas.com</link>
    <description>no description</description>
    <item>
      <title>CAMBRIDGE</title>
      <georss:where>
```

```

        <Point xmlns="http://www.opengis.net/gml"
srsName="EPSG:26986">
        <pos>232226.47 901710.31</pos>
        </Point>
    </georss:where>
    <georss:point>42.36522907219097 -
71.10877119738284</georss:point>
    </item>
</channel>
</rss>

```

JSON

JSON, short for JavaScript Object Notation, is a lightweight computer data interchange format. It is a text-based, human-readable format for representing simple data structures and associative arrays (called objects).

The official Internet media type for JSON is application/json. The JSON filename extension is .json.

The JSON format is often used for serialization and transmitting structured data over a network connection. Its main application is in Ajax web application programming, where it serves as an alternative to the XML format.

Data Outputs

Shapefiles

Shapefile is the most commonly used format for exchanging GIS data. ERDAS APOLLO™ supports shapefile output in zip format. To obtain shapefile output, append the "outputFormat=SHAPE" parameter to a WFS GetFeature request.

Following is a GetFeature request that returns Shapefile output:

```

<?xml version="1.0" encoding="UTF-8" ?>
<ogcwfs:GetFeature maxFeatures="20"
xmlns:ogc="http://www.opengis.net/ogc"
xmlns:ogcwfs="http://www.opengis.net/wfs"
version="1.0.0"
service="WFS"
outputFormat="SHAPE" >
<ogcwfs:Query typeName="iwfs:roads">
</ogcwfs:Query>
</ogcwfs:GetFeature>
</ogcwfs:GetFeature>

```


GML 2/3

GML is an open, non-proprietary language used to create geo-spatial objects for the purpose of data sharing. GML serves as a data transport for geo-spatial objects as well as providing a means for describing geo-spatial Web services. GML is constantly evolving and has quickly become the standard geo-spatial information (GI) format for all products that are based on international GI standards.

GML2 is the default output format of the ERDAS WFS 1.0.0 GetFeature request but GML2 output can also be explicitly requested by appending the "outputFormat=GML2" parameter to a WFS GetFeature request.

GML3 format is also supported by ERDAS WFS. Request a GML3 output by appending the "outputFormat=GML3" parameter onto a WFS 1.0.0 GetFeature request. Starting with ERDAS APOLLO, the OGC WFS 1.1.0 specification is supported. If a WFS 1.1.0 GetFeature request is sent, the default output format is GML3. This is only possible if the WFS provider is configured as a GML3 one. The feature types schema must include the GML 3.1.1 feature.xsd schema and the schema must validate against the GML 3.1.1 schemas. More information on setting up a GML3-compliant WFS is given in [Moving to GML3](#).

Note that default outputFormat behavior can be overridden by defining the <GMLOutputFollowModel>true</GMLOutputFollowModel> in the <Option> section of the mapping file. Setting it to true will lead to output being driven by the feature model used to set up the WFS.

GeoTIFF

When requesting coverages from a Web Coverage Service (WCS), the coverages can be output in GeoTIFF format. This output image can have one to n bands with 8-, 16- or 32-bit integer data (signed or unsigned), or 16-, 32- or 64-bit floating point data. In other words, the output can be any band combination and any data type.

Coverage data values do not represent pixel luminescence (red, green, blue, cyan, magenta, yellow and black), but a physically measured value (32-bit elevation float data or 16-bit temperature short data).

A GeoTIFF cannot be viewed using standard image applications.

JPEG2000, ECW, NITF, DTED

In a limited set of situations, ERDAS APOLLO allows data to be output in several other formats, thanks to the GDAL library.

The conditions to be able to produce JPEG2000, ECW, NITF or DTED output are:

- The service must be an ERDAS WCS servlet. It is recognized as the path in the URL contains "coverage". Example:
http://localhost/erdas-apollo/coverage/ATLANTA_MOSAIC.

- The type of the provider must be SimpleProvider, MultiSimpleProvider, IndexProvider or HierarchicalProvider.
- The request must be a WCS GetCoverage.
- For some formats like ECW, the appropriate proprietary library has to be linked with GDAL in order to be served. However they are not available on all platforms. Please refer to [Provider Types](#) and Table "GDAL-based Source Formats by Platform" for more details on formats served via GDAL.

To make sure the service can produce a coverage in one of those formats, run a DescribeCoverage command on that tile. At the bottom of the output document, there is a XML section similar to the example below:

```
<supportedFormats>
  <formats>GeoTIFF</formats>
  <formats>DTED</formats>
  <formats>ECW</formats>
  <formats>JPEG2000</formats>
  <formats>NITF</formats>
</supportedFormats>
```

ERDAS IMG

ERDAS IMAGINE is a complete suite of geoprocessing tools for geoscience and earth resource applications for use in image processing, GIS in remote sensing, and photogrammetry.

ERDAS IMG is the high performance raster data format that is used by ERDAS IMAGINE. ERDAS APOLLO provides full support for reading and writing ERDAS IMG format.

Coordinate Transformations

This chapter gives a detailed explanation on how to use the Coordinate Transformations on data using ERDAS servlets.

The ERDAS APOLLO product allows use of a variety of different coordinate systems on the data. We provide the ability to not only display data in a selected Spatial Reference System (SRS) but also provides an advanced and sophisticated engine that allows transformation of data from one SRS to another.

The ERDAS APOLLO product supports over 1,500 different SRS transformations. These SRS coordinate transformations are based on the EPSG classification adopted by international standards organizations such as OGC and ISO. For more information about the EPSG, OGC or ISO, consult the Concepts Guide under the Standards Section.

SRS Concepts

It may be required that the SRS employed be permitted to use other data that is in different SRSes.

There are several different flavors of coordinate transform systems (CTS) specializing in the preservation of the geographic shape, area, distance or direction for a specific spatial extent on the globe. Coordinates from different datasets will often have different reference systems. In order to use data from different coordinate reference systems, known point coordinates often must be transformed into the corresponding coordinates in a different coordinate reference system. The OGC Coordinate Transform Data Definition Specification defines which coordinate systems, all based on EPSG, AUTO or BNGrid, are to be used as well as the specific method to define transformations between coordinate systems. This definition data can be transferred between client and server software that uses OGC standard interfaces such as ERDAS servlets.

Add a Custom SRS

Only datasets with a valid SRS can be crawled in the Data Manager. If a dataset has a custom SRS that is not supported in ERDAS APOLLO, you can add it and then successfully crawl the data.

Follow these steps to add a custom SRS.

1. Add to ERDAS IMAGINE projection system
 - [Modify epsg.plb](#) on page 126
2. Add to ERDAS APOLLO

- [Create usersref.xml](#) on page 128
- [Modify coordinate_system_category.xml](#) on page 130
- [Rebuild and Deploy the Webapps](#) on page 130
- [Test the Custom SRS in the Data Manager](#) on page 131
- [Test the Custom SRS in the Web Client](#) on page 131

All of the SRS-related files are located in the cots.srs.jar file located in <APOLLO_HOME>webapps\erdas-apollo\profiles\advantage\WEB-INF\lib\cots-srs-1.3.jar. Unzip the file and browse to \com\ionicssoft\sref\impl\resource\ and the following SRS files are included.

- sref.xml - contains a factory reference location - do not modify this file
- factorysref.xml - default ERDAS SRS file - contains the information for all ERDAS-supported SRSs - do not modify this file
- ionicsref.xml - contains additional SRSs requested by ERDAS customers
- usersref.xml - contains any custom SRSs - this file is not included in the installation so you must create it when you add a custom SRS (see [Create usersref.xml](#) on page 128)



Be sure to back up all files that you modify.

Projection System Information

All of the SRS information that the ERDAS IMAGINE Projection System needs is contained in the following files.

- epsg.plb
- mapprojections.dat
- spheroid.tab
- units.dat
- sptable.tab

The first file, epsg.plb, is located in this directory:

```
<APOLLO_HOME>\tools\native\raster\etc\Projections
```

The last four files are located in this directory:

```
<APOLLO_HOME>\tools\native\raster\etc\
```

mapprojections.dat

This file contains definitions of all the most commonly used map projections.

The entries in the `epsg.plb` file must be linked to one of the projection definitions in `mapprojections.dat` using the projection identifiers. You do not need to modify this file to add a custom SRS. This is for advanced procedures not outlined in this chapter.

Figure 9: The Projection Identifiers in MapProjections.Dat

```
"Albers Conical Equal Area" {
  Internal 3
  "Spheroid" <spheroid>
  "Latitude of 1st standard parallel" <2:angle ns:dd>
  "Latitude of 2nd standard parallel" <3:angle ns:dd>
  "Longitude of central meridian" <4:angle ew:dd>
  "Latitude of origin of projection" <5:angle ns:dd>
  "False easting at central meridian" <6:distance ew:meters>
  "False northing at origin" <7:distance ns:meters>
}
"Lambert Conformal Conic" {
  Internal 4
  "Spheroid" <spheroid>
  "Latitude of 1st standard parallel" <2:angle ns:dd>
  "Latitude of 2nd standard parallel" <3:angle ns:dd>
  "Longitude of central meridian" <4:angle ew:dd>
  "Latitude of origin of projection" <5:angle ns:dd>
  "False easting at central meridian" <6:distance ew:meters>
  "False northing at origin" <7:distance ns:meters>
}
"Mercator" {
  Internal 5
  "Spheroid" <spheroid>
  "Longitude of central meridian" <4:angle ew:dd>
  "Latitude of true scale" <5:angle ns:dd>
  "False easting at central meridian" <6:distance ew:meters>
  "False northing at origin" <7:distance ns:meters>
}
```

The definitions in `mapprojections.dat` also indicate which parameters you need to supply in the `epsg.plb` for an SRS associated with the projection.

In the figure below, the items highlighted in purple are the parameters you must provide in an `epsg.plb` entry for an SRS associated with the Albers Conical Equal Area projection. The items highlighted in silver indicate the type of measurement for each item.

Figure 10: The Projection Parameters in MapProjections.dat

```
"Albers Conical Equal Area" {
  Internal 3
  "Spheroid" <spheroid>
  "Latitude of 1st standard parallel" <2:angle ns:dd>
  "Latitude of 2nd standard parallel" <3:angle ns:dd>
  "Longitude of central meridian" <4:angle ew:dd>
  "Latitude of origin of projection" <5:angle ns:dd>
  "False easting at central meridian" <6:distance ew:meters>
  "False northing at origin" <7:distance ns:meters>
}
```

spheroid.tab

This file contains a mathematical definitions of the most commonly used spheroids, along with definitions of the datums that are most commonly used with the spheroid. You do not need to modify this file to add a custom SRS. This is for advanced procedures not outlined in this chapter.

```
"GRS 1980" {
  9 6378137.0 6356752.31414
  "GRS 1980" 0 0 0 0 0 0
  "ETRS 1989" 0 0 0 0 0 0
  "SWEREF99" 0 0 0 0 0 0
  "CHTRF 1995" 0 0 0 0 0 0
  "NAD83" 0 0 0 0 0 0
  "NAD83 (DMA)" 0 0 0 0 0 0
  /*****
  ** The following 4 datum names have been adopted from DatumPro V3.01; not verified from other sources
  *****/
  "NAD83 (CSRS-1)" 0 0 0 0 0 0
  "NAD83 (CSRS-2)" -0.991 1.9072 0.5129 -1.25033e-07 -4.678500e-08 -5.652900e-08 0.0e+00
  "NAD83 (HARN-1)" 0 0 0 0 0 0
  "NAD83 (HARN-2)" -0.9738 1.9453 0.5486 -1.335700e-07 -4.871999e-08 -5.506999e-08 0.0e+00
  "NAD83 (Alaska)" 0 0 0 0 0 0
  "NAD83 (Aleutian Islands)" -2 0 4 0 0 0
  "NAD83 (Canada)" 0 0 0 0 0 0
  "MAY76 (Canada) (NTv2)" SURFACE BASEDATUM="NAD83 (Canada)"
  {
    RASTER RESAMPLE = "Bilinear"
    LATITUDE = "MAY76V20.GSB"
    LONGITUDE = "MAY76V20.GSB"
    GRIDBASEDATUM = "TODATUM"
  } DESCRIPTION="Natural Resources Canada, Geodetic Survey Division"
  "NAD83 (CONUS)" 0 0 0 0 0 0
  "NAD83 (Hawaii)" 1 1 -1 0 0 0
  "NAD83 (Mexico and Central America)" 0 0 0 0 0 0
  "HARN" SURFACE
  {
    RASTER RESAMPLE = "Bilinear"
    LATITUDE = "hpgn.dat"
    LONGITUDE = "hpgn.dat"
    GRIDBASEDATUM = "FROMDATUM"
  } DESCRIPTION="From HARN grid hpgn.dat"
  "GDA94" -16.237 3.51 9.939 1.4157e-06 2.1477e-06 1.3429e-06 1.91e-07
  /* Note about GDA94: The source for these parameters is unknown. Other sources indicate these parameters should all be 0. */
  "GDA94-ICSM" 0 0 0 0 0 0
  "AGD66 (NTv2)" SURFACE BASEDATUM="GRS 1980"
  {
    RASTER RESAMPLE = "Bilinear"
    LATITUDE = "A66_National_13.09.01.gsb"
    LONGITUDE = "A66_National_13.09.01.gsb"
    GRIDBASEDATUM = "TODATUM"
  } DESCRIPTION="Australian Geodetic Datum AGD66"
  "AGD84 (NTv2)" SURFACE BASEDATUM="GRS 1980"
  {
    RASTER RESAMPLE = "Bilinear"
    LATITUDE = "National_84_02.07.01.gsb"
    LONGITUDE = "National_84_02.07.01.gsb"
    GRIDBASEDATUM = "TODATUM"
  } DESCRIPTION="Australian Geodetic Datum AGD84"
```

units.dat

This file contains all of the units of measure most commonly used for coordinate systems and projections.

The file is divided into different blocks for all of the things that need to be measured, such as angles, distance, and area.

Each block contains a list of units that ERDAS IMAGINE can recognize, along with a conversion factor that is used internally by the software. You do not need to modify this file to add a custom SRS. This is for advanced procedures not outlined in this chapter.

```
angle {
  radians 1.0 ;
  radian 1.0 ;
  rad 1.0 ;
  dd PI / 180.0 ;
  dm PI / 180.0 / 60.0;
  ds PI / 180.0 / 3600.0;
  degrees PI / 180.0 ;
  degree PI / 180.0 ;
  deg PI / 180.0 ;
  dg PI / 180.0 ;
  grad PI / 200.0 ;
  gons PI / 200.0 ;
  gon PI / 200.0 ;
}
distance {
  meters 1.0 ;
  meter 1.0 ;
  m 1.0 ;
  centimeters 0.01 ;
  centimeter 0.01 ;
  cm 0.01 ;
  millimeters 0.001 ;
  millimeter 0.001 ;
  mm 0.001 ;
  kilometers 1000.0 ;
  Kilometer 1000.0 ;
  km 1000.0 ;
  nanometers 0.000000001 ;
  nanometer 0.000000001 ;
  nm 0.000000001 ;
  micron 0.000001 ;
  microns 0.000001 ;
  micrometers 0.000001 ;
  micrometer 0.000001 ;
  other 1.0 ;
  /*
  ** following items are U.S. Survey foot.
  */
  us_survey_foot 0.3048006096012192 ;
  us_survey_foot 0.3048006096012192 ;
  feet 0.3048006096012192 ;
  foot 0.3048006096012192 ;
  ft 0.3048006096012192 ;
  /*
  ** following items are related to Standard foot (0.3048).
  */
  international_feet 0.3048 ;
  international_foot 0.3048 ;
  inches 0.3048006096012192 / 12.0 ;
```


sptable.tab

This file contains definitions for the State Plane coordinate systems that are used in the United States of America. You do not need to modify this file to add a custom SRS. This is for advanced procedures not outlined in this chapter.

```
"ALABAMA" {
  "EAST"
    1
    NAD27 3101 -101
    0.6378206400000000e+07 0.6768657997291094e-02 -0.8505000000000000e+08
    0.9999600000000000e+00 0.0000000000000000e+00 0.0000000000000000e+00
    0.3003000000000000e+08 0.1524003048006096e+06 0.0000000000000000e+00
}
"ALABAMA" {
  "WEST"
    1
    NAD27 3126 -102
    0.6378206400000000e+07 0.6768657997291094e-02 -0.8703000000000000e+08
    0.9999333333333333e+00 0.0000000000000000e+00 0.0000000000000000e+00
    0.3000000000000000e+08 0.1524003048006096e+06 0.0000000000000000e+00
}
"ALASKA" {
  "ZONE NO. 10"
    2
    NAD27 6326 -5010
    0.6378206400000000e+07 0.6768657997291094e-02 -0.1760000000000000e+09
    0.1000000000000000e+01 0.5305000000000000e+08 0.5105000000000000e+08
    0.5100000000000000e+08 0.9144018288036576e+06 0.0000000000000000e+00
}
"AMERICAN SAMOA" {
  "----"
    2
    NAD27 9999 -5300
    0.6378206400000000e+07 0.6768657997291094e-02 -0.1700000000000000e+09
    0.1000000000000000e+01 -0.1401600000000000e+08 -0.1401600000000000e+08
    -0.1401600000000000e+08 0.1524003048006096e+06 0.9516931165862334e+05
}
"SAMOA (UNOFFICIAL)" {
  "----"
    2
    NAD27 9999 -5302
    0.6378206400000000e+07 0.6768657997291094e-02 -0.1700000000000000e+09
    0.1000000000000000e+01 -0.1401600000000000e+08 -0.1401600000000000e+08
    -0.1401600000000000e+08 0.1524003048006096e+06 0.9516931165862332e+05
}
"ARIZONA" {
  "EAST"
    1
    NAD27 3151 -201
    0.6378206400000000e+07 0.6768657997291094e-02 -0.1100100000000000e+09
    0.9999000000000000e+00 0.0000000000000000e+00 0.0000000000000000e+00
    0.3100000000000000e+08 0.1524003048006096e+06 0.0000000000000000e+00
}
```

Modify epsg.plb









The file `epsg.plb` contains all of the SRS definitions for the ERDAS IMAGINE Projection System. To add a new SRS to your ERDAS IMAGINE Projection System, create an entry for it in this file.

This entry will make use of information that is specified in the other four files (see above) that are used by the ERDAS IMAGINE Projection System.

The following figure shows an example of an SRS definition in the `epsg.plb` file. The following table describes the elements of the definition.

Figure 11: Example of an Entry in the File epsg.plb

```
"NAD83 (HARN) / California Albers (3311)" {
INTERNAL 3 "GRS 1980" "HARN" 0
2:5.9341194567807209E-001 3:7.0685834705770345E-001
4:-2.0943951023931953E+000 5:0.0000000000000000E+000
6:0.0000000000000000E+000 7:-4.0000000000000000E+006 "meters"
}
```

Element Color	Element Name	Description
	SRS Name	The name of the SRS as it will be displayed in the metadata.
	EPSG Code	The EPSG code for this SRS. You can check the EPSG Geodetic Parameter Registry at http://www.epsg-registry.org/ to verify that your number is unique.
	Projection Identifier	Identifies the projection in the sptable.tab file that matches the SRS you are adding.
	Spheroid Name	The name of the spheroid used for this projection. This spheroid must have an entry in the spheroid.tab file.
	Datum Name	The name of the datum used for this projection. This datum must have an entry in the spheroid.tab file. In the spheroid's definition in the spheroid.tab file, there is a list of the commonly associated datums. Find the name of the datum used for this SRS. Replace DATUM in the epsg.plb entry with the name of that datum.
	Zone Number	This is applicable only to the UTM and State Plane projections and is specified in sptable.tab . For everything else, this should be zero.
	Parameters	The parameters for this SRS, such as false easting, false northing, longitude of the central meridian, etc. The file sptable.tab indicates which parameters must be specified for this type of projection.
	Units	The unit of measurement used in the coordinate system. This unit must have an entry in the file units.dat .

1. Navigate to
<APOLLO_HOME>\tools\native\raster\etc\projections\epsg.plb

2. Open the file and add the projection parameters according to the structure described in [Figure 11](#). The parameters will vary depending on which projection the SRS is associated with. To find out which parameters are required for your SRS, open the file [sptable.tab](#).

- Each parameter in the `epsg.plb` file is preceded by a number and a colon, and the parameters are always numbered starting with 2.
- The parameter values must be entered in (scientific) E notation and require 17 significant digits and 3 digits for the exponent.

For example:

.006789247 should be entered as 6.78924700000000000E-003

12,450,000 should be entered as 1.24500000000000000E+007

- If the parameter represents an angle measurement, the value must be given in RADIANS, not degrees.

Create usersref.xml

This file is not included with the installation. You must create it to add a custom SRS. This file defines the datum, spheroid, and the custom projection parameters.

1. Open any text editor and type the following lines into the new file.

```
<?xml version="1.0" encoding="utf-8" ?>
```

```
<SREF>
```

```
</SREF>
```

2. Save the new file as `usersref.xml`.
3. Open the file `ionicsref.xml` and find an entry that uses the same projection as the SRS you want to create.
4. Copy the entire entry beginning with a **<PROJCS ID..** tag and ending with the tag **</PROJCS>**.
5. Paste the entry into the `usersref.xml` file, between the **SREF** and **/SREF** tags. Use this as a template to define your custom SRS.
6. Change the **PROJCS ID** number to the EPSG number you are using for this custom SRS.
7. Change the **NAME** value to the name you are using for the custom SRS.

8. The **UNIT ID** is a code that represents the units used in the coordinate system (feet, metres, kilometers, etc.) The valid codes are located at the top of the `factorysref.xml` file.
9. The **GEOCS ID** represents the datum or spheroid used for the custom SRS. The valid codes are located in the `factorysref.xml` file.
10. The parameters vary depending on the type of projection you are adding. You can get the proper values for the parameters by referring back to the parameters that you entered in the `epsg.plb` file.



The values for the parameters are always given in RADIANS in the `epsg.plb` file. They must always be indicated in ARC SECONDS in the `usersref.xml` file.

1. Save and close this `usersref.xml` file.



If the `usersref.xml` file will be used with another program such as ERDAS APOLLO Style Editor or the command-line tool, copy it to a location with the `com\ionicsoft\sref\impl\resource` structure and add the root of that structure to CLASSPATH.

Integrate usersref.xml

Because ERDAS APOLLO is an enterprise software system with different components in different locations, you add the `usersref.xml` in several different places.



Use any advanced zip utility to open the archive files.

1. Copy the newly created `usersref.xml` to the `.jar` file in the following location.

```
<APOLLO_HOME>\webapps\erdas-apollo\profiles\advantage\WEB-INF\lib\cots-srs-1.3.jar\com\ionicsoft\sref\impl\resource\
```

2. For the Web Client, copy the newly created `usersref.xml` to the `.jar` file in the following location.

```
<APOLLO_HOME>\webapps\apollo-client\default\WEB-INF\lib\cots-srs-1.3.jar\com\ionicsoft\sref\impl\resource\
```

3. For the Data Manager, the archive file is in another archive file. First open the following archive file.

```
<APOLLO_HOME>\configuration\org.eclipse.osgi\bundles\7\1\cp\lib\cots-srs.jar
```

4. Copy the newly created `usersref.xml` to the second `.jar` file in the following location.

```
cots-srs.jar\com\ionicsoft\sref\impl\resource\
```

Modify coordinate_system_category.xml

Next, add the new SRS to a category in the coordinate_system_category file.



Use any advanced zip utility to open the archive files.

1. Edit the coordinate_system_category.xml file in three places. Add the references to the custom projection (two custom SRSs are shown in the example shown below) at the end of the file in each location.

```
<crs:category name="Custom">
  <crs:code name="MGI AT-Styria 31 / Bessel" value="EPSG:44431" />
  <crs:code name="MGI AT-Styria 34 / Bessel" value="EPSG:44434" />
</crs:category>
```

2. Navigate to:

```
<APOLLO_HOME>\webapps\erdas-apollo\profiles\advantage\WEB-INF\lib\cots-srs-1.3.jar\com\ionicsoft\sref\impl\resource\
```

open the archive file and make the changes to coordinate_system_category.xml as shown in the example in [step 1](#).

3. For the Web Client, navigate to:

```
<APOLLO_HOME>\webapps\apollo-client\default\WEB-INF\lib\cots-srs-1.3.jar\com\ionicsoft\sref\impl\resource\
```

open the archive file and make the changes to coordinate_system_category.xml as shown in the example in [step 1](#).

4. For the Data Manager, the archive file is in another archive file. First open the following archive file.

```
\configuration\org.eclipse.osgi\bundles\7\1\cp\lib\cots-srs.jar
```

5. Open the second archive file and make the changes to coordinate_system_category.xml as shown in the example listed in [step 1](#).

```
cots-srs.jar\com\ionicsoft\sref\impl\resource\
```

Rebuild and Deploy the Webapps

After you create and modify the files as described above, you must rebuild the ERDAS APOLLO webapps and deploy them in JBoss.

1. Stop JBoss.
2. Ant jboss. Open a command line windows and type:




```
Cd <APOLLO_HOME> [enter]
```

```
<APOLLO_HOME>\tools\ant\bin\ant jboss
```

3. Wait until the build completes.
4. Deploy the new webapps in JBoss.
 - Navigate to <APOLLO_HOME>\dist\jboss and copy erdas-apollo.ear
 - Navigate to <APOLLO_HOME>\jboss\server\default\deploy and back up erdas-apollo.ear
 - Copy erdas-apollo.ear from the \dist folder to the default/deploy folder.
5. Delete the JBoss cache by deleting the <APOLLO_HOME>\jboss\server\default\work folder.
6. Start JBoss.

Test the Custom SRS in the Data Manager

Follow these instructions to make sure that your custom SRS is added.

1. Launch the ERDAS APOLLO Data Manager.
2. In the Explorer View, right-click on the ROOT node and create an aggregate.
3. Locate the property **Default SRS** and click on it.
4. Click the  button that appears in the **Value** column. The Spatial Reference Systems dialog box opens.
5. Locate your custom SRS in the list on this dialog, highlight it, and click **OK**.
6. Click Save  from the tool bar.
7. Click the Crawl  icon to create a crawler that will load data into your ERDAS APOLLO Catalog.
8. When you are asked to specify the directory to crawl, select the directory that contains your custom SRS imagery.

If the crawler still does not recognize the dataset, check the parameters in the `usersref.xml` file or in the imagery.

Test the Custom SRS in the Web Client

Follow these instructions to make sure that your custom SRS is added.

1. Launch the ERDAS APOLLO Web Client.
2. Click the SRS box on the Info Bar above the map.
3. Select **-Select-**. The Spatial Reference Systems dialog box opens.
4. Look for your custom SRS in the list on the Spatial Reference Systems dialog.

Usage and Syntax of the SRS/CRS Parameter

At the time OGC defined Web Services specification, for Web Map Services, it had to address the syntax for expressing coordinate systems. OGC decided to rely on the EPSG (European Petroleum Survey Group) definitions, and to adopt a simple syntax: code:value, where code is one of "EPSG" and "AUTO". "EPSG" is for codes defined in the EPSG database, and "AUTO" is for Automatic projections, as defined in the WMS 1.1.1 specification appendix. Examples of commonly used values are:

- WGS84 geographic system: EPSG:4326
- UTM 32 North based on WGS84: EPSG:32632
- Automatic UTM, in meter, centered on (-100,45):
AUTO:42003,9001,-100,45

Such expressions can be used in requests (HTTP-GET and HTTP-POST), can be found in Service Metadata (capabilities) documents and in other XML outputs such as response to a DescribeFeatureType, or in a GML output.

The next phase, for OGC, was to try and extend the expressions of coordinate systems, as well as standardize the syntax. This led to expressing coordinate systems as URNs (Universal Resource Names, defined in RFC 2141 in 1997 by the Internet Engineering Task Force) so that they could be registered in a standardization institute like IETF.

At IETF, the notation for CRSes defined by OGC must have the form: urn:opengis:def:crs:authority:version:code . "authority" can be one of "OGC" or "EPSG". "version" will generally be empty. "code" is the value given by the authority. For the examples above, the corresponding URN syntax is:

- WGS84 geographic system: urn:opengis:def:crs:OGC::84
- UTM 32 North based on WGS84: urn:opengis:def:crs:EPSG::32632
- Automatic UTM, in meter, centered on (-100,45):
urn:opengis:def:crs:OGC:42003:1:-100:45

In ERDAS APOLLO, the old code:value and the new URN syntaxes are supported, as well in requests and responses as in the configuration files.

For additional information, refer [Creating a Custom SRS](#) on page 74.

Administration of ERDAS APOLLO

This chapter provides explanations on what can be administered as well as guidelines on how to invoke the services to ensure that they are properly configured.

Introduction

Administration consists of interacting with the running Web services. Administration of Web services is done to ensure that they are behaving normally or to update their configurations.

Administering the services is a convenient way to monitor or change their behavior without having to stop them. The users of those services should not be impacted by administration tasks.

This section describes how to administer the product servlets as well as the underlying data connectors to the servlets.

Types of Administration

The current version of the product provides several different methods, tools and tasks to administer the services.

Some of the methods are Web-enabled as "checks" to the running services. These methods consist of parameters that can be given to the servlets (see [Checks](#) below). This allows the services to perform administration processing such as re-initialization, output version, license, debug information, or removal of cache files.

Several tools provided must be started using the system's command line. These tools perform automatic creation of indexes, environment or connection checks.

Finally, some administration tasks can only be done by manually modifying configuration files. They permit making more profound changes in the servlets behavior, such as the compression threshold, the metadata repository location or the styles library location.

Servlet-Engine Level Configuration Parameters

Most of the up-to-date servlet engines are configured in the same XML file, named "web.xml". This file is located in the WEB-INF directory of the current web application. If this web application is deployed as a web archive (WAR) file, similar to the ERDAS servlets, the web.xml file is in the WEB-INF directory in that war file.

For ERDAS servlets, configure the *servlet URL pattern and startup class* and define the *provider factory file (providers.fac)* location in the web.xml file. Other java-based properties can be defined in this file, but these parameters are not specific to ERDAS servlets. Types of java-based properties that can be configured in web.xml file include http.proxyHost and http.proxyPort. These types of parameters are added to the servlet configuration section of the web.xml file by adding a "<init-param>...</init-param>" block in the <servlet> definition section. (See the example below).

The providers.fac file default location is in the "resource" sub-directory of the servlet main class package. For example, the "wfs" servlet's startup class is named "com.ionicsoft.wfs.server.WFSServer". This means that its providers.fac default location is under "com/ionicsoft/wfs/server/resource". However, in the distribution, this location is overridden with the <APOLLO_HOME>/config/<servlet_type> directory. This allows easy provider modifications (either manually or with the Data Manager) without having to re-deploy the webapp.

Other initialization parameters can be given to the servlets. These parameters are not specific to ERDAS servlets but some of them could depend on the level of implementation of Java by the servlet engine.

Servlet-Engine Level Security

Several authentication mechanisms can be set up to request the client application to authenticate when querying a J2EE servlet engine or application server. Generally, the application servers allow three authentication mechanisms, BASIC, DIGEST and PKI, as well as the establishment of a secure channel. BASIC authentication on an SSL channel is recommended if integrity or confidentiality is requested; if not DIGEST authentication should be used on a classic HTTP connection.

The J2EE-based declarative security is based on the set up of an authentication "Realm" containing the definition of users and groups. Several options are possible depending on the type of application server used:

- Description in an XML file
- JDBC connection to a database
- LDAP directory

An extensive description of the configuration steps needed for coarse-grained security at the servlet engine level can be found in [Advanced Security](#) on page 195.

Servlet-Specific Configuration Parameters (providers fac)

Most of the configuration for ERDAS servlets is set up using the main providers factory file, providers.fac.

The providers.fac file needed by the servlet for data source access as well as global behavior tuning is taken from the <APOLLO_HOME>/config/erdas-apollo/<servlet_type> directory.

Example: Default providers.fac Location for ERDAS Servlets

For the "map" servlet the class name is "com.ionicsoft.wmtmap.servlet.GenericMapServlet". The default providers.fac file is stored in the directory <APOLLO_HOME>/config/erdas-apollo/providers/map.

For the "vector" servlet, the class name is "com.ionicsoft.wfs.server.WFSServer" and the default path is <APOLLO_HOME>/config/providers/erdas-apollo/vector.

For the "coverage" servlet, the class name is "com.ionicsoft.coverage.servlet.CoverageServlet" and the path is <APOLLO_HOME>/config/providers/erdas-apollo/coverage.

If more than one servlet of each type is needed or if the configuration directories have to be in another location, the providers.fac file can be re-located by changing the "ConfigUrl" parameter in the WEB-INF/web.xml file of the erdas-apollo webapp. If using a JBoss application server, the web.xml file is located in JBOSS_HOME/server/default/deploy/erdas-apollo.ear/erdas-apollo.war/WEB-INF. Its value is a URL to the file.



The URL types supported by HTTP and those supported by ERDAS servlets can both be used (See the Provider Configuration chapter, "URL parameters behavior" explanation for details on ERDAS-specific protocols).

Example: Providers.fac File re-location example

```
<init-param>
  <param-name>ConfigUrl</param-name>
  <param-value>obj:///./mydir/providerstest.fac</param-value>
</init-param>
```

Other initialization parameters can be given to the servlets. They are not specific to ERDAS servlets, but some of them could depend on the level of implementation of Java by the servlet engine.

Example:Servlet parameter to proxy requests through a firewall

```
<init-param>
  <param-name>http.proxyHost</param-name>
  <param-value>myProxyHost.com</param-value>
</init-param>
<init-param>
  <param-name>http.proxyPort</param-name>
  <param-value>8090</param-value>
</init-param>
```

Parameters in the providers fac File

The providers.fac file is used by ERDAS servlets to determine the behavior of and how to initiate the connection to the underlying data sources. In this chapter, only the servlet behavior defined in the "CONFIGURATION" part of the providers.fac file is explained. The data connection part is covered in the "Data Configuration" part of this guide.

Framework Configuration

Framework Configuration is achieved in the second part of the providers.fac file. Common behavior of the data providers can be defined by configuring the elements listed below in the global configuration section of the file, behind the <CONFIGURATION> tag.



There is generally only one instance of each of these parameters.

Table 5: Framework Configuration Elements

ELEMENTS	Descriptions and examples
CACHE	<p>Sets the directory in which caching will be achieved and the caching method applied. The "CONTROLDIR" boolean parameter controls the removal of expired directories under the cache directory. The default is false. If it is set, it will prevent from having embedded cache directories controlled by different servlets. See (1) below for more details.</p> <pre><CACHE CONTROLDIR="TRUE" USAGE="PERSERVLET" /></pre>

Table 5: Framework Configuration Elements (Continued)

ELEMENTS	Descriptions and examples
COMMANDS	<p>Allows restricting the use of the "debug" provider. Attributes, whose values are "true" and "false", are the names of the associated "cmd" values, STATE, GETLIST, DEBUG (for ON, OFF, DUMP). The DISABLED attribute allow to disabled all debug commands.</p> <pre data-bbox="461 457 1247 478"><COMMANDS STATE="FALSE" GETLIST="TRUE" DEBUG="FALSE" /></pre> <p>or</p> <pre data-bbox="461 646 863 667"><COMMANDS DISABLED="true" /></pre>
DEFAULT	<p>It defines some configuration variables. All attributes and all children elements can define a variable which can be accessed by the provider.</p> <pre data-bbox="461 793 630 814"><DEFAULT></pre> <pre data-bbox="461 865 977 886"><TmpPath>Temp path here</TmpPath></pre> <pre data-bbox="461 936 636 957"></DEFAULT></pre>
GARBAGE	<p>It has two attributes, LOOP and IDLE. It tells, in seconds, the looping time of the garbage thread and the maximum idle time before garbage collecting the providers. That means with default values that the garbage thread will check every 10 minutes if each provider has been used the last 10 minutes and will flush the ones that have been idle during this time. This parameter is optional and the default values are 10 minutes, for both attributes. To disable this feature, set the loop time to a negative value or 0.</p> <pre data-bbox="461 1329 932 1350"><GARBAGE LOOP="600" IDLE="600" /></pre>
GZIP	<p>Shows the maximum size a transiting file can be without requiring compression. The THRESHOLD attribute is expressed in bytes. By default, the value is 0 meaning that all files should be compressed. The ENGINECHUNK attribute, when set (default), allows the servlet engine to do "transfer chunked encoding" by itself.</p> <pre data-bbox="461 1644 863 1665"><GZIP THRESHOLD="2000000" /></pre>

Table 5: Framework Configuration Elements (Continued)

ELEMENTS	Descriptions and examples
LEGEND	<p>This parameter gives the location, file path or URL, to the Legend images to be used in LegendURL tags.</p> <pre data-bbox="459 388 1133 443"><LEGEND DIR="/home/ERDAS/legend" TEMPLATE="{absolute}{id}/{name}/{style}.gif" /></pre>
LOGCONFIG	<p>Configuration parameters relating to service logging. It allows the export of information to a repository of any type. This parameter and its deprecated logging system are now based on SLF4J and Log4J standards. See Logging on page 144 for a complete description.</p> <pre data-bbox="459 697 1073 835"><LOGCONFIG TYPE="FILE" FILENAME="D:/ERDAS/logs/wfslog" FILESIZE="1000000" MAXFILE="10" ENABLE="*" /></pre>
METADATA	<p>This parameter gives the location, file path or URL, to the Metadata files to be used in MetadataURL tags.</p> <pre data-bbox="459 1026 1019 1081"><METADATA DIR="/home/ERDAS/metadata" TEMPLATE="{absolute}{id}/{name}.xml" /></pre>
REGFUNC	<p>This parameter allows to declare a Java User Function for post-processing on WFS feature sets.</p> <pre data-bbox="459 1268 1292 1377"><REGFUNC ID="Summary" JCLASS="com.ionicsoft.test.wfs.functions.SummaryFunction"> <PARAM NAME="length" VALUE="5" /> </REGFUNC></pre>
STYLE	<p>It has two attributes: DIR that indicates the root directory of the rendering files and LIB that indicates the root directory of useful JavaScript functions. No default value is defined.</p> <pre data-bbox="459 1602 992 1656"><STYLE DIR="D:/ERDAS/rendering/" LIB="D:/ERDAS/renderlib/" /></pre>

Table 5: Framework Configuration Elements (Continued)

ELEMENTS	Descriptions and examples
TEMPMANAGER	<p>The directory where temporary files are stored. It defines the absolute path of the directory containing the generated temporary files. Otherwise the jdk temp directory is used. The variables expressed in the path are substituted, especially {tmp} or {TMP} are defined as the absolute path of the java temp directory (including the trailing separator). So you can use {tmp}local.</p> <pre data-bbox="461 491 878 516"><TEMPMANAGER DIR="c:/temp" /></pre>
TRANSLATOR	<p>It has four attributes, HOST, PORT, PROTOCOL and FILE. The first two (optional) parameters are used when the actual hostname and/or port of the server does not match those contained in the URLs returned to the client by the service. E.g., if the WFS builds a capabilities XML, the file will contain URLs that must have valid hostnames. If omitted, the PORT used will be the one defined in the request. The PROTOCOL parameter allows to mention another protocol (such as https). The FILE parameter allows to add something to the file part of the URL. By default, the current file part is used. Note that if FILE is set, the provider name is always added to the new file part.</p> <pre data-bbox="461 980 1192 1005"><TRANSLATOR HOST="myserver.erdas.com" PORT="8080"/></pre>
STORAGE	<p>It defines defines the persistent storage ara to save uploaded file. The absolute path is specified through the DIR attribute. The variables expressed in the path are substituted, especially {tmp} or {TMP} are defined as the absolute path of the java temp directory (including the trailing separator). So you can use {tmp}local.</p> <pre data-bbox="461 1295 862 1320"><STORAGE DIR="/home/area" /></pre>

1. The "INTERVM" caching method means that the cache directory can be shared by several processes or virtual machines. This share is achieved by a file, named lock.txt, that insures locking. Caution: If this file remains in the cache, the application may hang. The "PERSERVLET" option creates a cache that only pertains to the configured servlet. The "GLOBAL" option means that the cache is shared by all servlets. The first configuration wins. "NONE" means that no caching is used for the servlet.

WMS (map) Servlet

Among the Framework Configuration parameters listed above, some of them do not apply to the WMS servlet:

The parameter STYLE is useless as it is used only when applying portrayal styles to features in the WFS.

WFS (vector) Servlet

All of the listed tags apply to this servlet.

WCS (coverage) Servlets

The parameter LEGEND is currently not available.

Checks

Ensure that the services are configured and running properly before adapting them to the data. In this section, learn how to:

- Check the product license
- Check the product version
- Check the log enablement
- Enable the product's debug capabilities
- Check the data connections to the servlet environment
- Check the syntax of the providers.fac file

General Checks

Before configuration checks can take place, the administrator must make sure the underlying data server is up and running and the servlet engine is successfully started. HTTP requests containing set options and parameters can then be sent either to the overall servlet or to an individual provider.

Each type of data managed by an OGC-compliant Web Service gives the servlet its name, "map" for raster and proxy-WMS, "vector" for feature data and "coverage" for imagery. This name allows the building of the URL path of the servlet. e.g. `http://apollo.erdas.com/erdas-apollo/vector`. The next part of the URL is the Provider name. Each provider addresses a feature-, map- or coverage- source found in the data part of the providers.fac file.

Examples of Provider URLs

```
http://localhost/erdas-apollo/vector/ATLANTA_VECTOR  
http://localhost/erdas-apollo/coverage/ATLANTA_SINGLE
```



Note that a particular pseudo-provider, named "debug", is supported by the servlets. The debug provider allows global management of the servlet independent of any provider. Example: `http://localhost/erdas-apollo/map/debug`.

The table below is a description of debug options and parameters and the expected responses.

Table 6: Debug Request Parameters

cmd=gon	Puts the whole servlet in debug mode. Generally followed by cmd=gdump commands. Only applies if no <LOGCONFIG> element is defined.
cmd=gdump	Dumps the messages on the servlet. Must be preceded by cmd=gon. Only applies if no <LOGCONFIG> element is defined.
cmd=on	Puts the provider leveling debug mode. Only applies if no <LOGCONFIG> element is defined.
cmd=dump	Dumps the message on the provider. Only applies if no <LOGCONFIG> element is defined.
cmd=init	Re-initializes the servlet. It must be done if the providers.fac has been changed and needs to be re-read.
cmd=flush	Suppresses all the instances of all providers.
cmd=cache	Erases the cache content on the servlet side.
cmd=env	Dumps the servlet environment variables.
cmd=version	Provides information on ERDAS COTS version and external specs used (deprecated).
cmd=getlist	Displays the list of configured providers, along with the debug flag - ON or OFF. ON if a cmd=on or gon command was executed before.
cmd=license	Provides information about ERDAS license validity (host, date) and products
cmd=state	Displays the state of the servlet and the state of each provider

Note: Some or all of those debugging options can be deactivated in the configuration. See the "COMMANDS" parameter in [Servlet-Specific Configuration Parameters \(providers fac\)](#).

An example of the command to force reinitialization of all the providers remove cached requests, and obtain version information is:

http://localhost/erdas-apollo/vector/debug?request=debug&cmd=init,cache,version

An example of the command to display the log messages for a given provider along with environment information.

http://localhost/erdas-apollo/vector/MYPROVIDER?request=debug&cmd=dump,env

The "Service Tester" tool, included in ERDAS APOLLO distributions and documented in [ERDAS APOLLO Tools and Viewers](#), shows how to send HTTP-GET or HTTP-POST requests.



The providers.fac file is an XML file, with an associated – but not mandatory – DTD, named factory.dtd. This DTD should be used when editing and modifying the providers.fac file for input help and content validation.

Connections

JDBC connections

In previous chapters, connection information was provided for various data sources. Some of the data sources are reached using a Java connector and others use a JDBC connection. Each type of data source reachable via a JDBC connection uses one of several different drivers. Therefore, the connection string will vary accordingly. For example, connecting to Oracle using a thin or an OCI driver will change the syntax of the connection string.

Testing the JDBC connection to a data source can be done directly with the servlet. However, for the most common environment, Oracle, the JDBC Checkup Java class is provided for easier testing. Please refer to the Tools & Viewers chapter for a complete description of this tool.

Logging

ERDAS APOLLO logging is based on log4j, a Java-based logging utility which is a project of the Apache Software Foundation. Events logging is done through log4j Logger, with a given logging level. The logging outputs are done by Appenders. There are numerous Appenders available, with descriptive names, such as FileAppender, ConsoleAppender, SocketAppender, SyslogAppender, NTEventLogAppender and SMTPAppender. Multiple Appenders can be attached to any Logger, so it is possible to log the same information to multiple outputs; for example to a file locally and to a socket listener on another computer.

Appenders use Layouts to format log entries. A popular way to format one-line-at-a-time log files is PatternLayout, which uses a pattern string. There are also HTMLLayout and XMLLayout formatters for use when HTML or XML formats are more convenient, respectively.

In an ERDAS APOLLO JBoss installation, the log4j configuration file is located at <APOLLO_HOME>/jboss/server/default/conf/jboss-log4j.xml (single installation) or <APOLLO_HOME>/jboss/server/cluster/conf/jboss-log4j.xml (cluster installation). Both are configured by default to use the following two appenders.

- ConsoleAppender, redirecting logging to the JBoss console,
- DailyRollingFileAppender, writing logs to a daily-rolling file in the "log" folder of JBoss

Debugging

Debugging is possible if the logging configuration does not direct information to a file, a JDBC source or a JMS server. It can be requested at two levels - at the servlet and at the provider levels. When initializing a provider fails, debug at the servlet level. When initialization succeeds but it does not provide proper results, debug at the provider level.

To debug at servlet level, the debug mode has to be set on the "debug" pseudo-provider, by using the "cmd=gon" option ("g" is for "global"). Then, run the command followed by a request to dump the debug information. A sample sequence is:

Example:Sample Debug Sequence at the Servlet Level

```
http://localhost:8080/erdas-  
apollo/vector/debug?request=debug&cmd=gon  
http://localhost:8080/erdas-  
apollo/vector/myProvider?version=1.0.0&service=WFS&request=GetC  
apabilities  
http://localhost:8080/erdas-  
apollo/vector/debug?request=debug&cmd=gdump
```

To debug a single provider, the debug mode has to be set on the given provider, using the value "on" instead of "gon" for the "cmd" parameter. A sample sequence is:

Example:Sample Debug Sequence at the Provider Level

```
http://localhost:8080/erdas-  
apollo/vector/myProvider?request=debug&cmd=on  
http://localhost:8080/erdas-  
apollo/vector/myProvider?version=1.0.0&request=DescribeCoverage  
&typename=myCoverageType  
http://localhost:8080/erdas-  
apollo/vector/myProvider?request=debug&cmd=dump
```

Note: Some or all of those debugging options can be deactivated in the configuration. See the "COMMANDS" parameter in [Servlet-Specific Configuration Parameters \(providers fac\)](#).

An example of command to force reinitialization of all the providers with removal of cached requests and obtain version information is:

`http://localhost:8080/erdas-apollo/vector/debug?request=debug&cmd=gon`

An example of command to display the log messages for a given provider along with environment information is:

`http://localhost:8080/erdas-apollo/vector/MYPROVIDER?request=debug&cmd=dump,env`

Performance Tuning

After mastering the tasks within the ERDAS APOLLO documentation, additional configuration of the instance can be done to optimize performance. This chapter offers performance tuning pointers that address different aspects of the ERDAS APOLLO instance.

Introduction

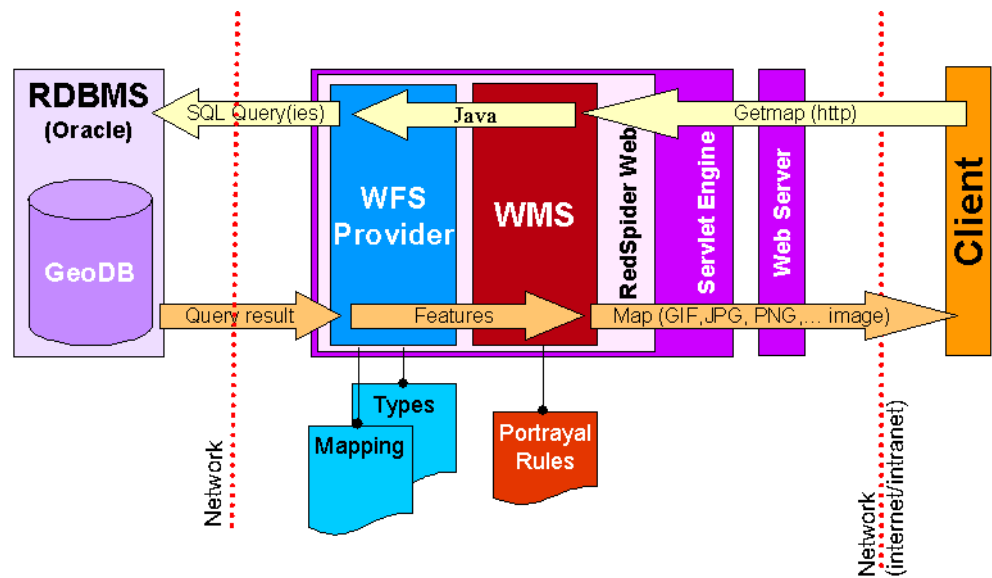
Getting the system up and running is the first step in building a production environment. As soon as it is up, performance quickly becomes the most important issue. It is important to quickly remove major inefficiencies and performance bottlenecks in order to ensure a level of performance that will satisfy users.

In this chapter, most of the focus is on tuning the output of GetMap requests, identifying at each step of map production which piece of the system is involved, and how its behavior can be optimized. Performance optimization process as it is applied to both vector files and spatial databases is differentiated from their portrayal onto maps. Also discussed will be performance issues related to the portrayal of raster, or coverage data. At the end of the chapter, additional tips will be provided for tuning the system's environment.

The chapter roughly follows the steps of producing a map (as in the picture below):

- The GetMap Request
- Data Extraction
- Portrayal
- Raster Sources
- Environment

Figure 12: The GetMap stream with an Oracle source



Tuning the GetMap Request

1. The more map layers requested, the more information received. At some scales, layers might hold an excessive level of detail justifying either their exclusion or pruning. This raises the question of how to make the layer choice seamless. ERDAS recommends configuring an OpenGIS Web Map Context document that holds the scale range parameters. Even though there are several Context builder tools on the market, it is recommended that either ERDAS APOLLO Style Editor or ERDAS Web Client be used to build Web Map Contexts.
2. The Coordinate Reference System (CRS) where the map is requested can be changed in the client application. However requesting coordinate transformations implies additional computing time that can be long when maps are large or transformation algorithms are not obvious. Therefore, it is recommended that map requests be in the native CRS of the service for improved performance. The user won't be aware of that information but the administrator who builds the initial "Web Map Contexts" should consider using the native CRS as much as possible.
3. Some servers also support "VendorSpecific" parameters that allow for optimized output. For example, when invoking an ERDAS servlet on vector data with GetMap requests, setting the USEBOX parameter to "FALSE" avoids the need to use the spatial index on the geometry, that can be time consuming for small scale requests.

4. The output format requested and the weight of each band of the image (8 bit, 24 bit), both have an impact on the response time. Requesting 24-bit non-compressed TIFF images will transport larger volumes than an 8-bit 256 color GIF. JPEG has varying levels of compression. PNG can be 8-bit or 24-bit, but Internet Explorer 6 does not support transparency in 24-bit PNG. So, properly choosing the output format with the "FORMAT" parameter and adequately adding the VendorSpecific "QUALITY" parameter can accelerate the output. For example, FORMAT=image/png&QUALITY=0 will produce an 8-bit PNG that takes a bit more time to calculate as most native data are 24 or 32-bit but the image is 3 times smaller.
5. Of course, 300x300 images will be output faster than 800x600 and smaller screens do not take advantage of large images. Do not hesitate to tune the client application's map view size.
6. If the Proxy WMS providers addressing third-party WMS's is configured, requests to that provider will timeout if the underlying WMS is not responding. It could lead to apparent overall client slowness, even if only one layer is slow.

Tuning the Data Extraction

In this section, vector data extraction scenarios for data persisted as both Shapefile and RDBMS will be considered. For raster, or coverage data, see [Tuning the Raster Data Sources](#). Tuning a database server is both art and science. Do not hesitate to request the help of a good database administrator.

Other tuning tasks are explained in the following sub-sections:

- tuning the RDBMS configuration
- tuning the database indexes
- tuning the native request

Tune the RDBMS configuration

1. For Oracle and other RDBMS, some of the elements influencing performance are:
 - Global Setting - (DB_BLOCK_SIZE, DB_BLOCK_BUFFERS, SHARED_POOL_SIZE, LOG_BUFFER, DB_WRITERS, ...).
 - JDBC Driver - (ERDAS WFS accesses Oracle through JDBC Thin and OCI drivers). OCI drivers are generally faster, but consider installing the Oracle Client on the Servlet Engine host.

- Oracle Version - e.g. Spatial indexes are optimized in more recent Oracle releases
 - Using Materialized Views
 - Oracle Partitioning
 - Spatial Indexing - Quad-Tree, R-Tree, GiST Tree (See next section)
 - Oracle RAC (Real Application Cluster)
2. When setting up a provider, the connection string must be included in the `providers.fac` file. This includes a URL-like expression with the database hostname and port, the user, the database ID, etc.

For Oracle, the JDBC driver allows you to add the **defaultRowPrefetch** parameter to obtain larger data sets. By default, only 10 rows are fetched at a time. Setting this parameter to 1000 when most of the requests are expected to output large volumes could improve the response time significantly. Note that it only impacts performance if there is network access between the servlet engine and the Oracle server.

For PostgreSQL and Microsoft SQL Server 2008, the JDBC driver allows you to add the **fetchsize** parameter if you want. If you do not add it, ERDAS APOLLO fetches 1000 rows at a time. If you do add it, and you set its value to a *positive* number, the number of rows that you specify will be fetched. If you add the parameter and set its value to any *negative* number, all of the rows will be fetched.

3. Each JDBC-type provider (Oracle, PostgreSQL) maintains a pool of JDBC connections to the database. The default pool size is set to 10 DB connections, and that pool is part of the ERDAS APOLLO product. Depending on the number of simultaneous connections, that size can be increased using the **poolsize** parameter. If running the servlets in an Application Server, delegate JDBC pooling to the application server pooling to take full advantage of its capabilities.

4. When running requests including spatial filters, significant performance improvements occur if you take advantage of the Oracle prepared statements mechanism. Indeed, this mechanism is managed by Oracle to let its engine reuse the execution plan of a request for the subsequent calls. By default in ERDAS APOLLO, the WFS requests are converted into "normal" SQL statements, not prepared statements. There are two ways to switch to the prepared statements syntax. The first is to set the GEOMTEXTSIZE parameter in your `providers.fac` to 1, instead of the default 500 value. It will force the usage of prepared statements as soon as a spatial filter is used in the request. The alternate solution is an Oracle option, by setting the Oracle `cursor_sharing` parameter to "force" in the relevant `init.ora` configuration. See http://www.quest-pipelines.com/newsletter-v2/cursor_sharing.htm for more detail on that option.

Tuning the Database Indexes

1. If using WFS providers on top of Oracle 9i, 10g or 11g, check that the indexes are properly defined and tuned:
 - For non-spatial properties, those which are frequently used in requests should be indexed to optimize searches.
 - For Point geometric properties (SDO_GEOMETRY fields), Quad-Tree indexes are the best, with appropriate SDO_LEVEL and SDO_TILES parameters. Oracle provides a tool that analyses the data and suggests the optimal parameters. Example: `CREATE INDEX ESA_FIRE_GEOM ON ESA_FIRE(GEOM) INDEXTYPE IS MDSYS.SPATIAL_INDEX PARAMETERS('SDO_LEVEL=10')`.
 - For other geometry types such as LineString, Polygons Oracle recommends using R-Trees. They are easily created by avoiding the SDO_TILES and SDO_LEVEL parameters in the index creation query, for example: `CREATE INDEX I_ANN_GEOM ON ANNOTATION(ANN_GEOM) INDEXTYPE IS MDSYS.SPATIAL_INDEX .`
2. For Shapefiles providers, create an R-Tree index using ERDAS's indexer tool (See [Shapefile RTree Builder](#)).
3. For PostgreSQL (or PostGIS) providers, three types of indexes can be used: B-Tree, R-Tree, and GiST indexes.



Note that frequent updates i.e., insert, delete, in the PostgreSQL database end up causing PostgreSQL to return strange and inconsistent results. It is recommended that the "vacuum" command be run on the database, to clean up the deleted tuples in the tables and fix the inconsistent results. This should be done following a successful backup.

4. For ArcSDE providers, use the native "ArcSDE grid" spatial indexing system or rely on the underlying database indexing systems (SDO or R-Tree for Oracle Spatial, ...).

Tuning the Native Request

A WMS GetMap request or a WFS GetFeature request will inevitably be converted into a query in the native language of the RDBMS. For Oracle, PostgreSQL and ArcSDE over Oracle or MS-SQL, the underlying query is in SQL. There are several ways to optimize the translation into native SQL query:

1. As explained in the first section, tune the GetMap request to limit the number of servers invoked and the number of layers requested. Each layer means one or more SQL queries the gain is obvious.
2. By default, a GetMap request will query ALL the columns of the table mapped with the WMS layer. When mapping the GML feature type definition to the database model, all of the columns do not have to be mapped. Only expose the properties which make real sense to the user. In particular, if only output maps, not feature collections, and simple styling, i.e., no classification, labelling, etc., are required, restrict exposure to the geometric set of columns possibly one or two others which are useful. This will considerably reduce the volume of data output from the database.
3. MapGen tags can be used to restrict requested properties. Be careful to keep all tags used in the portrayal styles, as well as the geometry. See [The Map Generation Transformer](#) on page 186 for more information on how to use the MapGen tags.
4. When a complex mapping is configured (one feature type mapped to more than one table), the service will often run several SQL queries for each request received. Performance becomes critical if the configuration is not optimal and indexes are not adequately created. See [Feature Mapping](#) for more information on complex mappings. When possible ERDAS recommends mapping one feature type to one table or view. Use the view mechanism if several tables are needed.
5. The MapGen tags also permits applying additional filtering clauses, with different expressions for each scale range using WHERE clauses.

6. By adding a GROUP BY clause to the native query, features can be grouped into smaller sets using the MapGen tags.



Look at the servlet's log file, that contains the native SQL queries sent to the database. Those queries can teach much about performance.

7. ERDAS APOLLO 9.3 can switch from one DB table to another depending on the scale factor given in the request. If several tables are established, each having the same properties names but a different content, the system calls the "lightest" table at the small scale first and then progressively calls "heavier" tables at larger scales. See "Scale Dependent Table" in the section [The Map Generation Transformer](#) on page 186.
8. Use prepared statements. Oracle is able to optimize the execution of the requests by calculating an Execution Plan before each request. This plan allows optimization using indexes, triggers, etc. However when each request is a literal with parts of it varying from one request to another, no optimization is possible. On the other hand, if the variant of the request is given as a set of parameters, named bind variables, the literal part is unchanged from one request to the other and optimization is possible. To activate preparedStatement use in the GetMap requests, set the minimum number of coordinates necessary to convert the request into a SQL query with prepared statements in the WFS to one (1). The parameter looks like: `<PARAM NAME="geomtextsize" VALUE="1"/>` . Note that the default value for that parameter is 500.

Tuning Portrayal

1. When designing portrayal styles for optimized performance follow these drawing guidelines: lines of width=1, single color instead of a pattern, no dashed lines are among the most impacting one.
2. If rendering geometries as symbols, favor SVG symbols over rasters like GIF and JPG. Consider converting raster symbols into SVG format.
3. Anti-clashing is the function that moves some objects to avoid overlay with others. This function slows down the process and is unjustified in most case. ERDAS recommends de-activating it.
4. Anti-aliasing provides smooth images but is also time consuming. It can be disabled through the ERDAS APOLLO Style Editor style properties panels.

5. Favor Fast 2D rendering (using Java 2D internal mechanism instead of going through the SVG-like tree. It does not apply to all cases of portrayal but for simple styling you can activate it by setting the property "parameters.directRenderingEnabled" to "true" in the Property style file. Other restrictions on using the Fast 2D rendering are described in "Portrayal Capabilities" and "Limitations".
6. At small scales, the level of detail in the rendering does not need to be as high as at large scales. Adapting the portrayal style to use the scale range will help. If server-side branching to one style or another is required, a specific Java rule using the Developer option of ERDAS APOLLO must be written.
7. Portraying on the client side means that a large volume of raw data (GML) is transported to the client application. Portraying on the server offers much faster output even though there is less flexibility on the client side. When adding a layer to the client, favor calling the service as a WMS instead of a WFS.
8. If addressing the service as a WFS and portray locally on the client side is still desired, reduced volume of raw data can be obtained by requesting the "Serialized Feature Collection" output format. The parameter value is "SERFC". It produces a binary stream that is encoded and decoded faster and has a much smaller volume than XML-encoded GML that still needs to be decoded on the client side.



NOTE: This is not yet adopted as an OGC specification and it is supported only by ERDAS's Web Services and its clients (Web Client, Style Editor).

9. When the geometries are large, multi-geometries or containing many points or lines, consider "generalizing" them, to lower the number of vertices or edges before sending the GML data to the client. Today, the generalization function can be explicitly called in a GetFeature request but it means that the full geometry is transferred to the invoker before being generalized. Therefore, it is best if a middleware application, running on the same machine as the services, does the request. Also, consider extending the portrayal rules to apply generalization when a GetMap is executed.
An alternative is to apply the generalization mechanism and store the generalized geometry as a new column in the table. Then, the MapGen tags, will ensure this geometry is used in lieu of the original one, for small scales.
10. SLD rules, while standard (OGC Specification), are more time-consuming than Property styles. Keep SLD rules when sending with a GetMap request but convert them into Property rules if they are permanently stored on the server.

Tuning the Raster Data Sources

1. Build a pyramidal provider that will address different mosaics depending on the scale range. ERDAS APOLLO includes a tool that takes an existing image or a set of tiled images and builds the mosaic with varying precision for various scale ranges. See [Typical Scenarios](#) and [Assembling Services and Combining Data](#) for instructions on how to set up a Pyramid WMS.
2. Choose a format that is the fast to parse, ECW or IMG.
3. If the raster data behind a single provider are big, a gain in performance can be realized by splitting them into tiles and having a IndexProvider configured. This provider implies having a world file per tile, or that information in the image header in case of GeoTIFF, and building the index file, generally a GML file. See [IndexProvider scenario](#) for more information on how to set up an IndexProvider.

Tuning Parameters and Configuration for WCS GIO Decoders

ERDAS APOLLO adds support for new formats by adding new decoders in the Coverage framework. GIO based WCS decoders are coverage plug-ins into the WCS Coverage framework. The GIO decoder is a complex raster sub-system by itself that feeds raster and metadata of the imagery to the WMS and WCS services. GIO decoders (Raster and Coverage decoder implementations) wrap native ERDAS IMAGINE raster engine using JNI and use the NCI (Native Code Isolation) framework to provide metadata and raster data.



As of APOLLO 11.0, the GIO decoders are not available on Linux platforms.

NCI framework is a set of libraries for managing a pool of Java processes (that use native code via JNI) isolated outside of a server JVM.

The Raster Decoder Server (RDS) is an external process to access raster datasets that can be pooled and managed using NCI Process Manager service. The ProcessManager service is responsible for spawning a new instance of RDS or utilizing one from a pool of existing RDS process in order to fulfill imagery request. The datasets cached, size of the pool, memory allocated, and other options are configurable via the following files: `rds.properties` and `processmanager.properties`.

The `rds.properties` file is located in the file `nci-rds.jar` that can be found in the directory `<APOLLO_HOME>\tools\native\nci`. To gain access to the file you can use a compression tool like WinZip, 7zip, WinRar, etc. Extract the `rds.properties` file, modify it and then replace the original contents of the jar file with the modified file.



It's important to understand that the application server must be stopped in order to modify the `nci-rds.jar` file. If the server is running you will encounter problems trying to save the modified `rds.properties` back into the jar as it will be in use by the RDS process. Failure to stop the server prior to modifying the JAR may result in corruption of the JAR and failure of the application.`rds.properties`:

Table 7: rds.properties Configuration elements

Elements	Description and examples
cache-these-many-datasets	Limit the number of datasets in cache to this value.
cache-per-band(MB)	Limit the number of blocks in cache, in Megabytes, so that it doesn't exceed this value.
raster-heap-size(MB)	Limit the maximum heap the native raster system can use. This helps prevent out of memory issues especially when generating pyramids for stipped TIFFs (or similar file formats) during crawling or serving WMS requests. Set to -1 if you don't want to it to use maximum available memory
warn-virtual-memory(MB)	This property is used by the native code when it reports the memory consumption of the RDS. Start freeing up resources (shrink the cached datasets) when the handle count reaches this value.
max-virtual-memory(MB)	This property is used by the native code when it reports the memory consumption of the RDS. Abort/quit RDS when the memory reaches this value.
warn-handle-count	This property is used by the native code when it reports the memory consumption of the RDS. Start freeing up resources (shrink the cached datasets) when the handle count reaches this value
max-handle-count	This property is used by the native code when it reports the memory consumption of the RDS. Abort/quit RDS when the handle count exceeds this value.
intf-impl	Add the interface-implementation fully classified names separated by hyphen ("-") as shown below: intf-impl=com.lggi.esp.coverage.decoder.raster.gio.GIORasterCoverageRemote-com.lggi.esp.coverage.decoder.raster.gio.GIORasterCoverageRemoteImpl

global-processmanager.properties:

This file is mainly for properties that are required to start RDS (Raster Data Server). The properties that exist in the file can be shared among multiple nodes in a cluster. This file exists in the shared folder.

Table 8: global-processmanager.properties Configuration elements

Elements	Description and examples
rds.max.read.threads	Only for ImageX decoders for ECW/JP2 - maximum number of processing threads. Default is 2.
rds.debugrds	To enable debugging of the RDS process. Set to true to enable debugging. Default is false.
rds.haltstart	Flag indicating if the RDS process should halt once it has started. Used for debugging. The default is false.
rds.max.pixel.request.size	Defines the maximum pixel request that GIORasterCoverageProxy can process. The default value is 25000000.
rds.proxy.directory	Defines the directory where proxy files are generated after pyramid generation of datasets.
processmanager.min.process.count	The minimum number of RDS processes to keep in the pool for servicing requests. The default is 1. For ImageX, the number is affected by rds.max.read.threads.
processmanager.max.process.count	The maximum number of RDS processes to keep in the pool for servicing requests. The default is 5. For ImageX, the number is affected by rds.max.read.threads.
processmanager.Keepalivetime.inmins	The time in minutes that an RDS process should remain in the pool before being cleaned up if there is no activity. The default is 10.
processmanager.getprocess.timeout.inseconds	The time in seconds between the processmanager.getprocess.numretries properties to get a free process from the pool. The default is 30.
processmanager.getprocess.delay.inseconds	Deprecated
processmanager.getprocess.numretries	When a request is received by the ProcessManager, the number of times it will try to get a free process from the pool of RDS processes before failing and generating an exception. The default is 2.

Table 8: global-processmanager.properties Configuration elements (Continued)

Elements	Description and examples
wait.time.before.killing.rds.if.not.responding	Time in seconds that the ProcessMonitor waits for an ACK before it terminates an RDS process. The default is 10.
rds.gio.ecw.logging.level	The logging level for ECW running in GIO. Valid values are 0-3. Zero indicates to log only errors and exception; three indicates verbose logging. The default is 0.

local-processmanager.properties:

This file (like global-processmanager.properties) also contains properties required by RDS, but these properties are node specific. In a cluster, different nodes have their own set of properties. This file exists in <INSTALL_DIR>/jboss/server/{default} or {cluster}/deploy/erdas-apollo.ear/erdas-apollo.war/WEB-INF/classes.

Table 9: local-processmanager.properties Configuration elements

Elements	Description and examples
rds.classpath	Defines the classpath for the use by RDS when a new process is created. It references the .jar files located in the <APOLLO_HOME>/rds directory by default.
rds.security.policy	Defines the default java security policy for access to RDS resources.
rds.log4j.properties	Defines logger configuration used by RDS for logging errors. It is declarative by nature and can be customized to use any of the valid loggers provided by Log4J or a custom logger based on Log4J api.
rds.java.home	Specifies which jdk to use for rds gio processes.
rds.java64.home	Specifies which 64-bit jdk to use for rds, such as anything that can take advantage of 64-bit like ImageX decoder.
rds.jvm.options	Defines all JVM options used to fine tune the RDS process. Each option is separated by a space. Example: -Xms64m Xmx128m -XX:+AggressiveOpts
rds.gio.ecw.log.file	The log file for ECW running in GIO which logs any exceptions.

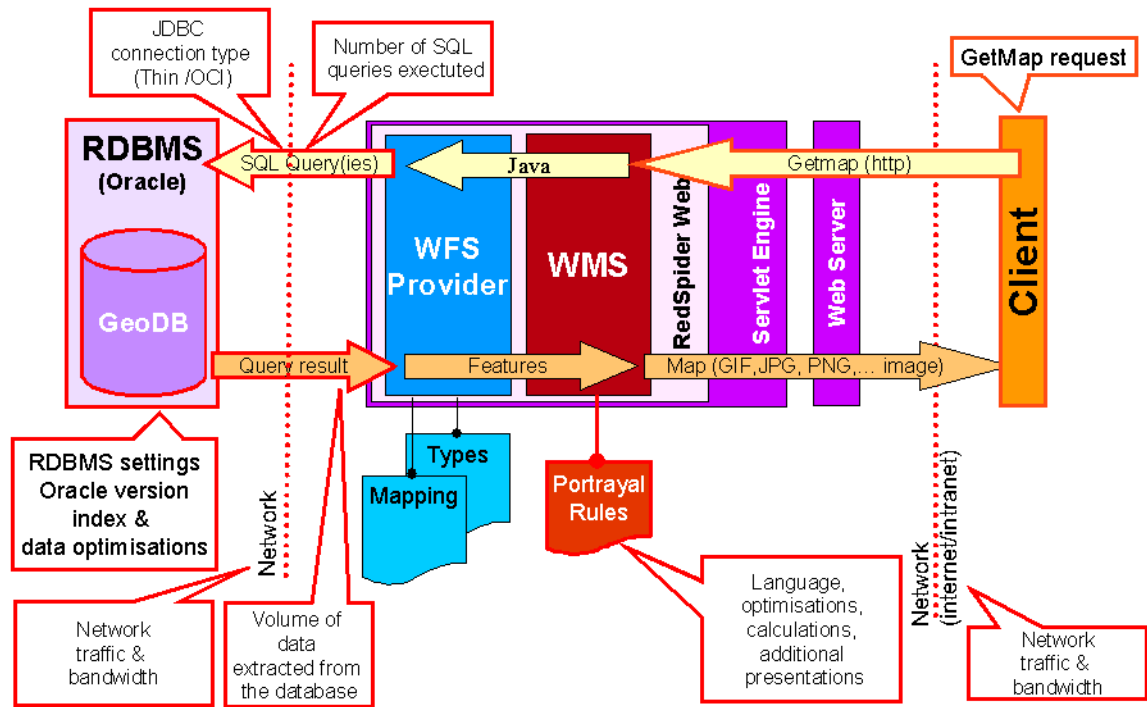
Tuning the Execution Environment

1. Java Virtual Machine (JVM) is the engine which activates the services and executes the requests. Please use Sun JDK 1.6.
2. The more RAM and the higher the heap size (parameter -Xmx) set to the JVM, the less swapping and garbage collection time will be consumed. Except if you have a good expertise on application servers tuning, DO NOT set the -Xms parameter, it often leads to worse performance than if not set.
3. For heavy loaded services or large outputs, "OutOfMemory error" messages may be encountered even though the -Xmx parameter is properly set. Consider setting the "MaxPermSize" parameter as well, it appears to help better use the heap size. When set as an option to the Java program, the syntax is '-XX:MaxPermSize=128m' in order to set it to 128 MebaBytes.
4. Servlet Engines are not all the same. Apache Tomcat is light and is one of the most commonly used in the world. Application Servers (JBoss, WebLogic) have a bigger infrastructure and more functionality, but require more configuration and tuning. ERDAS APOLLO Server supports the following Servlet Engines:
 - Tomcat
 - JBoss
 - Oracle WebLogic
5. During setup, configuration and tuning of the web services, having accurate logs help fine-tune the configuration. Following that, set the log level to "Warning" or only "Error" in order to avoid writing hundreds of lines of text in the log files. Still better, configure the web services to log asynchronously using JMS, ideally in a JDBC source.
6. For large scale environments with 100's of requests per second, consider using clustering solutions at the database level and/or at the Application Server level.
7. Frequently updated files or directories, like the caching directory, the data or log files, should be stored locally on the server machine rather than on a network drive or path. It will limit the network traffic and ensure those updates do not impact the response times.

Conclusions

The figure below summarizes the various types of optimizations that must be considered.

Figure 13: The GetMap optimizations with an Oracle source



Using Apache Ant to Rebuild the Webapps

The Ant script has been built during the installation and is located at `<APOLLO_HOME>/build.xml`, in order to rebuild the webapps with their default configuration at any time.



Ant is installed along with the product, in the tools/ant folder. For automatic call of this tool, add `$APOLLO_SERVER_INSTALL/tools/ant/bin` to your PATH.

To do this, enter **ant** (if it is in your PATH) or **<APOLLO_HOME>/tools/ant/bin/ant** instead, in the prompt of `$APOLLO_SERVER_INSTALL` directory as shown below:

Example: Rebuilding Webapps using Ant

```
ant tomcat6

Buildfile: build.xml

init:

tomcat6:

configure-eas-tomcat6:

init:

configure-tomcat6:

clean:
    [echo] erdas-apollo::clean - remove temporary/intermediate
files

setup-default:
    [echo] erdas-apollo::setup-default - copy and configure
default webapp to b
uild dir
    [mkdir] Created dir: $APOLLO_SERVER_INSTALL\webapps\erdas-
apollo\build
    [copy] Warning: $APOLLO_SERVER_INSTALL\webapps\erdas-
apollo\default not found.

build-erdas-apollo:

configure-eas-full:
    [copy] Copying 825 files to
$APOLLO_SERVER_INSTALL\webapps\erdas-apollo\build
    [copy] Copied 183 empty directories to 1 empty directory
under $APOLLO_SERVER_INSTALL\webapps\erdas-apollo\build

configure-eas-light:
```

```

configure-eas:

copy-custom-providers-resources:

replace-tokens:

build-eas-tomcat6:

package:
  [delete] Deleting: $APOLLO_SERVER_INSTALL\dist\tomcat6\erdas-
  apollo.war
  [zip] Building zip:
  $APOLLO_SERVER_INSTALL\dist\tomcat6\erdas-apollo.war
  [delete] Deleting directory
  $APOLLO_SERVER_INSTALL\webapps\erdas-apollo\build

clean:
  [echo] apollo-client::clean - remove temporary/intermediate
  files

setup-default:
  [echo] apollo-client::setup-default - copy and configure
  default webapp to
  build dir
  [mkdir] Created dir: $APOLLO_SERVER_INSTALL\webapps\apollo-
  client\build
  [copy] Copying 1668 files to
  $APOLLO_SERVER_INSTALL\webapps\apollo-client\build

build-erdas-apollo:

replace-tokens:

build-eas-tomcat6:

package:
  [delete] Deleting:
  $APOLLO_SERVER_INSTALL\dist\tomcat6\apollo-client.war
  [zip] Building zip:
  $APOLLO_SERVER_INSTALL\dist\tomcat6\apollo-client.war
  [delete] Deleting directory
  $APOLLO_SERVER_INSTALL\webapps\apollo-client\build

BUILD SUCCESSFUL
Total time: 2 minutes 34 seconds

```



If you want to build the webapps for other application server, you just have to use a goal other than the default one:

- generic: generate war files for generic application servers
- jboss: generate ear files for the JBoss 4.2 application server
- tomcat6: generate war files for the Tomcat 6 application server

- `weblogic`: generate war files for the WebLogic 10.3 application server

Deploying WAR Files on Supported Servlet Engines

Web Applications are commonly packaged as a single file with the ".war" extension. During installation of your ERDAS APOLLO Server, the webapps are automatically generated and stored in the installation directory. The way to deploy one or more of those web applications in a servlet engine is specific to each of those products. This chapter gives guidelines on how to deploy your applications in some of the supported servlet engines.

As soon as something has to be modified in your web app (new license, patched library, ...), you first need to update the expanded directories in the installation directory. Then, you have to rebuild your war files and redeploy them. To rebuild the war files, see [Using Apache Ant to Rebuild the Webapps](#).

JBoss

See the deployment on JBoss in the *ERDAS APOLLO QuickStart Guide*.

Jakarta Tomcat

See the deployment on Tomcat in the *ERDAS APOLLO QuickStart Guide*.

ERDAS APOLLO Tools and Viewers



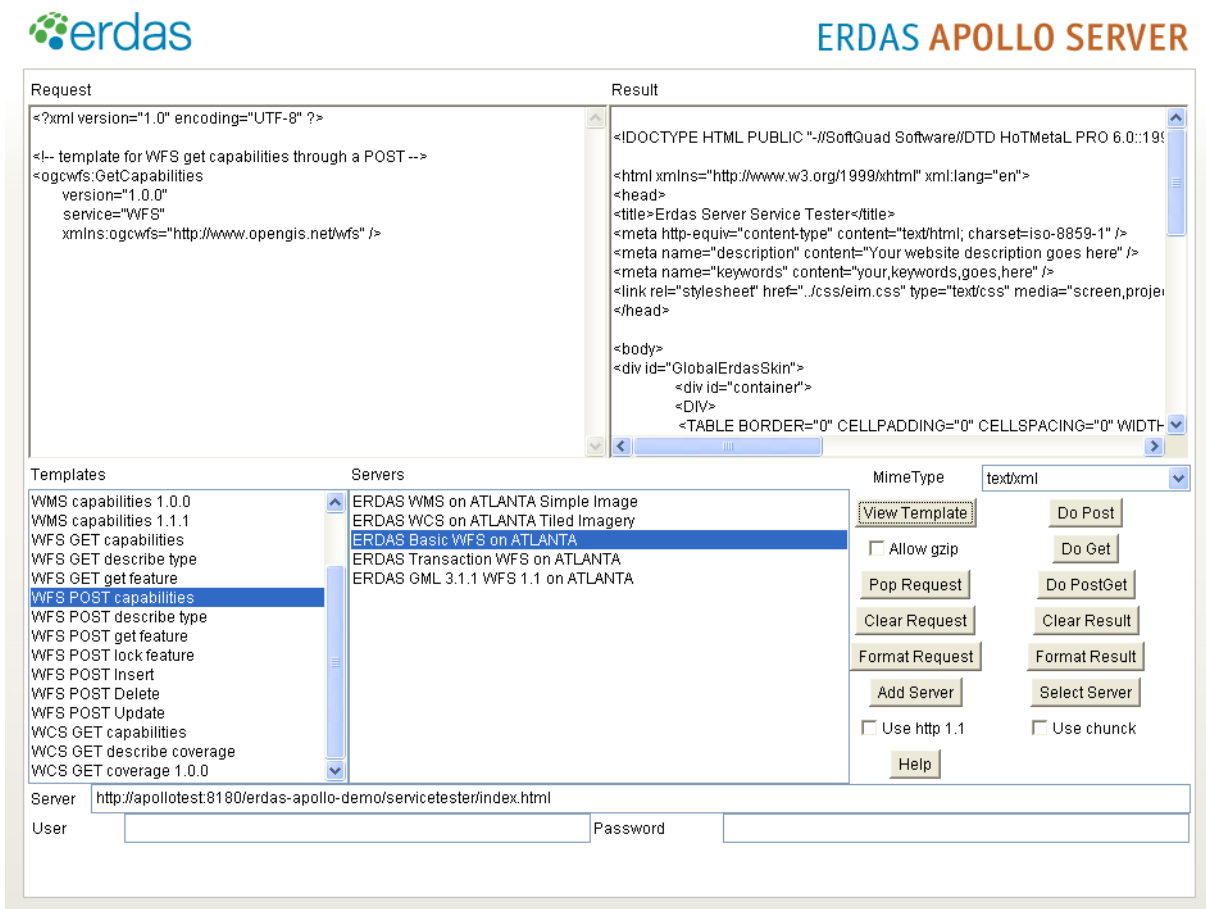
This chapter uses Unix syntax and scripts, generally suffixed by .sh when providing examples of commands. If using a Windows platform, remove the suffix or replace it with .bat.

Service Tester

The Service Tester runs as a Java Applet in the Web browser. It allows building and sending OGC-WMS and OGC-WFS requests to any compliant server. You must install the JAVA plug-in for your web browser. Refer to the JAVA website for the plug-in installer for your browser. The Service Tester tool is available at:

<http://myhost:80/erdas-apollo/servicetester/index.html> and an extensive online help provides a complete description of its functionality. The screenshot below is an example of what the tool looks like.

Figure 14: Service Tester applet



Some typical use scenarios of the Service Tester are:

1. In the server list zone, select "ERDAS Basic WFS on ATLANTA" and click the "Select Server" button.



You will find it at: <http://myhost:80/erdas-apollo/servicetester/index.html>.

2. In the templates zone, select "WFS POST capabilities" and click on the "View template" button. The request appears in the "Request" zone.
3. Click on "DoPost". The response, and XML document, appears in the "Result" zone.
4. The server URL can be encoded manually in the bottom text field.
5. Manually type the request in the "Request" zone.

Customizing Service Tester Templates

Since ERDAS APOLLO 9.3, the set of templates available in the lower-left zone of the tool can be customized.

The list of templates is configured in a text file, located in a sub-directory relative to the tool's HTML page under `com/ionicsoft/wfs/tester/template`. The default text file name is `templates.txt` but this name can be changed through the "TEMPLATEFILE" parameter in the tool's HTML page. Note that the template file name can be changed but not its location.

By default, the template file contains a set of templates which are stored in the `postapplet.jar` archive located in the same directory as the tool's HTML page. It is possible to extract some of those templates and change them or create custom ones. In this latter case, it will be necessary to update the `templates.txt` file to mention the custom templates. Templates can be removed from the list.



The template directory also contains a 'servers.txt' file, corresponding to the "SERVERLIST" parameter in the tool's HTML page. This file contains a list of predefined services that appear in the lower zone of the tool. List of visible services can be changed either in the HTML page or in that file.

Data Indexer

In order to publish a seamless collection of images or coverages, first configure each of them individually, format, world file, extent, etc., and then add an indexing system on top to allow fast and efficient extraction of the relevant items. To build those indexes as well as allow fast search in Shapefile documents, various tools are provided as part of the distribution:

- Indexing a set of coverages using a WFS.
- Indexing a Shapefile

Image Indexing with the Data Manager

Once a collection of images has been established, possibly with their respective world file, they can be viewed either as single images in a WMS, either as a set of layers, one per image or as a seamless collection of images. For those last two cases, the images need to be indexed so that at runtime the WMS can rapidly find each image based on the indexing properties. The Data Manager can be used to achieve this indexing, through the "Index Data" checkbox in the Create Service wizard. The indexing operation takes place as soon as you click the Finish button at the end of the wizard.

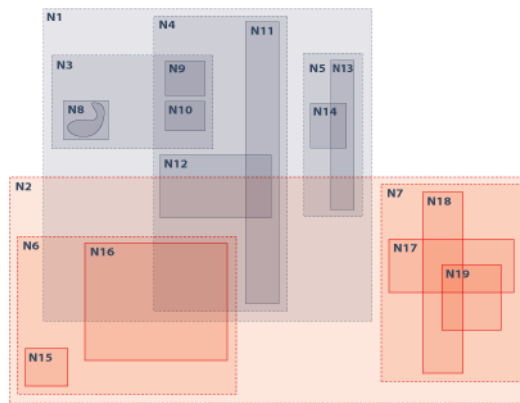
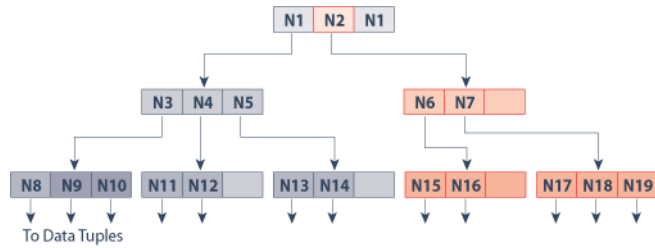
Coverage Indexer

Along with the dataset files, there can be metadata files (See the WCS Provider types for more information on coverage metadata configuration). To index them, use the Data Manager GUI.

Shapefile RTree Builder

In order to handle spatial data efficiently, a database system needs an indexing mechanism that will help it retrieve data items quickly according to their spatial location. However, traditional indexing methods, e.g., linear, hash, B-Tree, Quad-Tree are not well-suited for data objects located in multi-dimensional spaces. The schema below shows how R-Tree with its dynamic index structure meets this need.

Figure 15: RTree Structure



An R-tree is a height-balanced tree similar to a *B-Tree* with records in its leaf nodes containing pointers to data objects in the database. A *n*-dimensional rectangle determines the bounding box of the indexed spatial object. Each parent node entry contains the smallest rectangle that spatially surrounds all rectangles in the child node as shown in the schema above.

The RTree Builder tool creates a file with the name of the Shapefile and the .rtr extension. That file, named "index file" allows faster searches in the data, thanks to the "RTree" indexing mechanism.

The RTree Builder tool is part of the command-line tools provided in the distribution. It is also available behind the "Index Data" link in the Administration Console when managing a ShapeFile provider. To use the command-line tool, open a console window. If <APOLLO_HOME> represents the directory in which ERDAS APOLLO was installed, type:

```
cd <APOLLO_HOME>/tools/ows
./runrtreebuilder.sh <APOLLO_HOME>/data/erdas-
apollo/shapes/atlanta roads 30 15
```



Running the script with no argument will produce an explanation of each command:

```
Usage :com.ionicsoft.wfs.provider.shapev2.RTreeBuilder basedir
shapename maxcapacity mincapacity
basedir : base directory of the shape files
shapename : the shape file name pattern (without any extension)
maxcapacity : RTree maximum node capacity
mincapacity : RTree minimum node capacity
Example : com.ionicsoft.wfs.provider.shapev2.RTreeBuilder
          <APOLLO_HOME>\data\erdas-apollo\shapes\atlanta
futurelanduse 6 3
```



A recent performance analysis indicates that the values '100, 50' are best choice for most types of data.

Vector Services Utilities

At the time of setting up a ERDAS WFS provider, you need to have an XML Schema document for your feature types and a mapping document for the servlet to achieve the correspondence between your schema and the data source. In addition to the automatic generation capabilities found in the Administration Console, various command-line tools allow to build those documents as well as copy some data from one vector source to another.

Schema Generator

This tool is part of the command-line tools provided in the distribution. Its role is to build the database generation script based on a given XML Schema of feature types. It exists for two databases: Oracle and PostgreSQL.

To use it, open a console window. If <APOLLO_HOME> represents the directory in which ERDAS APOLLO Server was installed, for Oracle type:

```
cd <APOLLO_HOME>/tools/ows
cp <APOLLO_HOME>/config/erdas-
apollo/providers/vector/boston_ora.* .
./runoraschemagen.sh -schema boston_ora.xsd -mapping
boston_ora.xml -out boston_ora.sql
```

For PostgreSQL, the script is named runpgschemagen.sh.

Running the script with no argument will produce an explanation of each command:

```
command line arguments are
Common options
    -schema SCHEMAURL [-mapping MAPPINGURL]
where SCHEMAURL is a URL to the schema file
where MAPPINGURL is a URL to the mapping file
-verbose to output more information
```

```

-forceauto to use an auto mapping if none is provided
-usetargetspace to dump types belonging to the schema target
namespace
-ignoreknowntypes to ignore types derived from feature
association , geometry property types...
-usepkstring to specify the use of a string as the primary key
type in mapping auto SQL generation options
-bestfit tries to only use types reachable from the schema target
namespace
-out to output to the given file in utf-8 format
SQL generation options:
-----
-intermedia to generate Oracle Intermedia compatible layout
-remove to generate remove orders
-delete to generate delete orders
-lock to generate the tables used in the lock mechanism (no other
option is required)
Mapping generation options:
-----
-autogen to generate autogen mapping
Autogen Specific options:
-----
-infogen to generate info section with all operations enabled if
not present
-srs SRS to specify an srs for the info gen section
-allowfidinsertion to allow the insertion of fid during insert
operation
-lax to apply the laxist GML model verification (see also the
allowLAXGMLModel mapping tag)

```

The output of the command is displayed in the file which name is prefixed with "-out", or on the standard output.

From-SQL Generator

This tool is able to build mapping and schema files based on database models. It currently supports six data source types: Oracle, PostgreSQL, ESRI Shapefile, ESRI ArcSDE, Microsoft SQL Server 2008, and DGN files (through the FIO plug-in). For each database type, a different script is executed. That tool is also accessible through the Data Manager when managing vector services: the "Create Types and Mapping" link executes the same processing.

To use the command-line tool, open a console window. If <APOLLO_HOME> represents the directory in which ERDAS APOLLO Server was installed, for Oracle data type:

```

cd <APOLLO_HOME>/tools/ows
./runfromsqlora.sh

```



Running the script with no argument will produce an explanation of each command:

```
command line arguments are
-connection CONURL -table TEMPLATE [-schema SCHEMA][--mappingfile MF] [--typefile TF]
[-srs SRS] [--gml3] [--cl x] [--agressive] [--deprecated]
or
-factory FACFILE -name NAME -table TEMPLATE [--mappingfile MF] [--typefile TF]
or (for Shapefiles)
-PTH LOCAL -MULTIPATH MULTIPATH -table TEMPLATE [-schema SCHEMA][--mappingfile MF]
[--typefile TF] [--srs SRS] [--gml3] [--cl x] [--agressive] [--deprecated]
where CONURL is the connection string
where FACFILE is the factory file
where NAME is the provider name
where LOCAL is the path to the shapefiles directory (only applies to generator based
on Shapefiles)
where MULTIPATH is a semi-colon-separated set of paths (this parameter has precedence
on the "PATH" parameter)
where TEMPLATE is a table template (accepts '%', a single name or a comma separated
list)
where SCHEMA is the schema name
where SRS is the default srs to use (syntax can be code:value or
urn:opengis:def:crs:...)
where SRSSOVER allows to force the overwrite flag and ignore the data SRS
where MF is the output mapping file (default is mapping.xml)
where TF is the output types file (default is types.xsd)
where gmlX is gml3 to target gml3 feature model, gml3.2 to target gml3.2 feature
model
where cl is the GML-Simple Profile conformance level
where aggressive scans the SQL source to set the geometry type
where deprecated allows the use of gml3 deprecated geometries
```

It is important to understand that the `-connection` option lets you add other parameters such as `-schema`, `-srs`, `-gml3`, ... so that you can have a mapping and a types file built from just a connection to the database. In counterpart, the `-factory` option starts from a pre-defined provider, with a mapping and types file already built. And those additional parameters, instead of being passed as arguments to the command, are taken from the existing files. The main use case for this `-factory` option is to build a more complete and explicit mapping file based on a simple one such as one expression and SQL mapping.

Example:

```
cd <APOLLO_HOME>/tools/ows
./runfromsqlora.sh -connection
oracle://myhost/user+myuser/password+mypassword/SID+mysid
-table ROADS -schema ATLANTA -mappingfile roads_map -typefile
roads_typ -srs EPSG:2240 -gml3 -agressive
```

- For Oracle data, the `-table` and `-schema` arguments values must be uppercase.
- For PostgreSQL data, the script is named `runfromsqlpg.sh`. The `-table` and `-schema` arguments values must have the exact case (generally, lowercase).
- For Microsoft SQL Server 2008 data, the script is named `runfromsqlsqlserver.sh`.
- For ESRI ArcSDE data, the script is named `runfromsqlsde.sh`.



For ArcSDE connections to succeed, add the `jsdeXX_sdk.jar` file (and possibly the `jpeXX_sdk.jar` and `icu4j.jar` files) provided with the ESRI ArcSDE product in the `<APOLLO_HOME>/tools/ows/lib` directory. `XX` stands for the ArcSDE version.

- For Shapefile data, the script is named `runfromsqlshp.sh`. The `-table` values must be lowercase.



For this tool to successfully run on Shapefiles, reference a factory file (parameter `-factory`) with an entry for the shapefiles, or use the `-PATH` or `-MULTIPATH` parameter for a local shapefile directory.

- For DGN V7 and V8 data, the script is named `runfromsqlfme.bat`.

Remarks:

- The produced mapping file contains all the columns from the table. Manually remove the columns that are not to be published.
- The mapping file does not mention a Primary key column. It is set to `<NoPrimary>`. This can be replaced with a tag that has an actual set of columns that will be used to produce the "fid" or "gml:id" attribute.
- The mapping table does not mention any `<Lock>` column needed for transactional feature types.
- The mapping and types files are overridden if they exist.
- The mapping and types file paths can use absolute or relative paths, relative being relative to the tool directory. `..` can be used, and under Windows, `/` or `\` or mixed.

- The srs parameter is used to fill the mapping file with the Bounding Box information. The syntax used for the srs parameter will be set into the mapping file, either code:value or urn:opengis:def:crs:... . For databases where the srs value is set in the database (in ArcSDE, for example), the srs given as argument will be added to the mapping file. The original srs will be set first, unless the -srsover is set to true. In that case, only the srs given as argument is added to the mapping file. In addition, the "overwrite" attribute of the <SRS> element is set to true.
- The set permitted Operation is "Query". This needs to be changed if transactions are to be allowed.
- If some of the published columns from the mapping file are removed, also remove the published properties from the schema file.
- Under Oracle, the schema file gives geometries a type named "gml:GeometryAssociationType" or "gml:GeometryPropertyType" if the -gml3 option is used. Replace it with the actual geometry property type: gml:PointPropertyType, gml:LineStringPropertyType, gml:PolygonPropertyType or Multi-* geometry property types as described in the OGC WFS 1.0.0 specification. An alternative is to use the -agressive option to let the tool guess the appropriate geometry type.
- The GML-Simple Feature profile option (-cl n) only produces valid output if combined with the -gml3 option. The "n" argument is the conformance level, which value can be 0, 1 or 2. Currently ERDAS APOLLO produces the same output whatever level is chosen.
- An alternative to the command-line From-SQL generator tools is to use the Data Manager, in which the action "Create Types and Mappings" does the same job as those command-line tools.

WFS Loader

Often WFS feature collections, available either as GML files or in a WFS, need to be populated in another WFS. The WFS Loader tool helps achieve this transfer operation.

The WFS Loader tool is located in the <APOLLO_HOME>/tools/ows directory and can be executed from the system's command prompt. As soon as an instructions file (see below) has been completed and is passed as argument to the tool, run this tool in a console window to load features into the chosen WFS.

Sample request:

```
cd <APOLLO_HOME>/tools/ows
./runwfsloader.sh ./xmlscripts/GML2WFST.xml
```


Configuration of the input feature collection and the output WFS-T:

The configuration of the input feature collection and destination WFS-T can be accomplished by setting up an XML file, named XML Script. A set of sample scripts are provided in the distribution, under <APOLLO_HOME>/tools/ows/xmlscripts. In this directory, the GML2WFST.xml script allows the population of a set of GML files into a WFS-T. The WFS2WFST.xml script extracts the features from a WFS to load them into the WFS-T. The following example displays the content of the GML2WFST.xml script for a fictive set of sample data intended to populate a set of GML files.

```
<SCRIPT>
<!-- this script must be executed from the
<APOLLO_HOME>/tools/ows directory -->
  <TRACE VALUE="*" />
  <DEFINELOG
    TYPE="FILE"
    FILENAME="./xmlscripts/gml2wfstlog"
    FILESIZE="5000000"
    MAXFILE="10"
    ENABLE="*"
    ERRORLEVEL="100"
  />

  <!-- Relative "file" url is relative to the current directory
  when running the command -->
  <Factory NAME="file:///./providers.fac" />

  <FeatureServer ID="FS" NAME="MY_WFST_ORA" />

  <!-- We loop in a gml files directory -->

  <LOOPDIR VALUE="../../data/erdas-apollo/gml" ID="GML_FILE" >

    <Load FROM="GML_FILE" SCHEMA="FS" ID="FC1" ZIP="false"
    COUNT="true" />

    <Insert SIZE="1" ERROR="false" >
      <PARAM VALUE="FS" />
      <PARAM VALUE="FC1" />
    </Insert>

  </LOOPDIR>

  <!-- The <LOOPDIR> block can be repeated with other directories
  -->

  <Destroy NAME="FS" />
</SCRIPT>
```

XML Scripts Structure

The general structure is:

```
<SCRIPT>
<TRACE ... />
<DEFINELOG>
...
</DEFINELOG>
<Factory ... />
<FeatureServer ... />
<LOOPDIR ... >
  <Load ... />
  <Insert ... >
  </Insert>
</LOOPDIR>
<Destroy ... />
</SCRIPT>
```

The starting XML tag is always <SCRIPT> and the corresponding ending tag is </SCRIPT>.

The first two tags, namely <TRACE> and <DEFINELOG> are for logging and debugging purposes. This allows for the storage of information useful for debugging and system checks in log files.

To set the level of debug information required for all subsequent instructions in the script set the <TRACE> tags. The debug levels are: info, warning, minor, fatal, debug. The level "" signifies that all debugging information is to be displayed.

The <DEFINELOG> tag sets the log file and log level to be used when the subsequent Feature Servers are invoked. This tag has the same role and attributes as the <LOGCONFIG> tag found in the ERDAS Web Services providers.fac files. See [Servlet-Specific Configuration Parameters \(providers fac\)](#). This tag, when defined, will replace the setting in the default providers.fac files, thus grouping the logging information in a single file that is bound to the tool.

The next two tags, <Factory> and <FeatureServer>, relate to the destination WFS-T.

The <Factory> tag references the WFS configuration file, generally named providers.fac, that is used to access a WFS. In the example above, the file that must be referenced by a URL is named wfs.fac and it is in the current directory <APOLLO_HOME>/tools/ows. This file contains the definition of one or more WFS providers: provider type, connection string to a database, mapping and schema files. See [Data services](#) for more information about WFS provider configurations.

The <FeatureServer> tag allows opening a connection to a given WFS, identified by its provider name (the NAME attribute), and by the previous <Factory> definition. The "ID" attribute will contain an identifier that will be used in subsequent instructions.

The next tag, <LOOPDIR>, allows looping in the given directory as well as applying the actions defined in the block between <LOOPDIR> and </LOOPDIR>. The "VALUE" attribute provides the GML files with a directory path. The "ID" attribute is an identifier that will be used in subsequent instructions.

Note that if there are several GML files directories, the <LOOPDIR> block can be repeated.

In the loop, call the <Load> instruction to load the feature collection from a GML file. Then, call the <Insert> instruction to save this collection into the destination WFS.

The <Load> tag attributes reference the current GML directory. The "FROM" attribute is the looping variable in the GML directory. The features are loaded and validated against the Feature Types schema defined for the feature server referenced by the "SCHEMA" attribute. Do not compress the data (ZIP ="false") but make a count of the features (COUNT="true"). The loaded feature collection will have the "FC1" identifier.

The <Insert> tag saves the collection, referenced through the tag <PARAM VALUE="FC1"/>, into the WFS which is referenced by the tag <PARAM VALUE="FS" />. Note that the <Insert> tag has a "SIZE" attribute set to "1" and an "ERROR" attribute set to "false". This means that one feature will be saved at a time. The failing inserts will be logged but they will not interrupt the process. Once the features are set, the SIZE value can be raised to decrease the uploading time.

The last tag, <Destroy>, will release the feature server connection.

Customized Configuration of the WFS Loader Tool

1. Update the wfs.fac file to have a provider correspond to the WFS. ERDAS recommends defining a new provider with the proper connection string and the appropriate schema and mapping files. Ensure that the database tables are created before running the script including the "LOCKTIMEOUT" table used for WFS Locking. Check also that the mapping file allows the "Insert" operation to be performed on the features types to use.
2. Put the GML files in one or more directories under <APOLLO_HOME>/tools/ows/xmlscripts and check that the feature definition corresponds to the schema of the WFS.

3. Duplicate the GML2WFST.xml files and update the copy to adapt the <FeatureServer> and <LOOPDIR> tags according to steps 1 and 2.
4. Run the "runwfsloader" tool and pass it as argument the URL to the XML script either using an absolute URL (file:/// ...) or a relative path (./xmlscripts/...).

Features in a WFS

If the features are in a WFS instead of in a directory of GML files, it will be necessary to execute a GetFeature request on the originating WFS to place the resulting feature collection in the destination WFS-T.

A sample script corresponding to this situation is located at <APOLLO_HOME>/tools/ows/xmlscripts/WFS2WFST.xml. The code is:

```
<SCRIPT>

  <TRACE VALUE="*" />

  <DEFINELOG
    TYPE="FILE"
    FILENAME="./xmlscripts/wfs2wfstlog"
    FILESIZE="1000000"
    MAXFILE="10"
    ENABLE="*"
  />

  <Factory NAME="file:///../../config/erdas-
  apollo/providers/vector/providers.fac" />
  <FeatureServer NAME="ATLANTA_VECTOR" ID="FSFROM" />
  <FeatureServer NAME="TARGET_WFST" ID="FSTO" />

  <DefineRequest ID="R" >
    <GetFeature xmlns:gml="http://www.opengis.net/gml"
      xmlns:ogc="http://www.opengis.net/ogc"
      xmlns:wfs="http://www.opengis.net/wfs">
      <wfs:Query typeName="roads" >
        <ogc:PropertyName>*</ogc:PropertyName>
      </wfs:Query>
    </GetFeature>
  </DefineRequest>

  <GetFeature ID="FC1" COUNT="true" >
    <PARAM VALUE="FSFROM"/>
    <PARAM VALUE="R"/>
  </GetFeature>

  <Insert >
    <PARAM VALUE="FSTO" />
    <PARAM VALUE="FC1" />
  </Insert>
```

```

<Destroy NAME="FC1" />
<Destroy NAME="FSFROM" />
<Destroy NAME="FSTO" />

</SCRIPT>

```

In this script, a second <FeatureServer> tag has been defined for the originating WFS (ATLANTA_VECTOR). The same definition rules apply to this tag as for the destination WFS, except that this WFS does not need to be transactional.

A <DefineRequest> tag, which is an OGC-WFS GetFeature request, has been used to define the extraction request.

The query is actually executed through a subsequent <GetFeature> tag that references the WFS ID (FSFROM) and the DefineRequest ID (R). The resulting feature collection is given an ID (FC1), so that it can be used in the subsequent <Insert> instruction.

The ending <Destroy> tag has been repeated for each WFS and feature collection.

Pyramid and Mosaic Builder

When rich or complex data (vector, coverages, imagery) are available but someone just wants to serve basemaps, it is necessary to reduce or tune the data size or quality by building a pyramid or creating raster tiles based on an existing WMS. The following tools each allow to produce a set of images suitable for basemaps.

Pyramid Builder

This tool is able to create a pyramid of GeoTIFF images. The resulting pyramid can be used to setup a high-speed WMS on a layer of images.

The pyramid builder takes a directory of images, any format ERDAS Image Server supports, and builds n- layers of GeoTIFF files, each of these layers being decimated at a different level. The decimation level is the integer number by which the pixel width and height of each image will be divided to create the level of the pyramid. From one level to another, the tool uses the bilinear interpolation method.

From a performance point of view, it is not worth building a pyramid with lots of small images at one level. The tool is able to cluster the images if their pixel width or height is smaller than a specified value. The empty spaces in the mosaic will be filled with the color defined by the '-bc' parameter.

SYNOPSIS

```

runpyramid.sh -d [SOURCE FILE/DIRECTORY] -o [OUTPUT DIRECTORY] -
s [SOURCE SRS] -m [MINIMUM IMAGE SIZE]

```

```

        -bc ([RED],[GREEN],[BLUE]) [DECIMATION
LEVEL],[DECIMATION LEVEL], ... ,[DECIMATION LEVEL]

DESCRIPTION

create a pyramid of geotiff images, using images from [SOURCE
DIRECTORY]
source images srs is [SOURCE SRS], result images pixel size is
at least [MINIMUM IMAGE SIZE]
create one directory per [DECIMATION LEVEL] with image resolution
1/[DECIMATION LEVEL]
[DECIMATION LEVEL] should be an integer greater than 0, default
is 1

-d source directory or source file, required
-o output directory, optional
-s source srs, required except if source files are valid Geotiff
-m cluster the result layer of geotiff images, if their pixel
sizes are smaller than [MINIMUM IMAGE SIZE], optional
-bc background color used for the clustering,
[RED],[GREEN],[BLUE] are values between 0 and 255, default value
is (0,0,0), optional

```

For example, A user wants to create a pyramid from a layer of 9 images, each one is 20000*10000 pixels.

```

cd <APOLLO_HOME>/tools/ows
./runpyramid.sh -d ./layer/ -o ./pyramid/ -s EPSG:26910 1,4,16,64

```

This command will create 4 directories, each one with 9 GeoTIFF files of sizes: (20000,10000), (5000,2500), (1250,625), (312,156). The images at the highest level of the pyramid are small, which is not optimal. The user can use the "-m" cluster option to avoid this:

```

./runpyramid.sh -d ./layer/ -o ./pyramid/ -s EPSG:26910 -m 400 -bc
(255,255,255) 1,4,16,64

```

The highest level of the pyramid will then be clustered, producing a single GeoTIFF file of size (937*468). The empty spaces in the mosaic will be filled with the white color.

WMS Tiler

This tool produces a mosaic of Geotiff image tiles based on a request to a remote WMS service. The tool provides a wide set of options to let you accurately define the size of each tile, its accuracy, the scale at which the maps are extracted from the remote service,

In a second phase, the produced tiles can be indexed and then exposed as a new WMS service, which could provide much better performance than the original WMS, either because that original service is not always available, or because it manages vector or coverage data which are often overrated when a basemap is expected.

Note that this tool, if executed several times with a different scale value, will let you build a pyramid of tiles which content varies from one scale range to another. At the most, you could use as remote WMS a WMS over an OGC Web Map Context, this context addressing several different WMSes.

NAME

WMStiler: create Geotiff images tiles from a remote WMS

SYNOPSIS

```
./runwmstiler.sh -box [xmin],[ymin],[xmax],[ymax] -srs [srs]
-scale [scale]
    -tile [xSize],[ySize] -buffer [percent]
    -url [WMS url] -layers [layername1],...,[layernameN]
    -styles [stylename1],...,[stylenameN]
    -dir [out directory path] -file [output root file
name]
    -thread [thread count] -noclobber [true|false]
```

DESCRIPTION

create Geotiff images tiles from remote WMS layers (-url, -layers and -styles parameters).

The tool will create (h*v) Geotiff files of pixel sizes defined by the -tile parameter.

Each tile will cover the exact area necessary to match exactly the scale (-scale).

The numbers of tiles are computed so that the whole layer will cover at least the defined bbox (-box and -srs).

Tile request bounding boxes are increased by a percentage (-buffer), then cropped.

Each tile is written into a directory (-dir) using the name [-file]_xindex_yindex.

```
-box the box to cover
-srs the request srs (expressed as EPSG:code)
-scale the scale denominator of the resulting layer tiles
(e.g. 100000)
-tile the pixel sizes of the resulting tiles (default is
512,512)
-buffer the percentage of buffering around the requested tiles
(between 0 and 1, default is 0.25)
-url the url of the source wms
-layers the list of source wms layer names
-styles the list of source wms layer styles (default is
default)
-dir the output directory path
-file the output tiles root file name (default is [WMS
name]_Tile)
-thread the number of threads, default is one
-noclobber if true, existing tiles will not be re-
fetched/overwritten, default is false
```

EXAMPLE

```

./runwmstiler.sh -box -180,-90,180,90 -srs EPSG:4326 -scale
170640906 -tile 256,256
                -buffer 0.25 -url http://myhost:80/erdas-
apollo/vector/WORLDWIDE
                -layers cities,admin98 -dir ./WMStilerTest

```

Catalog Web Interface

The Catalog Web Interface is a web application offering to users the ability to manage - publish, search and browse - their data. It also offers administration tools for the Catalog service.

Basic workflows on this Catalog Web Interface are introduced in the User guide.

Log In to the Web Application

If browsing data is accessible for everyone, several operations are limited to users who are logged in (f.i. publishing content, detailed in [Publishing content](#)).

To log in to the application, you need credentials (login and password). These credentials allow you to fetch some corresponding roles that are needed to activate some specific actions of the web interface.

The login process itself is explained in [Authentication](#).

Searching and Browsing Content

Browsing the catalog is available from the *Browse* page (see upper left of the web interface). This page shows a form containing a drop-down list and a text field.

This form is quite simple to use. The drop-down list enables the user to filter his search to specific object type. Available types are: All, Services, WFS, WMS, WCS, Feature Types, Map Layers and Coverages.

In the text field, the user can enter keywords matching data that need to be discovered. Those keywords can be specified using advanced formatting and facilities. Here is an excerpt:

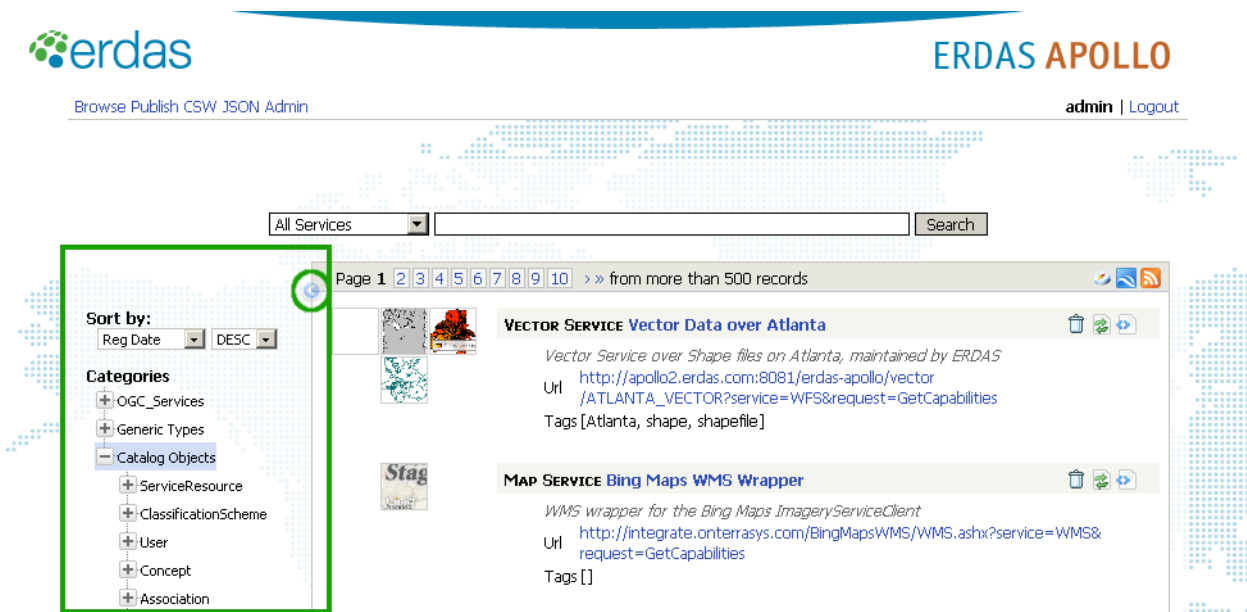
Table 10: Keywords Operators for Advanced Searches

Symbol	Usage	Effect
AND	Between two keywords.	Logical AND. Example: "road AND Atlanta".
OR	Between two keywords.	Logical OR. Example: "road OR Atlanta".
()	Surrounding keywords.	Logical Group. Example: "(road OR Atlanta) AND city".

Advanced Search

The advanced search panel can be opened by clicking the arrow on the left edge of the browse panel.

Figure 16: Advanced Search



This panel offers sorting options and also a tree view of the object types available in the catalog. Single clicking on an object type will search for records of that type. Double clicking displays the definition of the object type itself.

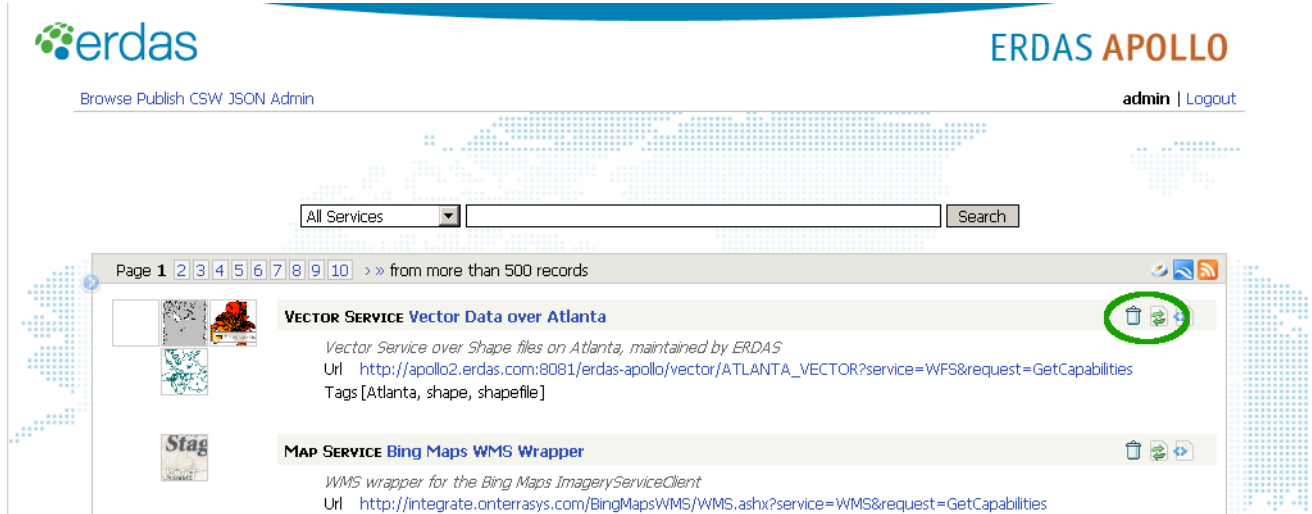
Use the contextual help (by pressing CTRL+ALT) to have more information on that panel.

Publishing content

This enables users to save their data into the catalog using an address and a corresponding type.

The user is considered as the owner of data published that way. It means that he's the only person (except the administrator) allowed to delete or refresh (re-publish) them. Those actions are displayed in the web interface using simple links:

Figure 17: Advanced Operations



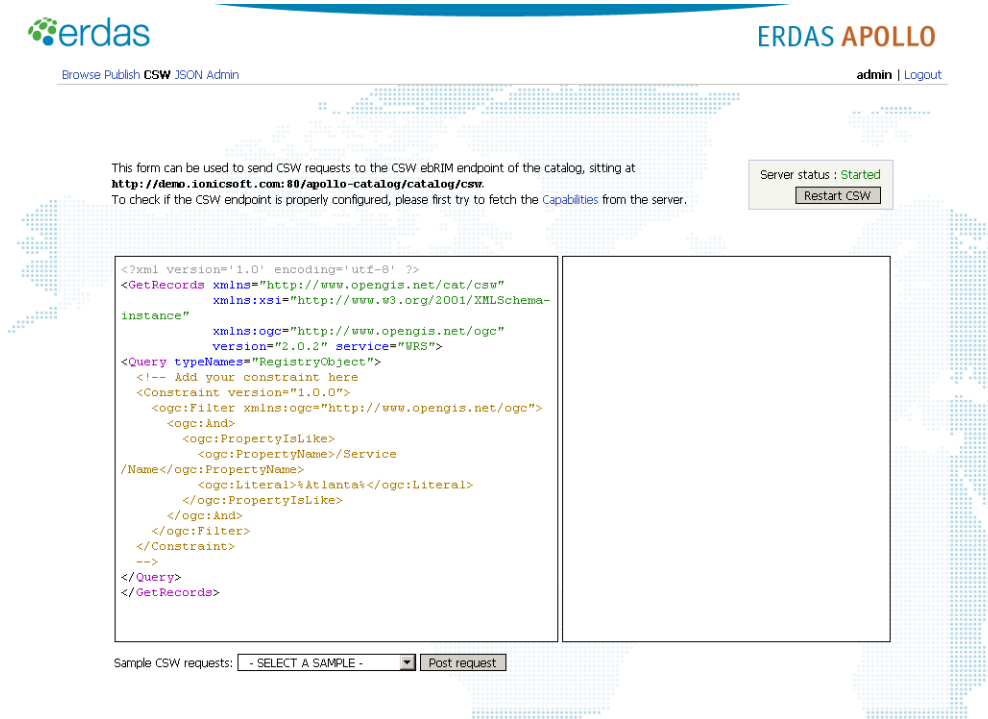
The process is explained in [Publish a service](#).

Testing the CSW endpoint

When authenticated with admin role, a 'CSW' tab appears at the top left of the web interface, linking to a CSW testing page.

This page offers a way to send CSW requests directly to the CSW ebRIM endpoint of the Catalog, sitting at <http://<serverURL>/erdas-apollo/catalog/csw>.

Figure 18: CSW Panel



The left panel can be used to edit CSW requests. When clicking on 'Post request', the request will be sent to the server, and the response from the server will be displayed in the right panel. A set of typical CSW requests are available in the drop down below the left form.

On the upper right of the page, a panel indicates the status of the CSW endpoint, i.e. whether it is started or not, together with a button to force a restart of the CSW endpoint. It must be noted that the CSW endpoint will start automatically on demand; this status and button are for debug purposes only, to force a restart and a cache flush of the CSW stack.

Administration options

The catalog web interface offers as well some facilities to manage the catalog. Those functionalities are in the *Admin* page (see upper left of the web interface).

Only users with an 'admin' role can access this page: if the user don't have this role, the *Admin* link is hidden to him.

Here are the functionalities:

- See the list of the roles.
- Re-index the keywords: this operation is helpful if, for whatever reason, the lucene index is not synchronized anymore with the catalog content. The main case is to be able to move the server without having to move the lucene index as well.
- Manage DB schemas: this displays a page that lists the currently installed DB schema(s) and their version. If the current DB schema is not in sync with the APOLLO software version, this page allows you to run the upgrade process.

The administrator's activity scope is extended to the data rights. Indeed, the administrator has all rights on data: he can delete or refresh everything, even he is not the owner of data.

Specifying the Storage Directories for Metadata, Thumbnails, & Pyramids

The ERDAS APOLLO system uses files to store information about the thumbnails, pyramid layers for catalog items, metadata for catalog items, and the output from geoprocesses executed in the web client. By default, those files are located in the directories specified in the table below.

Table 11: Location of Metadata, Pyramid Layer, Thumbnail, and Geoprocess Output Files

Resource Name	Directory
Thumbnails	<APOLLO_HOME>\storage\coverage\EAIM
Pyramid Layers	<APOLLO_HOME>\storage\pyramids\EAIM
Metadata	<APOLLO_HOME>\storage\metadata\coverage\EAIM
WPS Output	<APOLLO_HOME>\storage\wps\isrms\process_output

To store these files in different directories, you will need to edit some files in ERDAS APOLLO.

Changing the Storage Location for Metadata Files

1. Navigate to the directory
 <APOLLO_HOME>\config\erdas-apollo\providers\coverage.
2. Open the file `providers.fac`.
3. At the bottom of that file, you will find a block of configuration parameters.
 Find the one that says **METADATA TEMPLATE**

```
<CONFIGURATION>
  <LOGCONFIG
    TYPE="FILE"
    FILENAME="C:/ERDAS/EAIM_Server/logs/coverageLog"
    FILESIZE="1000000"
    MAXFILE="10"
    ENABLE=""
    DELETEONCLOSE="false"
    ERRORLEVEL="0"
    MEMORYSIZE="200"
  />
  <GARBAGE LOOP="600" IDLE="600" />
  <GZIP THRESHOLD="50000000" />
  <TRANSLATOR HOST="achamberstest" />
  <METADATA TEMPLATE="{absolute}{id}/{name}.xml" DIR="C:/ERDAS/EAIM_Server/config/erdas-apollo/metadata/coverage" />
  <CACHE DIR="C:/ERDAS/EAIM_Server/cache/erdas-apollo/coverage" USAGE="PERSERVLET" />
  <TEMPMANAGER DIR="C:/ERDAS/EAIM_Server/config/erdas-apollo/storage/coverage"/>
  <STYLE DIR="C:/ERDAS/EAIM_Server/config/erdas-apollo/rendering" VERSION="2" LOADER="sid" />
  <LEGEND TEMPLATE="{absolute}{id}/{name}_{style}.png" DIR="C:/ERDAS/EAIM_Server/config/erdas-apollo/legend/coverage" />
  <STORAGE DIR="C:/ERDAS/EAIM_Server/config/erdas-apollo/storage/coverage" />
  <JMX REGISTER="true"/>
  <SECURITY ALLOWEDPATH="C:/ERDAS/EAIM_Server/config:C:/ERDAS/EAIM_Server/data" />
  <DEFAULT>
    <GDALPath>C:/ERDAS/EAIM_Server/tools/native/gdal</GDALPath>
  </DEFAULT>
</CONFIGURATION>
```

4. Change the directory currently specified to the directory where you want ERDAS APOLLO to store your metadata files.
5. Save file `providers.fac` file and close it.

If you have already started your application server, you will need to restart it for the change you just made to take effect.

Changing the Storage Location for Thumbnail Files

1. Navigate to the directory
`<APOLLO_HOME>\config\erdas-apollo\providers\coverage.`
2. Open the file `providers.fac`.
3. At the bottom of that file, you will find a block of configuration parameters.
 Find the one the one that says **LEGEND TEMPLATE**.

```
<CONFIGURATION>
  <LOGCONFIG
    TYPE="FILE"
    FILENAME="C:/ERDAS/EAIM_Server/logs/coverageLog"
    FILESIZE="1000000"
    MAXFILE="10"
    ENABLE=""
    DELETEONCLOSE="false"
    ERRORLEVEL="0"
    MEMORYSIZE="200"
  />
  <GARBAGE LOOP="600" IDLE="600" />
  <GZIP THRESHOLD="50000000" />
  <TRANSLATOR HOST="achamberstest" />
  <METADATA TEMPLATE="{absolute}{id}/{name}.xml" DIR="C:/ERDAS/EAIM_Server/config/erdas-apollo/metadata/coverage" />
  <CACHE DIR="C:/ERDAS/EAIM_Server/cache/erdas-apollo/coverage" USAGE="PERSERVLET" />
  <TEMPMANAGER DIR="C:/ERDAS/EAIM_Server/config/erdas-apollo/storage/coverage"/>
  <STYLE DIR="C:/ERDAS/EAIM_Server/config/erdas-apollo/rendering" VERSION="2" LOADER="sid" />
  <LEGEND TEMPLATE="{absolute}{id}/{name}_{style}.png" DIR="C:/ERDAS/EAIM_Server/config/erdas-apollo/legend/coverage" />
  <STORAGE DIR="C:/ERDAS/EAIM_Server/config/erdas-apollo/storage/coverage" />
  <JMX REGISTER="true"/>
  <SECURITY ALLOWEDPATH="C:/ERDAS/EAIM_Server/config;C:/ERDAS/EAIM_Server/data" />
  <DEFAULT>
    <GDALPath>C:/ERDAS/EAIM_Server/tools/native/gdal</GDALPath>
  </DEFAULT>
</CONFIGURATION>
```

4. Change the directory currently specified to the directory where you want ERDAS APOLLO to store your thumbnail files.
5. Save the `providers.fac` file and close it.

If you have already started your application server, you will need to restart it for the change you just made to take effect.

Changing the Storage Location for Pyramid Files

The storage location for pyramid files is defined in three separate files:

- `im-providers.fac`
- `rds.policy`
- `processmanager.properties`

If you want to change the storage location of your pyramid files, you will have to change the definitions in each one of these files in order for pyramid handling to continue to work properly.

Changing the Pyramid Storage Location in `im-providers.fac`

1. Navigate to the directory
`<APOLLO_HOME>\config\erdas-apollo\providers\coverage.`
2. Open the file `im-providers.fac`.

There are two different blocks of settings in this file. One set is for users who are logged in to the APOLLO system. The other set is for users who are not logged in to the ERDAS APOLLO system. You must change the pyramid storage location in both blocks of settings in order for your system to work properly. You must also specify the same path in both blocks of settings.

To change the pyramid storage location in the block of settings for users who are logged in to the system:

1. Find the **PyramidDir** parameter inside the **EAIM** block of settings.
2. Replace the directory path in the PyramidDir parameter with the path to the location where you want to store the pyramid files.

```
<CREATE ID="EAIM" JCLASS="com.ionicsoft.wmtmap.provider.coverage.HierarchicalProvider">
  <PARAM NAME="name" VALUE="EAIM"/>
  <PARAM NAME="title" VALUE="ERDAS Apollo Advantage"/>
  <PARAM NAME="abstract" VALUE="ERDAS Apollo Advantage default Coverage Provider "/>
  <PARAM NAME="metaurll" VALUE=""/>
  <PARAM NAME="keywords" VALUE="Imagery, Archive, Ionic, ERDAS, Geotiff"/>
  <PARAM NAME="backgroundValue" VALUE="0"/>
  <PARAM NAME="srs" VALUE="EPSG:4326"/>
  <PARAM NAME="mode" VALUE="dynamic"/>
  <PARAM NAME="exposure" VALUE="SHOW_AGGREGATES"/>
  <PARAM NAME="maxcache" VALUE="500"/>
  <PARAM NAME="maxStitch" VALUE="500"/>
  <PARAM NAME="tmppath" VALUE="C:/ERDAS/EAIM_Server/config/erdas-apollo/storage"/>
  <PARAM NAME="indexingProvider" VALUE="./babel-application-context.xml"/>
  <PARAM NAME="indexingServer" VALUE="WRS"/>
  <PARAM NAME="indexingType" VALUE="BABEL"/>
  <PARAM NAME="queryables" VALUE="file:///C:/ERDAS/EAIM_Server/config/erdas-apollo/queryables.xml"/>
  <PARAMBLOCK NAME="contact">
    <PARAM NAME="Organization" VALUE="ERDAS, Inc."/>
    <PARAM NAME="AddressType" VALUE="Postal"/>
    <PARAM NAME="AddressBody" VALUE="5051 Peachtree Corners Circle"/>
    <PARAM NAME="City" VALUE="Norcross"/>
    <PARAM NAME="State" VALUE="GA"/>
    <PARAM NAME="PostCode" VALUE="30092"/>
    <PARAM NAME="Country" VALUE="USA"/>
    <PARAM NAME="Person" VALUE="Customer Support"/>
    <PARAM NAME="Voice" VALUE="+1 770 776 3400"/>
    <PARAM NAME="Email" VALUE="support@erdas.com"/>
    <PARAM NAME="OnlineResource" VALUE="http://www.erdas.com"/>
  </PARAMBLOCK>
  <PARAM NAME="GDALPath" VALUE="C:/ERDAS/EAIM_Server/tools/native/gdal"/>
  <PARAM NAME="owsinfourl" VALUE="C:/ERDAS/EAIM_Server/config/erdas-apollo/providers/coverage/eaim_md.xml"/>
  <!-- uncomment this to use an authentication mechanism is provided at the web-app container level -->
  <PARAM NAME="securityresolver" VALUE="container"/>
  <!-- Also change the property in rds.policy and processmanager.properties files related to
  gio pyramid generation where proxy files are created-->
  <PARAM NAME="PyramidDir" VALUE="C:/ERDAS/EAIM_Server/storage/EAIM"/>
  <PARAM NAME="VERYLARGETRIGGER" VALUE="2500" />
  <PARAM NAME="WMS_REPROJECTION_QUALITY" VALUE="75" />
  <PARAM NAME="WCS_REPROJECTION_QUALITY" VALUE="75" />
  <PARAM NAME="NODEEXCLUSION" VALUE="single"/>
  <PARAM NAME="decoderListUrl" VALUE="C:/ERDAS/EAIM_Server/config/erdas-apollo/providers/coverage/decoder.txt"/>
  <PARAM NAME="metadataListUrl" VALUE="C:/ERDAS/EAIM_Server/config/erdas-apollo/providers/coverage/metadata.txt"/>
</CREATE>
```

To change the pyramid storage location in the block of settings for users who are not logged in to the system:

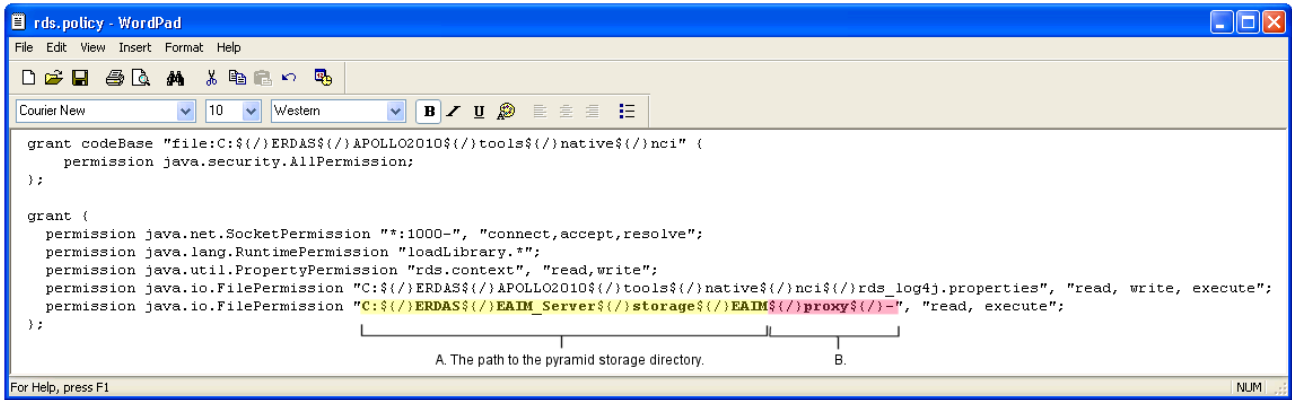
1. Find the **PyramidDir** parameter inside the **EAIM_PUBLIC** block of settings.
2. Replace the directory path in the PyramidDir parameter with the path to the location where you want to store the pyramid files.

```
<CREATE ID="EAIM_PUBLIC" JCLASS="com.ionicsoft.wmtmap.provider.coverage.HierarchicalProvider">
  <PARAM NAME="name" VALUE="EAIM"/>
  <PARAM NAME="title" VALUE="ERDAS Apollo Advantage Public"/>
  <PARAM NAME="abstract" VALUE="ERDAS Apollo Advantage Public default Coverage Provider "/>
  <PARAM NAME="metaurll" VALUE="" />
  <PARAM NAME="keywords" VALUE="Imagery, Archive, Ionic, ERDAS, Geotiff"/>
  <PARAM NAME="backgroundValue" VALUE="0"/>
  <PARAM NAME="srs" VALUE="EPSG:4326"/>
  <PARAM NAME="mode" VALUE="dynamic"/>
  <PARAM NAME="exposure" VALUE="SHOW_AGGREGATES"/>
  <PARAM NAME="maxcache" VALUE="500"/>
  <PARAM NAME="maxStitch" VALUE="500"/>
  <PARAM NAME="tmpPath" VALUE="C:/ERDAS/EAIM_Server/config/erdas-apollo/storage"/>
  <PARAM NAME="indexingProvider" VALUE="./babel-application-context.xml"/>
  <PARAM NAME="indexingServer" VALUE="WRS"/>
  <PARAM NAME="indexingType" VALUE="BABEL"/>
  <PARAM NAME="queryables" VALUE="file:///C:/ERDAS/EAIM_Server/config/erdas-apollo/queryables.xml"/>
  <PARAMBLOCK NAME="contact">
    <PARAM NAME="Organization" VALUE="ERDAS, Inc."/>
    <PARAM NAME="AddressType" VALUE="Postal"/>
    <PARAM NAME="AddressBody" VALUE="5051 Peachtree Corners Circle"/>
    <PARAM NAME="City" VALUE="Norcross"/>
    <PARAM NAME="State" VALUE="GA"/>
    <PARAM NAME="PostCode" VALUE="30092"/>
    <PARAM NAME="Country" VALUE="USA"/>
    <PARAM NAME="Person" VALUE="Customer Support"/>
    <PARAM NAME="Voice" VALUE="+1 770 776 3400"/>
    <PARAM NAME="Email" VALUE="support@erdas.com"/>
    <PARAM NAME="OnlineResource" VALUE="http://www.erdas.com"/>
  </PARAMBLOCK>
  <PARAM NAME="GDALPath" VALUE="C:/ERDAS/EAIM_Server/tools/native/gdal"/>
  <PARAM NAME="owsinfourl" VALUE="C:/ERDAS/EAIM_Server/config/erdas-apollo/providers/coverage/eaim_md.xml"/>
  <!-- uncomment this to use an authentication mechanism is provided at the web-app container level -->
  <PARAM NAME="securityresolver" VALUE="container"/>
  <!-- Also change the property in rds.policy and processmanager.properties files related to
  geo pyramid generation where proxy files are created-->
  <PARAM NAME="PyramidDir" VALUE="C:/ERDAS/EAIM_Server/storage/EAIM"/>
  <PARAM NAME="VERYLARGETRIGGER" VALUE="2500" />
  <PARAM NAME="WMS_REPROJECTION_QUALITY" VALUE="75" />
  <PARAM NAME="WCS_REPROJECTION_QUALITY" VALUE="75" />
  <PARAM NAME="NODEEXCLUSION" VALUE="single"/>
  <PARAM NAME="decoderListUrll" VALUE="C:/ERDAS/EAIM_Server/config/erdas-apollo/providers/coverage/decoder.txt"/>
  <PARAM NAME="metadataListUrll" VALUE="C:/ERDAS/EAIM_Server/config/erdas-apollo/providers/coverage/metadata.txt"/>
</CREATE>
```

3. Save file `im-providers.fac` file and close it.

Changing the Pyramid Storage Location in rds.policy

1. Navigate to the directory
<APOLLO_HOME>\tools\native\nci.
2. Open the file rds.policy.



3. Find the line that contains the path to the previous pyramid storage directory.
4. This entire line is actually pointing to a directory called **proxy** that exists inside the **original** pyramid storage directory. You need to change this line so that it points to a directory called **proxy** inside the **new** pyramid storage directory.

You can accomplish this by just changing part A shown in the diagram above. This is the path to the pyramid storage directory. Do not change part B.

Inside this file, paths must be written using $\{\}$ as the path separator rather than just the \backslash path separator that you typically use to write a Windows path.

5. Save and close the rds.policy file.

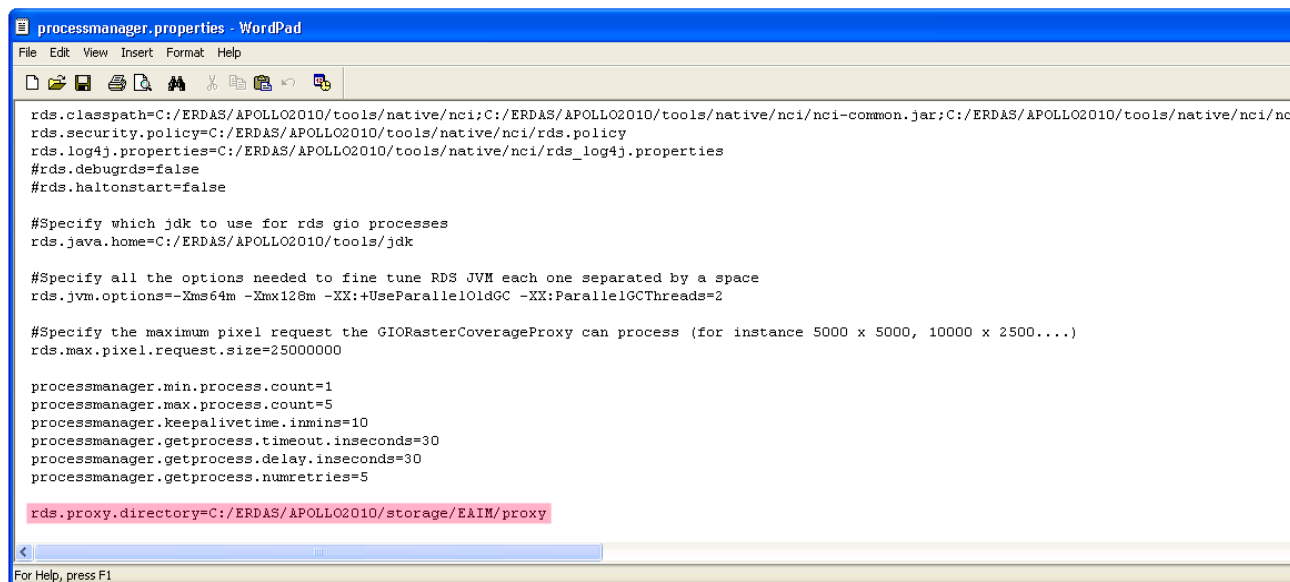
Changing the Pyramid Storage Location in processmanager.properties

1. If you are using JBoss as your application server, navigate to the directory
<APOLLO_HOME>\jboss\server\default\deploy\
erdas-apollo.ear\erdas-apollo.war\WEB-INF\classes.

If you are using WebLogic as your application server, navigate to the directory

<APOLLO_HOME>\dist\weblogic\
erdas-apollo.ear\erdas-apollo.war\WEB-INF\classes

Open the file local-processmanager.properties.



General Server Configuration

Install Properties

The `install.properties` file contains a lot of the most basic options that control how your ERDAS APOLLO Server product will work. These options were initially set by the installer program when you installed ERDAS APOLLO. All of the information

This file is located directly inside of the `<APOLLO_HOME>` directory.

The following table shows the properties that you can change in the file and describes the behavior that the property controls.

Table 12: Customizable Parameters in the Install.Properties File

Property Name	Description
platform.home	The ERDAS APOLLO installation directory.
apollo.server.host	server host name
apollo.server.port	server port number
apollo.shutdown.port	server shutdown port number
apollo.server.admin. port	server administration port number

Table 12: Customizable Parameters in the Install.Properties File

Property Name	Description
apollo.providers.vector.home	path to directory containing providers.fac for vector (WFS) offerings
apollo.providers.map.home	path to directory containing providers.fac for map (WMS) offerings
apollo.providers.coverage.home	path to directory containing providers.fac for coverage (WCS) offerings
apollo.providers.process.home	path to directory containing providers.fac for process (WPS) offerings
apollo.providers.wrs.home	path to directory containing providers.fac for catalog (WRS) offerings
apollo.providers.admin.home	path to directory containing providers.fac for administration
apollo.im.home	home directory for image management components (usually installation directory)
apollo.webapp.hostAndPort	commented out by default
apollo.webapp.name	commented out by default
hibernate.connection.driver_class	type of database connection for Hibernate, e.g. org.postgresql.Driver
hibernate.connection.url	connect string to database for Hibernate
hibernate.connection.username	database connection username for Hibernate
hibernate.connection.password	database connection password for Hibernate
babel.home	home directory for Babel catalog components (usually installation directory)
apollo.server.home	
release.babel.hibernate.dialect	dialect for database connections, e.g. com.erdas.rsp.hibernate.postgis.PostgisDialect
release.babel.jdbc.driver.fqn	name of JDBC driver for Babel database connection
release.babel.db.user	user id for Babel database connection
release.babel.db.password	password for Babel database connection
release.babel.db.url	connect string to database for Babel
babel.hibernate.dialect	dialect for Babel Hibernate database connection, e.g. com.erdas.rsp.hibernate.postgis.PostgisDialect
babel.db.user	user ID for Babel database connection

Table 12: Customizable Parameters in the Install.Properties File

Property Name	Description
babel.db.password	password for Babel database connection
babel.db.url	connect string to database for Babel
babel.jdbc.driver.fqn	
babel.db.host	host name for Babel database connection
babel.db.port	port number for Babel database connection
babel.db.sid	database SID for Babel database connection
babel.log.home	
ionic.catalog.product.name	
apollo.java.home	JAVA Home for ERDAS APOLLO
eaim.server.home	home directory for image management components (usually installation directory)
eaim.server.host	host name for image management service
eaim.server.port	host port for image management service
eaim.server.smtp.host	host for SMTP connection
eaim.server.smtp.port	port for SMTP connection
eaim.server.smtp.user	user ID for SMTP connection
eaim.server.wcs.url	URL for WCS service
eaim.server.wrs.url	URL for WRS service
eaim.server.wfs.url	URL for WFS service
eaim.server.wps.url	URL for WPS service
eaim.server.wps.transientprovider.url	URL for transient WMS providers used by WPS
eaim.server.catalog.url	URL for ERDAS APOLLO Catalog service
eaim.server.clipzipship.url	URL for Clip/Zip/Ship service
eaim.server.quartzinterface.url	URL for Quartz interface
eaim.server.crawler.user	user ID for crawling
eaim.server.crawler.pass	password for crawling
eaim.server.quartz.jdbc.delegate.class	class name for Quartz JDBC connection
apollo.im.home	installation directory
gio.home	path to GIO
platform.gio.home	also path to GIO

Table 12: Customizable Parameters in the Install.Properties File

Property Name	Description
platform.gio.arch	processor architecture for GIO
platform.gio.arch.mode	
eaim.server.streamedraster.access.url	URL for streamed (ECWP) raster access
eaim.server.streamedraster.access.enabled	if True, ECWP access is enabled
eaim.server.wps.gio.exedir	executable directory for WPS GIO
eaim.server.wps.gio.ismsdir	ISMS directory for WPS GIO
eaim.server.wps.gio.name	name of WPS GIO
eaim.server.wps.gio.exe	executable for WPS GIO
eaim.server.iws.home	path to IWS components, usually something like {\$platform.home}/tools/native/iws
ic.context.path	path to default context file
ic.logging.type	web client logging type (e.g. FILE)
apollo.client.components	name of properties file containing components information for apollo client (file must be in classpath). By default either apollo-im-components.properties (Professional) or apollo-im-components.properties (Essentials SDI)
apollo.client.contexts	name of properties file containing context information for apollo client (file must be in classpath)

Hiding Clear Text Passwords in Configuration Files

The passwords that are used to gain access to the ERDAS APOLLO Server and the ERDAS APOLLO database are stored in the configuration files in clear text and are shown in exactly as you typed them in.

If this is a security concern for your organization, you can hide these passwords.

Server Configuration Files

Some of the passwords are stored in ERDAS APOLLO Server configuration files for use by the system. You can encrypt the passwords in these files so that the system can still read them, but a human reader will not be able to read them and know exactly what they are.

The application server that you are using for ERDAS APOLLO Server determines which server configuration files contain passwords.

If you are using **JBoss**, the password is always found in the files `apollo-ds.xml` and `server.properties`.

If you are using **WebLogic**, it is only found in the `server.properties` file.

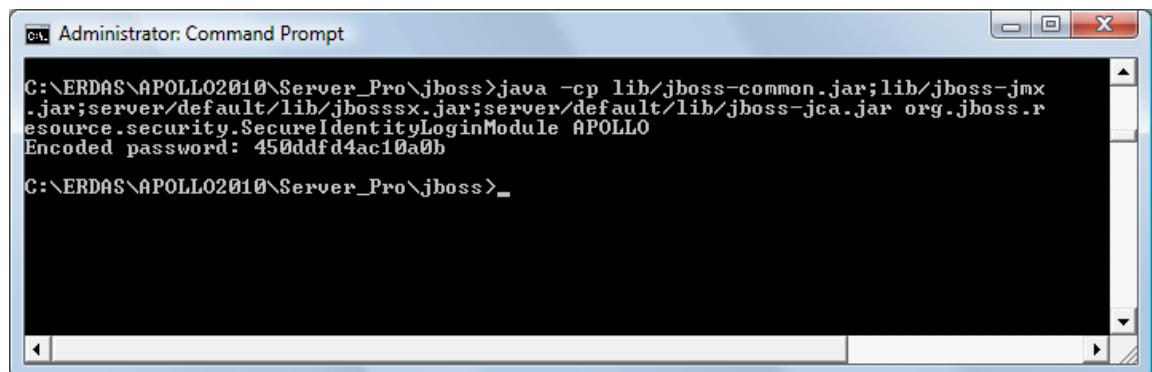
To encrypt the ERDAS APOLLO database password in the configuration file `apollo-login-config.xml`:

1. Navigate to the directory
`<APOLLO_HOME>/jboss/server/default/conf.`
2. Open the file `apollo-login-config.xml` for editing.
3. Open a command line window.
4. Navigate to the directory `<APOLLO_HOME>\jboss.`
5. Type in the following command at the prompt (or find it in the file `apollo-login-config.xml` and copy and paste it). Substitute the clear text database password for the `<CLEAR_TEXT_PASSWORD>` placeholder.

This will invoke the JBoss secure identity login module, which will encrypt your clear text password.

```
java -cp lib/jboss-common.jar;lib/jboss-jmx.jar;server/default/lib/jbossx.jar;  
server/default/lib/jboss-jca.jar  
org.jboss.resource.security.SecureIdentityLoginModule <CLEAR_TEXT_PASSWORD>
```

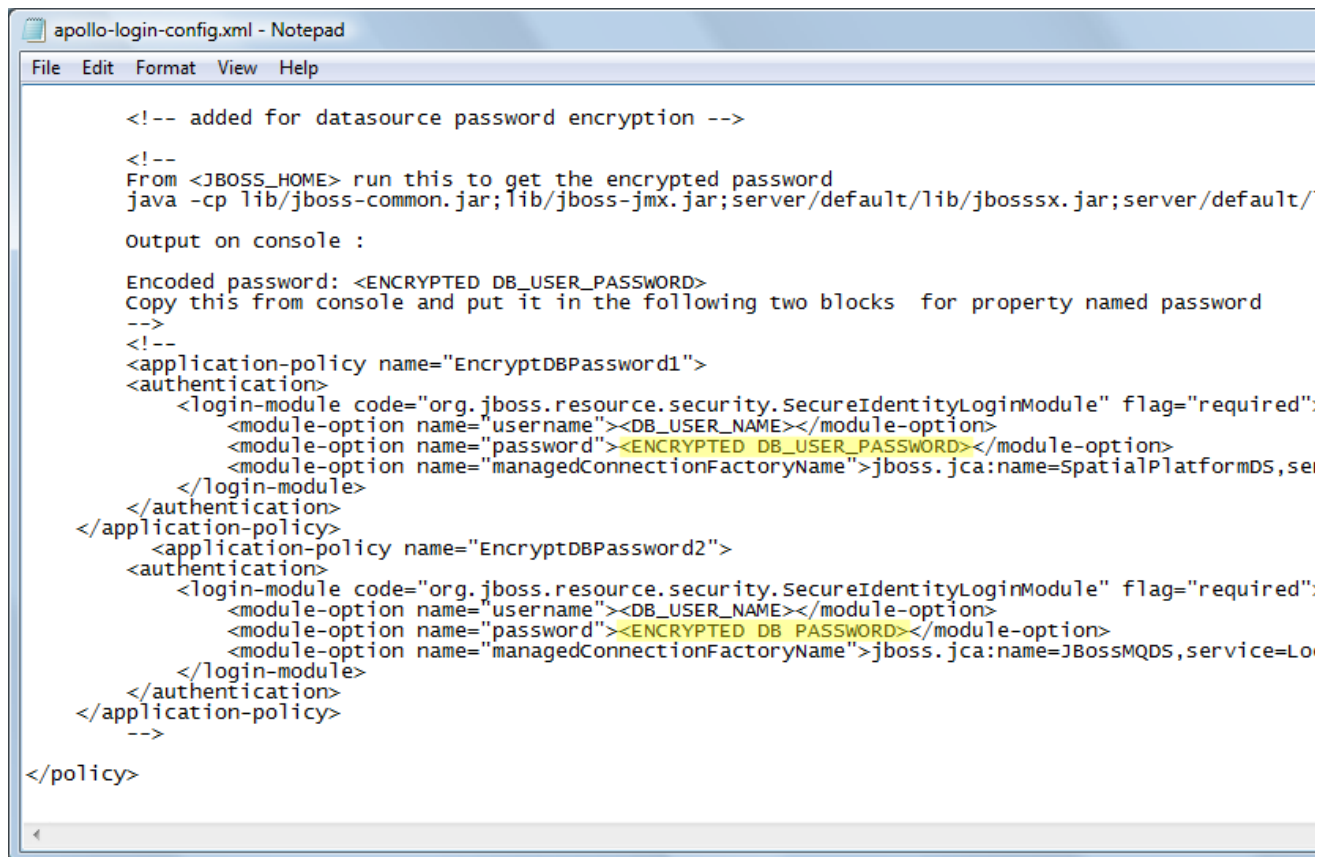
6. The secure identity login module displays the encrypted password in the command line window.



```
Administrator: Command Prompt  
C:\ERDAS\APOLLO2010\Server_Pro\jboss>java -cp lib/jboss-common.jar;lib/jboss-jmx  
.jar;server/default/lib/jbossx.jar;server/default/lib/jboss-jca.jar org.jboss.r  
esource.security.SecureIdentityLoginModule APOLLO  
Encoded password: 450ddfd4ac10a0b  
C:\ERDAS\APOLLO2010\Server_Pro\jboss>_
```

Leave the command line window open. You will need it again in one of the next steps.

7. Place this encrypted password inside the highlighted locations inside the `apollo-login-config.xml` file.



```
apollo-login-config.xml - Notepad
File Edit Format View Help

<!-- added for datasource password encryption -->
<!--
From <JBASS_HOME> run this to get the encrypted password
java -cp lib/jboss-common.jar;lib/jboss-jmx.jar;server/default/lib/jbosssx.jar;server/default/
output on console :

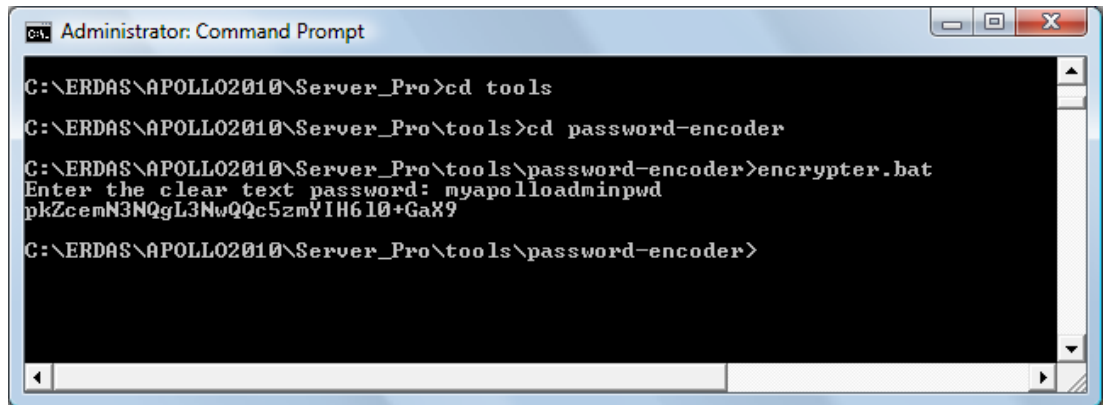
Encoded password: <ENCRYPTED DB_USER_PASSWORD>
Copy this from console and put it in the following two blocks for property named password
-->
<!--
<application-policy name="EncryptDBPassword1">
  <authentication>
    <login-module code="org.jboss.resource.security.SecureIdentityLoginModule" flag="required":
      <module-option name="username"><DB_USER_NAME></module-option>
      <module-option name="password"><ENCRYPTED DB_USER_PASSWORD></module-option>
      <module-option name="managedConnectionFactoryName">jboss.jca:name=SpatialPlatformMDS,se
    </login-module>
  </authentication>
</application-policy>
<application-policy name="EncryptDBPassword2">
  <authentication>
    <login-module code="org.jboss.resource.security.SecureIdentityLoginModule" flag="required":
      <module-option name="username"><DB_USER_NAME></module-option>
      <module-option name="password"><ENCRYPTED DB_PASSWORD></module-option>
      <module-option name="managedConnectionFactoryName">jboss.jca:name=JBossMQDS,service=Lo
    </login-module>
  </authentication>
</application-policy>
-->
</policy>
```

8. Save and close the `apollo-login-config.xml` file.

To encrypt the ERDAS APOLLO system passwords in the configuration file `server.properties`:

1. Open a command line window (or go to the one that is already open).
2. Navigate to the directory `<APOLLO_HOME>/tools/password-encoder`
3. Type `encrypter.bat` at the prompt and press Enter. This runs the password encoder tool, which will prompt for a clear text password.

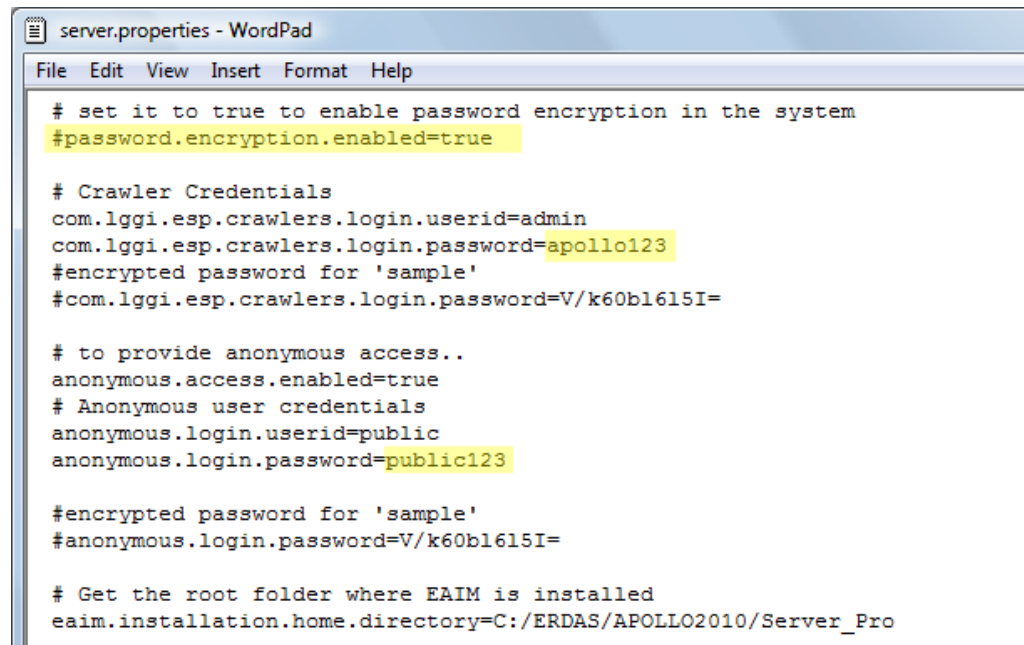
4. The tool will encrypt the password you entered and display it in the command line window.



```
Administrator: Command Prompt
C:\ERDAS\APOLLO2010\Server_Pro>cd tools
C:\ERDAS\APOLLO2010\Server_Pro\tools>cd password-encoder
C:\ERDAS\APOLLO2010\Server_Pro\tools\password-encoder>encrypter.bat
Enter the clear text password: myapolloadminpwd
pkZcemN3NqgL3NwQqc5zmYIH610+GaX9
C:\ERDAS\APOLLO2010\Server_Pro\tools\password-encoder>
```

5. You will need to use this tool to encrypt your admin password and your public password.
6. Navigate to the directory <APOLLO_HOME>/config/erdas-apollo.
7. Right-click on the file `global-server.properties` and select **Open With > Wordpad** in the menu that appears.
8. Find the property ***password.encryption.enabled***. It is located at the very top of the file.

Uncomment that property by removing the # in front of it.



```
server.properties - WordPad
File Edit View Insert Format Help
# set it to true to enable password encryption in the system
#password.encryption.enabled=true

# Crawler Credentials
com.lggi.esp.crawlers.login.userid=admin
com.lggi.esp.crawlers.login.password=apollo123
#encrypted password for 'sample'
#com.lggi.esp.crawlers.login.password=V/k60b1615I=

# to provide anonymous access..
anonymous.access.enabled=true
# Anonymous user credentials
anonymous.login.userid=public
anonymous.login.password=public123

#encrypted password for 'sample'
#anonymous.login.password=V/k60b1615I=

# Get the root folder where EAIM is installed
eaim.installation.home.directory=C:/ERDAS/APOLLO2010/Server_Pro
```


9. Find the property **com.lggi.esp.crawlers.login.password**. Change the value for that property from the clear text admin password to the encrypted admin password you obtained when you ran the password encoder tool.
10. Find the property **anonymous.login.password**. Change the value for that property from the clear text public password to the encrypted public password you obtained when you ran the password encoder tool.

NOTE: Although the public password is only used internally by the ERDAS APOLLO system to grant guest access to unauthenticated users and cannot provide access to sensitive information, you must encrypt it as well when you set the `password.encryption.enabled` property to true.

11. After you change both the `apollo-login-config.xml` and `global-server.properties` files, you will need to restart the JBoss application server.

Administrative Tools Configuration Files

In the `build.properties` file located in the directory `<APOLLO_HOME>/tools/schema-generator`, the passwords are stored so that if you ever need to use the ant tool to rebuild `erdas-apollo.ear` or `apollo-client.war`, the correct passwords will be included in the newly built files. To hide these passwords, you can simply save a copy of this file to a secure location and delete it from the directory `<APOLLO_HOME>/tools/schema-generator`. Restore it to its normal location if and when you need to use the ant tool to rebuild.

In the `build.properties` file located in the directory `<APOLLO_HOME>/tools/harvester-console`, the APOLLO admin password is stored for the process of harvesting services from an older version of ERDAS APOLLO and placing them in the catalog for ERDAS APOLLO 2010 or higher. After you initially upgrade to ERDAS APOLLO 2010, you probably will not need this file again, but you should still save it to a safe location in case something happens to your catalog and you need to rebuild it. After a copy of the file is saved in the safe location, you can delete it from the directory `<APOLLO_HOME>/tools/harvester-console` and restore it to its normal location if and when you need to rebuild the catalog.

Configuration and Customization

Internationalization

When you install ERDAS APOLLO, the language of the ERDAS APOLLO Web Client defaults to the locale of the server computer from which the ERDAS APOLLO Web Client is served, and if that information is not present, it defaults to American English. The ERDAS APOLLO Web Client includes a drop-down box in the upper right corner that allows the user to explicitly choose the presentation language of the ERDAS APOLLO Web Client from the given choices of English, French, German, Polish, Dutch, Japanese, or Chinese. If you select another presentation language, that language will be used for the duration of the session.

You can configure the ERDAS APOLLO Web Client to open with the language you choose, and you can also add support for additional languages.

In order to perform these customizations, you need to know which files the ERDAS APOLLO Web Client uses to store the language information. All of the language files for the ERDAS APOLLO Web Client are located in the directory

```
<APOLLO_HOME>/webapps/apollo-client/default/  
WEB-INF/classes.
```

The files are:

- `tilapia.properties` - contains the setting for the default language for the ERDAS APOLLO Web Client.
- `apollo-client.properties` - contains a list of all the languages supported by the ERDAS APOLLO Web Client.
- `tilapia-i18n.properties` (English)

OR

```
tilapia-i18n_XX.properties
```

language file for the Toolkit where `XX` = fr (French), de (German), pl (Polish), nl (Dutch), ja (Japanese), or zh (Chinese).

- `apollo-client-i18n.properties`

OR

`apollo-client-i18n_XX.properties`

Language file for the ERDAS APOLLO Web Client where `XX` = fr (French), de (German), pl (Polish), nl (Dutch), ja (Japanese), or zh (Chinese).

NOTE: "i18n" is an abbreviation for Internationalization denoted by the 18 letters between the first "I" and the last "n".

Change the Default Language

ERDAS APOLLO reads a setting in a file to determine which files it should read to obtain the labels for the controls. You can change this setting and direct ERDAS APOLLO to use a different set of language files. If you want to use a language other than those listed above, create the language files before you change this setting.

1. Navigate to the directory

```
<APOLLO_HOME>/webapps/apollo-client/default/  
WEB-INF/classes.
```

2. Open the file `tilapia.properties`.
3. Find the two lines shown below.

```
# # Set a default locale for the application  
# locale.default=
```

4. Enter the desired locale code after `locale.default=`.
5. Uncomment (activate) the ***locale.default*** property by removing the `#` in front of the property.
6. Save and close `tilapia.properties`.
7. Restart the application server.

Additional Languages

You can customize your ERDAS APOLLO Web Client to support any language whose characters can be represented in Unicode.

If you are using the ERDAS APOLLO Web Client, create new `tilapia-i18n` and `apollo-client-i18n` language files for your language and add this new language to the list of supported languages in the `apollo-client.properties` file.

If you are only using the Web Toolkit create a new tilapia language file. You will not need to create a new apollo-client language file, because it only contains labels for the ERDAS APOLLO Web Client. You also will not need to add this new language to the list of supported languages in `apollo-client.properties`.

After you create and edit the necessary files, you will need to rebuild the `apollo-client.war` file and redeploy it to your application server.

Create new language files

1. Navigate to the directory

`<APOLLO_HOME>/webapps/apollo-client/default/
WEB-INF/classes.`

2. Create an empty `tilapia_i18n_xx.properties` file (and a `apollo-client-i18n_XX.properties` file if you are translating the web client), where `xx` is a two-letter code that represents the name of the language. Look up the ISO 3166 codes for the translated language at

`http://www.iso.org/iso/english_country_names_and_code_`
`elements.`

3. Access `http://<APOLLO_HOME>/apollo-client/tools/i18n.jsp` and the I18n Helper Tool opens. The tool shows the strings that need a translation and the strings that are already written in the `i18n.properties` file for all loaded locales.

Available locales

English, French, German, Polish, Dutch, Japanese, Chinese

Missing translations

English	Dutch
-	<input type="text"/>
- The clip geometry for entry '%0\$' is not valid in its output SRS	<input type="text"/>
- The coverage '%0\$' and its clip geometry do not overlap	<input type="text"/>
- The size of the request (%0\$ MB) exceeds the maximum request size (%1\$ MB)	<input type="text"/>
- Unknown error for entry '%0\$': %1\$	<input type="text"/>
Bold	<input type="text"/>
Bottom Center	<input type="text"/>
Bottom Left	<input type="text"/>
Bottom Right	<input type="text"/>
Do you wish to continue anyway?	<input type="text"/>
Download Original File	<input type="text"/>
Download the original file	<input type="text"/>

4. Enter each translated text in the box to the right of the English version of the text. Duplicate any formatting place holders such as %s or %d. These place holders must remain intact to hold strings and numeric values.
5. When complete, click Save at the bottom of the page. The tool downloads your changes to a zip file with one or both of the following entries (depending on what strings you translated).
 - tilapia-i18n.changes (for the Web Toolkit)
 - apollo-client-i18n.changes (for the ERDAS APOLLO Web Client)
6. Copy the *.changes file(s) to your tilapia-i18n-xx.properties or apollo-client-i18n-xx.properties files accordingly.

Add to the list of supported languages

1. Navigate to the directory
<APOLLO_HOME>/webapps/apollo-client/default/
WEB-INF/classes.
2. Open the file apollo-client.properties.

3. Find the lines shown below.

```
# Locales
locale.info=EN|English
locale.info=FR|Fran\u00e7ais
locale.info=DE|Deutsch
```

Add another line with the **locale.info** property, and set the value of the property using the following format:

<LANGUAGE_CODE>|<LANGUAGE_NAME>

where <LANGUAGE_CODE> matches with the code you used to represent the language within the file name and <LANGUAGE_NAME> is the name of the language that appears in the dropdown box on the ERDAS APOLLO Web Client that will allow the user to select the presentation language.

Note that the name of the language must be expressed using only ASCII characters and Unicode escape (\uXXXX) characters.

If you added Spanish to the list, the list would look like the following:

```
# Locales
locale.info=EN|English
locale.info=FR|Fran\u00e7ais
locale.info=DE|Deutsch
locale.info=ES|Espa\u00f1ol
```

Rebuild/Redeploy the apollo-client.war file

1. Create an ANT_HOME system variable with the path to the directory <APOLLO_HOME>/tools/ant/bin.
2. After you have created the system variable, open a command line window and type:

```
cd <APOLLO_HOME>/tools/ant/bin <press ENTER>
ant tomcat55 <press ENTER>
```

The argument of the "ant" call should indicate the name of the application server that you are using.

You can open and read the `build.xml` file located in the <APOLLO_HOME> directory to obtain the correct argument for your application server.

ERDAS APOLLO Web Client Configuration

3. Wait until the build is successful, then go to the directory `<APOLLO_HOME>/dist/<APPSERVER_NAME>` and copy the file `apollo-client.war`.
4. Redeploy the new `apollo-client.war` file for your application server.

For JBoss

Paste into the directory

`<APOLLO_HOME>/jboss/server/default/deploy`

For Tomcat (5.5 and 6)

Paste into the directory `<APOLLO_HOME>/tomcat/webapps`

5. Restart the application server.

The files that make up the ERDAS APOLLO Web Client are placed together in the directory `<APOLLO_HOME>/webapps/apollo-client` when you install the ERDAS APOLLO Server.

Many customers like to customize their web clients. To do this, you will need to open the `<APOLLO_HOME>/webapps/apollo-client` directory, find the file or files that contain the properties you want to change, and make the required changes. After you have changed all of the files, you will need to run an Apache Ant script that will compress that directory into a **Web AR**chive (WAR) file. That war file is deployed to the application server that you are using for the ERDAS APOLLO Server.



If you need to use the ERDAS TITAN client for WPS execution, do not remove the APOLLO-CLIENT.WAR from the ERDAS APOLLO Application Server.

Properties Files

The ERDAS APOLLO Web Client uses a number of .properties files to store configuration parameters.



Anything you type in these properties files may potentially be viewable by anyone who can view the ERDAS APOLLO Web Client online. Do not put any information you want to keep confidential inside these files!

Default Hierarchy

The `root.properties` file is in `WEB-INF/classes/tilapia.properties` (this is hard-coded). All properties files need to be in the classpath (such as in `WEB-INF/classes`).

- `tilapia.properties`
 - `apollo-client.properties`
 - `apollo-client-contexts.properties` (specified in a `tryimport` in `apollo-client.properties`)
 - `apollo-im-components.properties` Or `apollo-server-components.properties` (specified in a `tryimport` in `apollo-client.properties`)

Entries

Name	Description	Note
<code>tryimport</code>	look for the specified file in the classpath and if it is there, import it as an additional properties file	
<code>feature.panel.config</code>	feature panel configuration as described in the <i>ERDAS APOLLO Solutions Toolkit Main Guide</i> , Section 3.6.5.2.4	
<code>locale.default</code>	Default locale - see Internationalization on page 199	
<code>i18n.files</code>	Location for internationalization files - see Internationalization on page 199	
<code>layerinfohandlers.wfshandler.maxfeatures</code>	Maximum number of features to display in the Layer Info tool	

Name	Description	Note
metadata.tc211.stylesheet		
metadata.stylesheet.iso19139	The path to the XSL file for displaying ISO19139 metadata	
editors.timestampinterval.editor.dat eoffset	The default date offset (in days) between the start and end date	
context.startup	Context files that appear in the Context list	
context.overview.default	Path to the default context file	
jsonhelper.ident	Indentation of the JSON output in the logs	
log.type	Type of log (such as FILE, etc.)	
log.enable	Enable the log	
log.filename	Path to log file, such as /temp/mylog	
log.maxfile	Number of log files to create (the logger will create new log files as needed, cycling from mylog0 to mylogmaxfile -1)	
log.filesize	Maximum size of log file in bytes before a new log file is created	
service.ias.url	IAS URL	see eaim.server.wcs.url
service.catalog.url	Catalog URL	see eaim.server.catalog.url
service.wrs.url	WRS URL	see eaim.server.wrs.url
service.wps.url	WPS URL	see eaim.server.wps.url
service.clipzipship.url	Clip/Zip/Ship service URL (see eaim.server.clipzipship.url)	
service.quartzinterface.url	Quartz JSON Interface URL (see eaim.server.quartzinterface.url)	
service.wrs.type	WRS type: Babel or RSCatalog	
service.streamedraaster.access.enabled	True: streaming (ECWP) access enabled	see eaim.server.streamedraaster.access.enabled
service.streamedraaster.access.url	URL for streaming (ECWP) access	see eaim.server.streamedraaster.access.url
service.ias.iso19115Xslt	Path to the XSL file for displaying ISO19115 metadata (can be undefined)	

Name	Description	Note
service.ias.queryables	Path to the XML file containing Queryables information	
modules.search.layers.assumeERDAS	If set to True, assume modules.search.layers references a WFS from ERDAS (allows optimizations)	
modules.search.layers.useWms	If set to True, if the modules.search.layers WFS server exposes a WMS interface it be used for rendering	
thumbnail.width	Width of thumbnails in pixels	
thumbnail.height	Height of thumbnails in pixels	
thumbnail.create	If set to True, automatically create new thumbnails	
mail.smtp.host	SMTP host for Clip/Zip/Ship messages	see eaim.server.smtp.host
mail.smtp.port	SMTP host for Clip/Zip/Ship messages	see eaim.server.smtp.port
mail.smtp.user	SMTP host for Clip/Zip/Ship messages	see eaim.server.smtp.user
ui.objectinspector.defaultresultsperpage	Default number of results to display in object inspectors	
modules.search.max.nb.results	Maximum number of search results (-1 for unlimited)	
modules.search.thumbnail.popup	if set to True, display thumbnails in the popup when hovering over a result	
modules.search.thumbnail.details	if set to True, display thumbnails in the extended details panel	
scripts.bundles.desc	path to the file containing the description of custom modules	
wms.imageformat	sets default file format for WMS requests	valid: image/gif, image/png, image/jpeg

Components

Components are discrete parts of the web client application that can be added or removed independently. The installer provides two files, `apollo-im-components.properties` for Professional and `apollo-server-components.properties` for Essentials-SDI. It may be desirable to remove components by deleting or commenting out their entries. Alternatively, a different properties file containing components information can be provided by providing a different `tryimport` entry in `apollo-client.properties`.

Entry	Usage
<code>components.active</code>	each entry will be added to the application on startup
<code>components.search.typesfilter</code>	the entries that appear in the "types" dropdown list in the Search panel
<code>components.search.defaulttype</code>	the "types" entry that is selected by default in the Search panel

Available components:

- **Browse:** The Browse tab, which allows the user to:
 - see all of the known services in tree form
 - add new services
 - create new transient services by uploading data from the local file system
- **Edit:** The Edit tab for viewing and editing features stored in a WFS
- **Filter:** The Filter tab for viewing and modifying the filter on a WFS layer
- **Search:** The Search tab allowing the user to search the catalog for available resources
- **OverviewMap:** The Overview Map displaying a large scale overview of the user's current map view
- **WPS:** The Process tab for managing WPS processes
- **DownloadImagery:** The Download tab for managing images for the Clip/Zip/Ship operation

Contexts

The ERDAS APOLLO Web Client provides a tool that allows users to pick from a list of predefined context files. By convention this list is populated from entries in `apollo-client-contexts.properties`. An example is provided in `apollo-client-contexts-samples.properties`. The first entry will be loaded at application startup by default. The format for these entries is pipe (|) delimited as follows:

- **context.startup**
<path>|<title>|<documentation-page>|<overview-map-path>
- **path**
the path to the context file, such as `/context/default-basemap.xml`
- **title**
the title that will appear in the dropdown list, such as "Default Basemap"
- **documentation page**
the path to an HTML page providing additional details of the context file (can be blank)
- **overview map path**
the path to the context file that will be used in the overview map. If blank, defaults to **ic.context.path**

The ERDAS APOLLO Style Editor

ERDAS APOLLO Style Editor is a Java Swing client that can be used to both explore and style geographical data. The ERDAS APOLLO Style Editor can access OpenGIS services such as Web Map Servers, Web Feature Servers, Web Coverage Servers. The ERDAS APOLLO Style Editor also helps in the creation of styles which are used to render maps by the Portrayal Service.

Exploring Data

This section gives a first introduction of the ERDAS APOLLO Style Editor user interface. You will learn how to manipulate data sources, apply them to your project and navigate through the data.

Getting started

Starting the ERDAS APOLLO Style Editor

Once installed, you can start ERDAS APOLLO Style Editor in the following way:

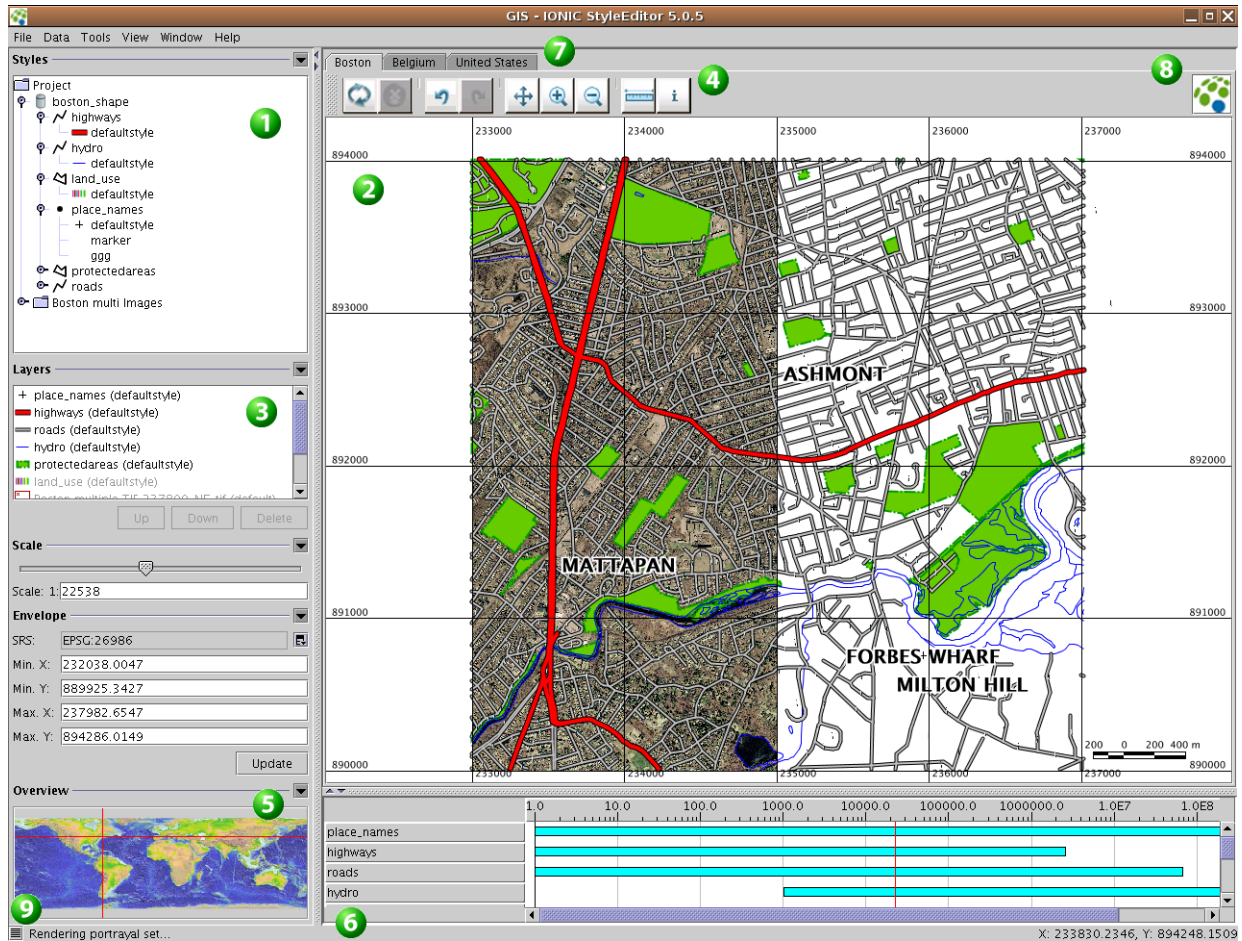
- On Windows platforms: Double-click the ERDAS APOLLO Style Editor icon located on your desktop (if you choose to create a desktop shortcut during the installation) or use the *Start* menu.
- On UNIX platforms: cd to the directory where you installed ERDAS APOLLO Style Editor and type **`./styleeditor.sh`**.

After the splash screen, you should see the ERDAS APOLLO Style Editor main window described in the next section.


The ERDAS APOLLO Style Editor Main Window Described

The following picture presents the tool's main window. Note that the data presented here varies according to your copy of ERDAS APOLLO Style Editor and how you obtained it.

Figure 19: ERDAS APOLLO Style Editor Main Window



- **1** presents the current project structure as a tree. This panel shows all the data sources you added to the project: WFS, SHAPE files, WMS, georeferenced images, WCS, ... Please refer to [Data Sources](#) for more information.
- **2** is the map panel. Unless you specified a particular device screen (see [Views](#) for more information about device screens), the map panel will be resized when the main window is resized.
- **3** shows the list of layers that have been added to the preview together with buttons to reorder the list and remove layers from the preview. Please refer to the [Layers](#) for more information.
- **4** is the undockable ERDAS APOLLO Style Editor toolbar that exposes several icons whose functions are described in the [Map Navigation](#).

- **5** is a collapsible overview area that shows the currently displayed box on the whole world as a yellow area or as a red cross. Refer to [Map Overview](#) for more information.
- **6** shows a status bar with information such as cursor coordinates or measured lengths and areas. It contains a split pane which allows you to reveal the *Scale Range Manager* described in [Scale Range Management](#).
- **7** presents the different views on the project as a range of tabs, as described in [Views](#).
- **8** shows the ERDAS APOLLO Server logo which indicates the tool activity. The logo gets animated while ERDAS APOLLO Style Editor is performing an operation. Note that the current operation can be canceled at any time by pressing the Stop button .
- **9** in the left corner of the status bar, this button gives you access to the Status History where you can consult the previous performed tasks.

Configuration

This section explains how to personalize some of the options ERDAS APOLLO Style Editor uses to determine its behavior. The options are placed in the Style Editor Preferences window, accessible in the tools menu.

Figure 20: Preferences item in the Tools menu

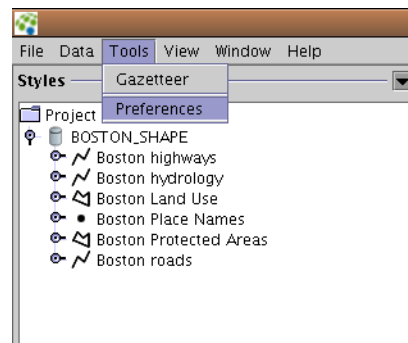
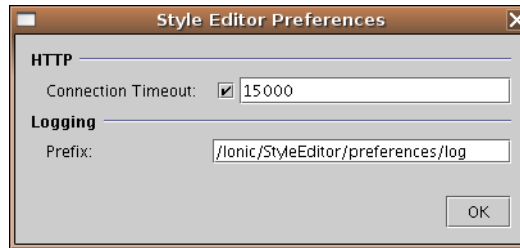


Figure 21: Preferences Window



HTTP Connection Timeout

This option determines the maximum time ERDAS APOLLO Style Editor waits for a server response. Selecting a time limit is useful to prevent situations where ERDAS APOLLO Style Editor would be kept waiting forever for a non-responsive server. The initial value is already set to produce reasonable behavior with the majority of the situations. Expert users may want to adjust the value, and even deactivate it, to meet more specific situations.

Procedure Setting the Connection Time-out

1. Select the Preferences option from the Tools menu.
2. In the HTTP group, use the checkbox to activate or deactivate the Connection Time-out.
3. Set the text field with the desired value in milliseconds.



This initial value is set to 15000 milliseconds.

Logging

ERDAS APOLLO Style Editor automatically creates log files you can consult, for instance to view the generated map requests. By default, the files have a prefix `log`, and are placed in the `preferences` folder.

Procedure Setting the Log Path

1. Select the Preferences item from the Tools menu.
2. In the Prefix text field, set the path and prefix for the log files.

Managing Projects

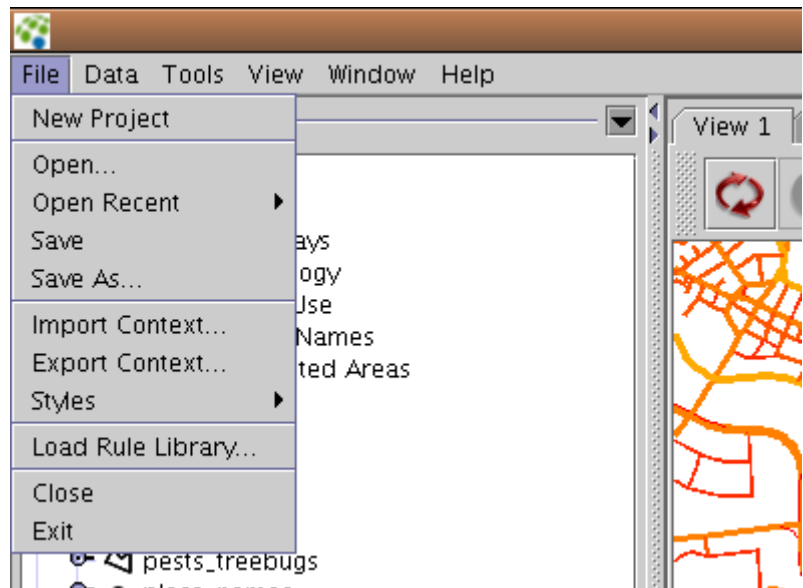
ERDAS APOLLO Style Editor stores its configuration in a centralized file called a project file. The project file contains information such as:

- a list of different views, each representing a map
- a list of data sources
- a list of layers
- style configuration
- miscellaneous settings

Project files are associated with the `.gar` extension.

Project management (new, open, ...) is carried-out throughout the File menu.

Figure 22: The File Menu



Creating a New Project

To create a new project, select New Project in the File menu. A new, empty, project will be created in a new window.

When you start ERDAS APOLLO Style Editor (except for the very first time), a new blank project is opened for you to work with.

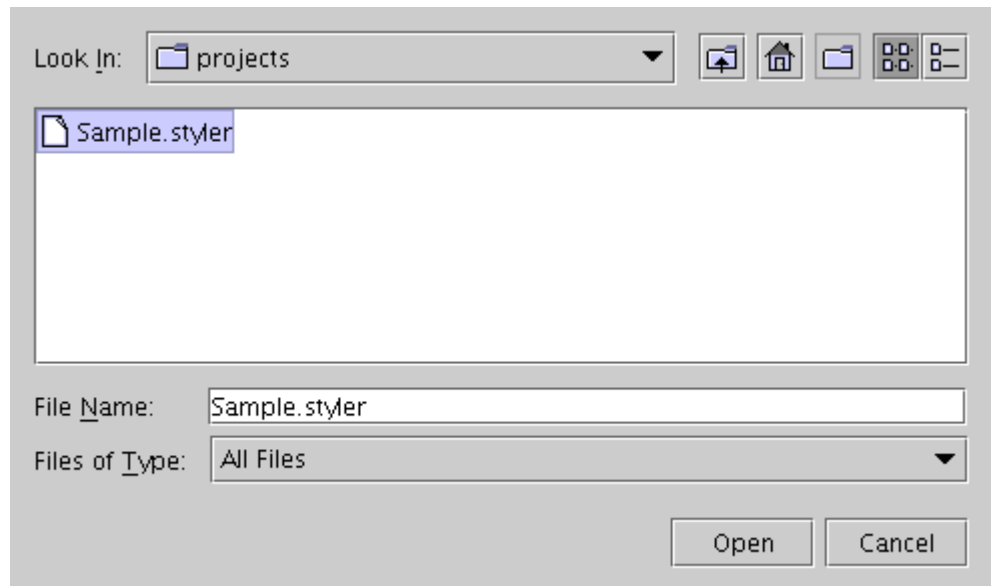


To use your new project, you may want to start adding data sources. Read the next sections of this guide to learn how to add data sources.

Opening an Existing Project

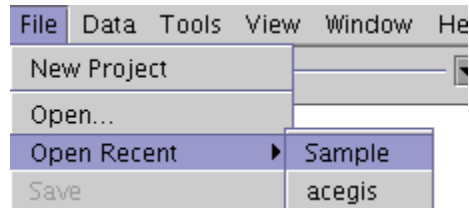
To open an existing project, select Open... from the File menu. When asked to choose a file, select a project file with an extension of .gar or .styler. Typically, the projects are stored in the "projects" subfolder of your ERDAS APOLLO Style Editor installation.

Figure 23: Open Project



You can also open a project using the "Open Recent >" menu item, which allows a direct access to the 10 most recently opened projects.

Figure 24: Open Recent Project



Data Sources

Kinds of Data Sources

A data source is an OpenGIS/ISO compliant service or a GIS resource such as a local Shapefiles directory. Data sources are used by the ERDAS APOLLO Style Editor to render maps, but also to query them about their capabilities, the list of layers from a WMS, feature descriptions from a WFS, etc.

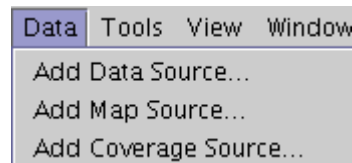
The current version of ERDAS APOLLO Style Editor supports the following data sources.

- Remote WFS through the HTTP protocol
- Local Feature Server, from a `.fac` file
- Local Shapefiles directory
- Remote WMS through HTTP
- Local georeferenced image
- Remote WCS with remote CPS
- OpenGIS WMS Contexts

Adding a Data Source

Adding a data source, whatever its type, is accomplished by using the **Data** menu shown below:

Figure 25: Data Menu



The next sections present the detailed procedure for each data source type.

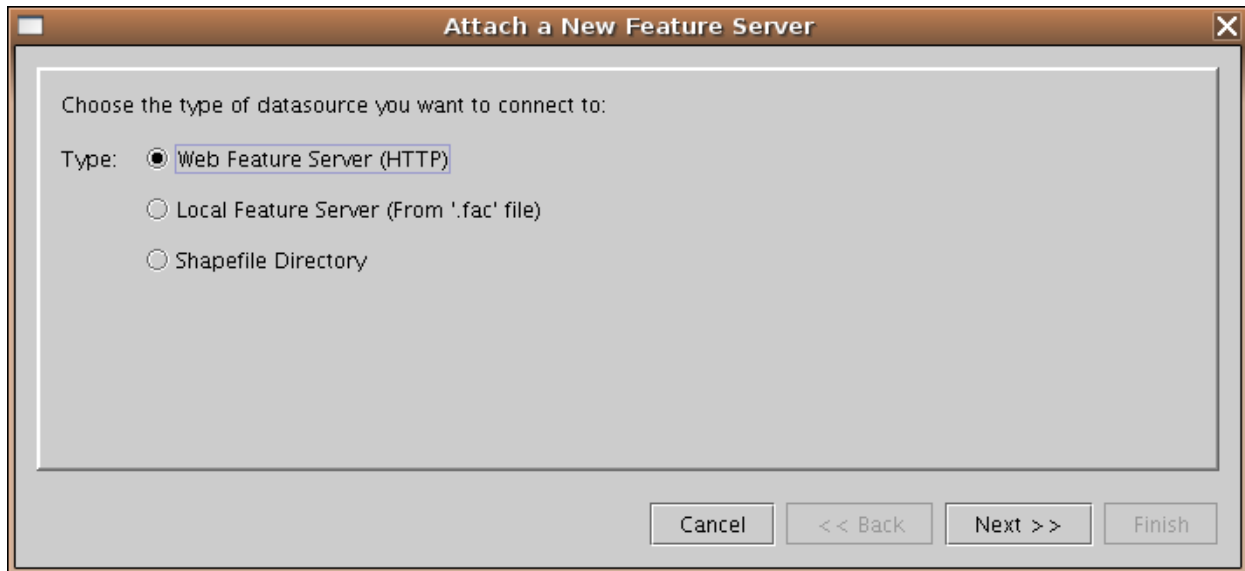
Adding Features Resources

This section explains how to add remote OpenGIS/ISO Web Feature Servers and Shapefiles directory to the project.

Procedure Adding Features Resources

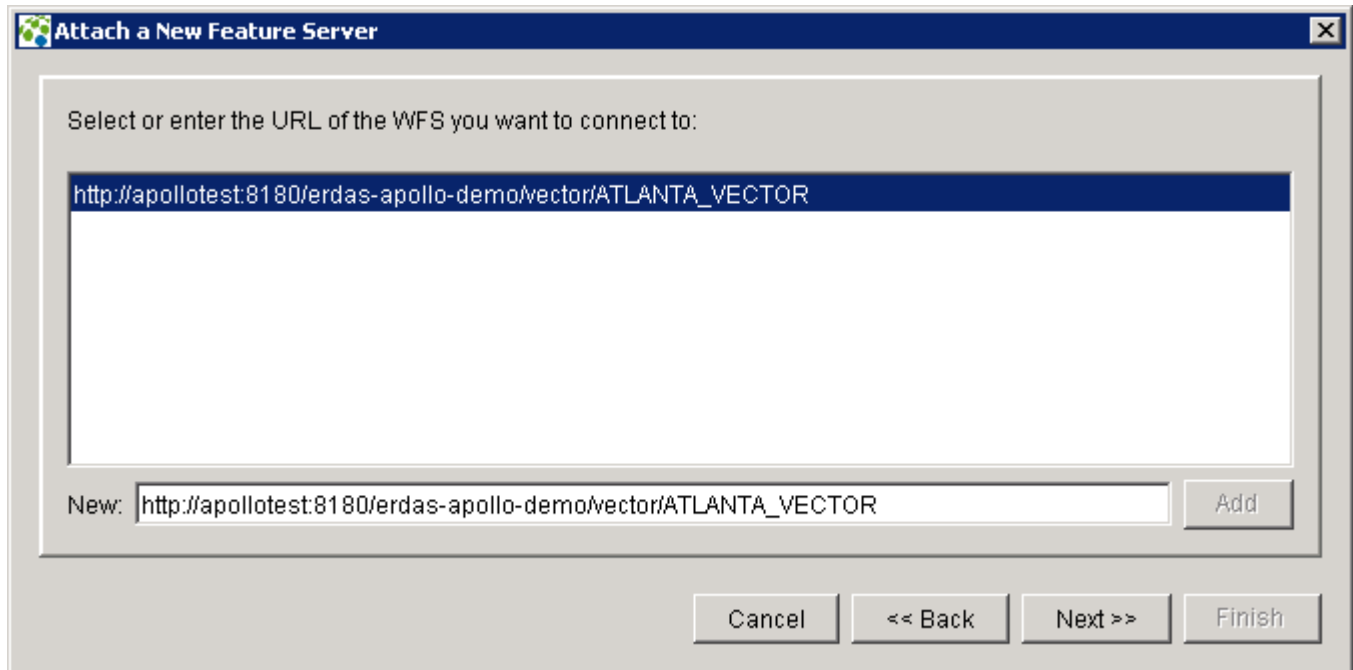
1. Choose **Data/Add Data Source...** in the menu then choose either to add a WFS, a Local Feature Server or a collection of Shapefiles located in a given directory.

Figure 26: Add Data Source



2. Select the type of feature service to access:
 - Adding a Web Feature Server
 - To add a reference to a remote Web Feature Server, select **Web Feature Server (HTTP)**:
 - In the next panel, enter the service URL and click the **Add** button, or choose a previously entered URL from the list. Note that all valid URLs are automatically collected for future usage.

Figure 27: Attach a Newap Feature Server - Step 2



- You then have to provide both a *Name* and a *Title* for the service. Name is used in styles in order to achieve the mapping between a service and a rule bundle while title is the human-readable title that will be displayed in the Project Structure panel.



The Name must be in lowercase and must correspond to the name of the provider specified in the configuration of the WFS. Please refer to [Servlet-Specific Configuration Parameters \(providers fac\)](#).

Figure 28: Attach a New Feature Server - Step 2

Attach a New Feature Server

Enter an unique name for the new datasource (should correspond to the lowercased WFS provider name, if any):

Name: atlanta_vector2

Title: ATLANTA_VECTOR

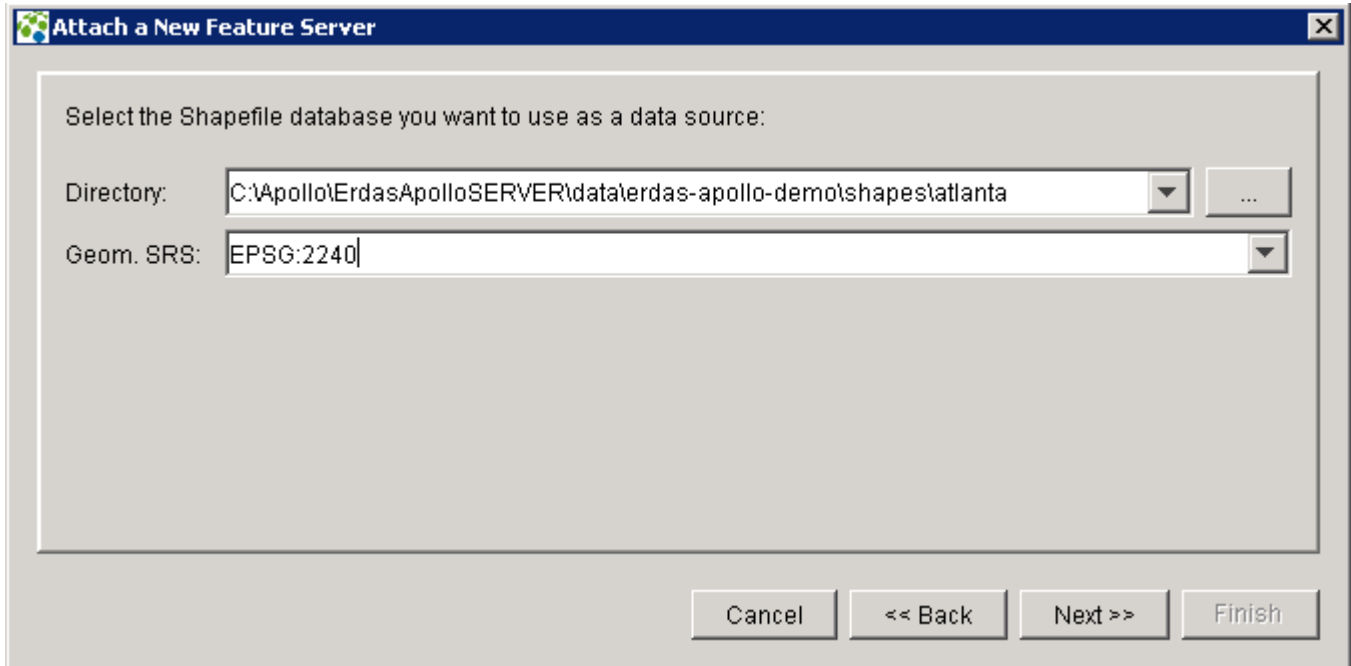
Cancel << Back Next >> Finish

- Click the **Finish** button.
- Adding a Local Feature Server
 - Choose **Local Feature Server**.
 - You may directly insert the path to the intended feature, select the " ... " button to browse your files or choose one feature from the data source history list.
 - Select **Finish**.
- Adding Shapefiles
 - To add a collection of Shapefiles, choose **Shapefile Directory**.
 - First choose a directory that contains Shapefiles (common extensions are `.shp`, `.shx`, `.dbf`).
 - Since Shapefiles do not export any SRS information, you have to manually specify the native SRS in which your data is expressed.



The drop-down list of SRS is an history of your previous selections, not a list of suggested SRS for the given Shapefiles directory.

Figure 29: Add Shapefiles



- In the next panel, enter a *Name* for your data source.



The Name must be in lowercase and must correspond to the name of the provider that will be used when configuring your Portrayal Service.

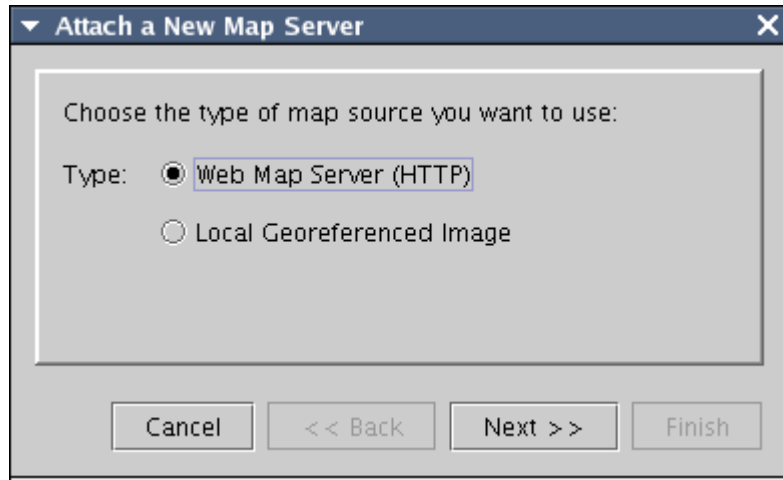
Adding Raster Resources

This section explains how to add remote OpenGIS/ISO Web Map Servers and local georeferenced images to the project.

Procedure Adding Raster Resources

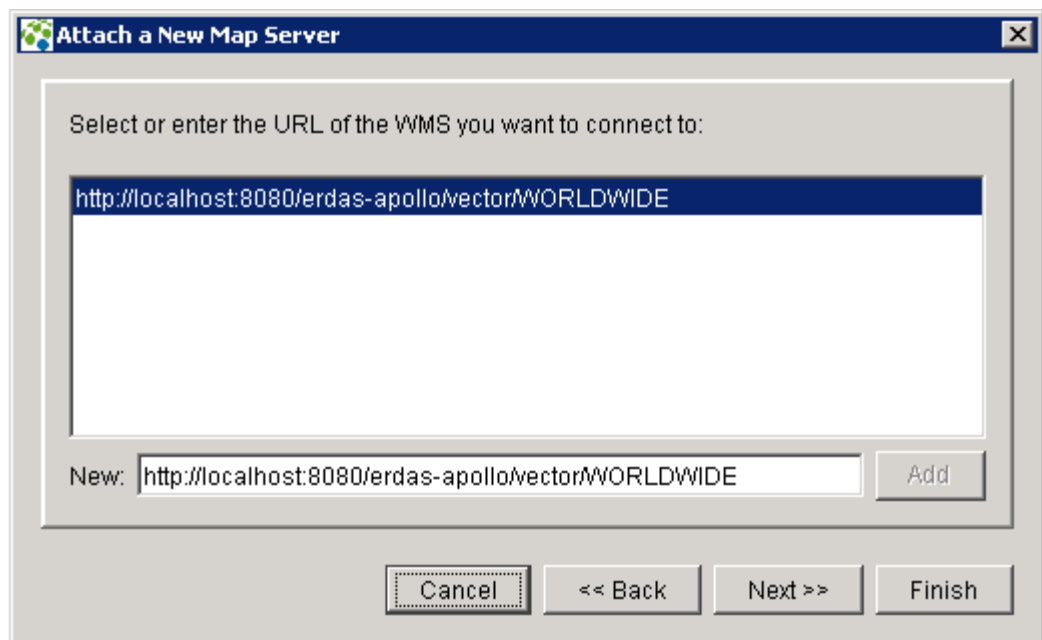
1. Choose **Data/Add Map Source...** in the menu then choose either to add a WMS or a georeferenced image located in a given directory.

Figure 30: Add Map Server



2. Adding a Web Map Server - To add a reference to a remote Web Map Server, select the corresponding radio button then follow these steps:
 - In the next panel, enter the service URL and click the **Add** button, or choose a previously entered URL from the list. Note that all valid URLs are automatically collected for future usage.

Figure 31: Attach a New Map Server



- Click the **Finish** button.
3. Adding a Local Georeferenced Image - To add a georeferenced image, choose the corresponding radio button then fill in the following panel:



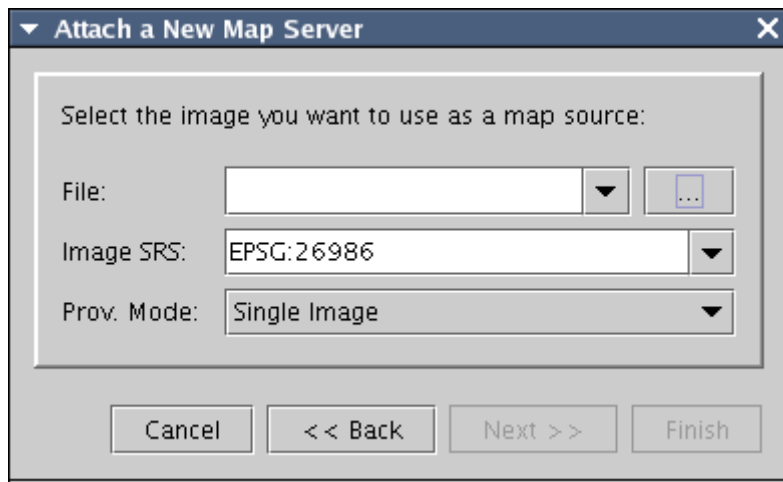
Georeferenced image are not being saved in the project file in this version of ERDAS APOLLO Style Editor. You will need to manually add them on every new launch of the program.

- First select a georeferenced image from your hard disk.
- Manually specify the native SRS of the image, even if your georeferenced image already defines an SRS.



The drop-down list of SRS is an history of your previous selections, not a list of suggested SRS for the given georeferenced image.

Figure 32: Attach a Georeferenced Image



- Select a provider mode. Please refer the ERDAS APOLLO Server Administrator's Guide for more details on provider modes.
- Click the **Finish** button.

Adding Coverage Resources

This section explains how to add remote OpenGIS/ISO Web Coverage Servers and their associated Coverage Portrayal Services to the project.

A Coverage Portrayal Service is a special kind of WMS that understands SLD stylesheets containing raster coverage rendering operations.

Procedure Adding Coverage Resources

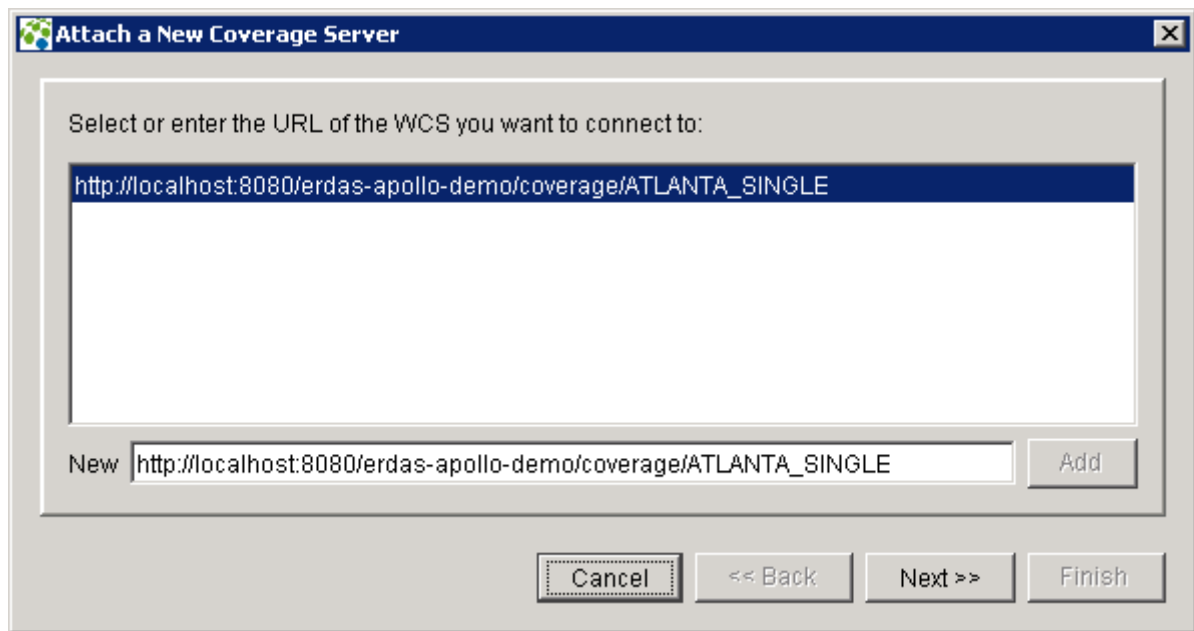
1. Choose **Data/Add Coverage Source...** from the Data menu.



The coverage service doesn't currently support on-the-fly coordinate transform. For example, to use the ATLANTA_SINGLE coverage service, you will first need to change the coordinate system to EPSG:2240, which is the SRS supported by ATLANTA_SINGLE.

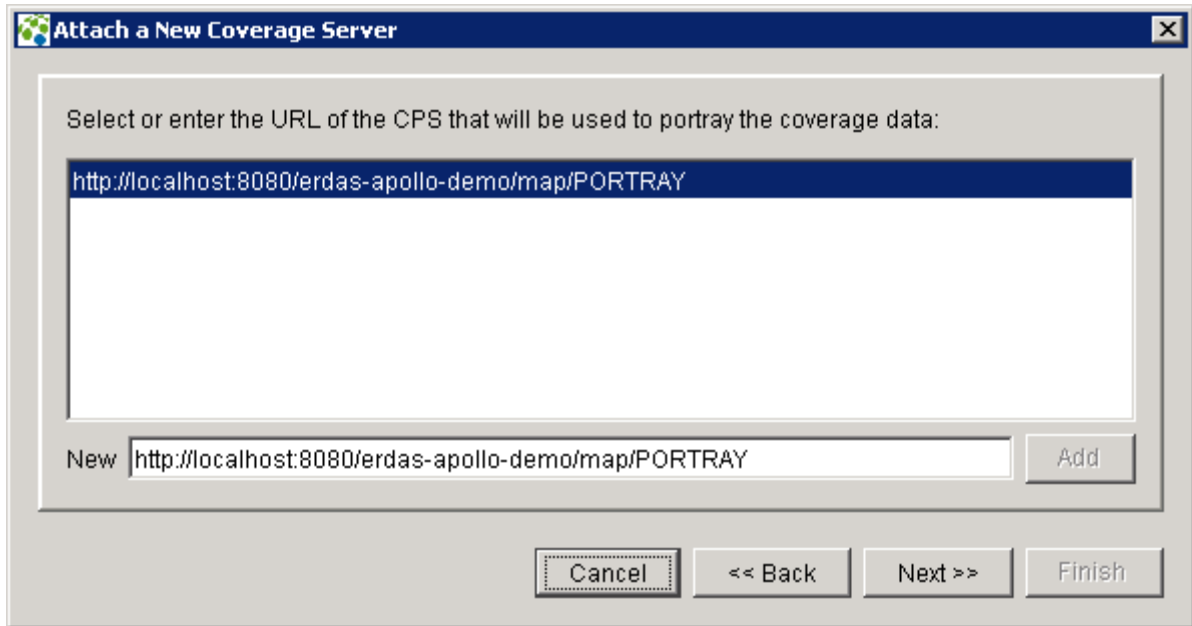
2. Enter the URL of a Web Coverage Service and click **Add**.

Figure 33: Coverage Source



3. Enter the URL of a Coverage Portrayal Service, click **Add**.

Figure 34: Portrayal Service URL



4. Enter a unique **Name** and a **Title** for the coverage. The name must be in lowercase and should preferably correspond with the WCS provider.
5. Select **Finish**.

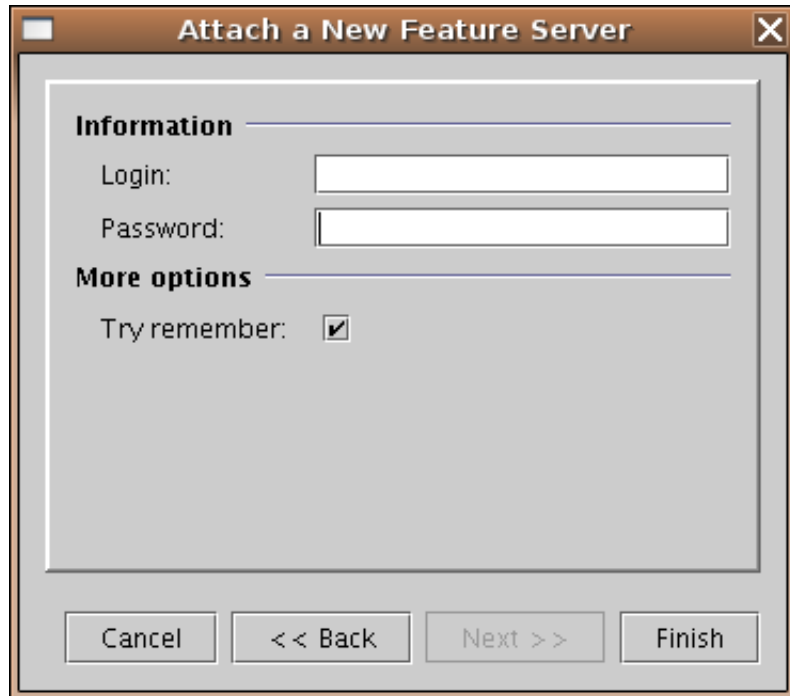
Authenticated Connections

Some data services (WFS, WMS or WCS) may require authentication in order to establish connection. The authentication is done through a login and password which can be defined when adding the data source.

Procedure Establishing an Authenticated Connection

1. Follow the normal Add Data Source procedure for the specific type you wish to add, as described from [Adding Features Resources](#) to [Adding Coverage Resources](#).
2. In the final step of the procedure, select **Next** (instead of **Finish**).
3. Enter the server's **Login** and **Password**.

Figure 35: Secure Connection Window



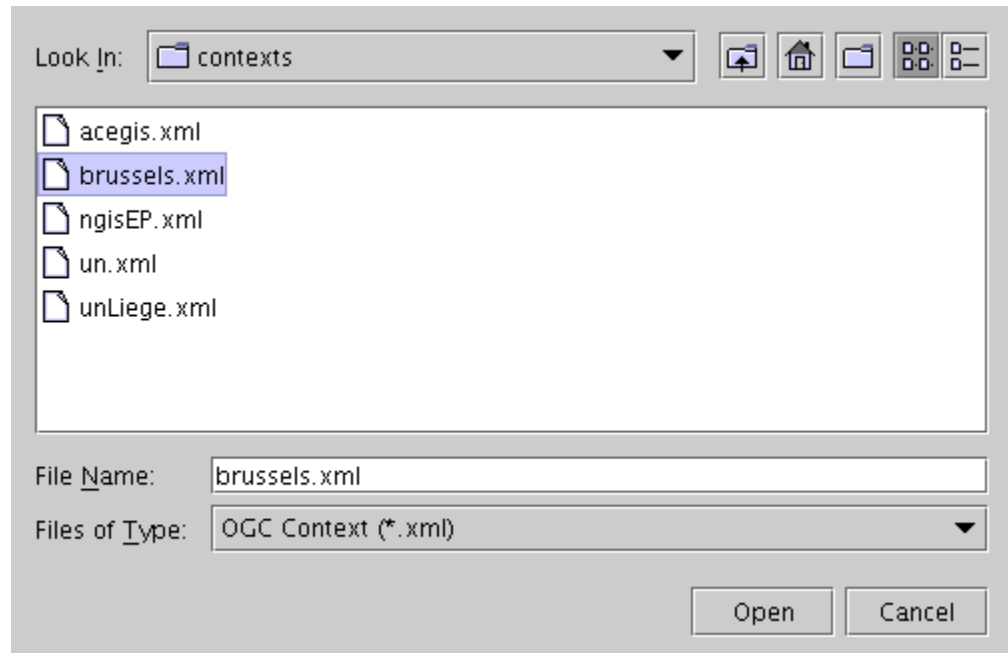
4. With the **Try remember** check box, select if you want ERDAS APOLLO Style Editor to memorize the login and password for future utilization, even if the service URL is used as another type of data source.
5. Select **Finish**.

Adding Resources from a Context

ERDAS APOLLO Style Editor makes intensive use of OpenGIS/ISO Web Map Contexts. Please read the *Concepts Guide, chapter 3* to learn more about contexts or consult the [OGC](#) website for the specification.

Importing a context means to add new data sources to your project from a local XML file. To import a local context into ERDAS APOLLO Style Editor, select the **Import Context...** from the File menu, choose a OpenGIS context XML file you want to import, as shown by the following.

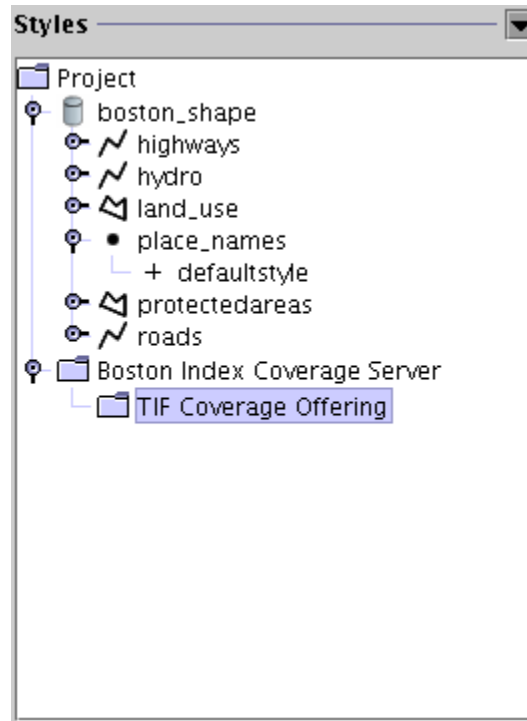
Figure 36: Import Context



Browsing a Data Source

The data source panel presents, in a tree structure, all the resources, or data sources, that have been added to the project. Among the resources, you'll find WMS, WFS, WCS, etc.

Figure 37: Data Source Panel



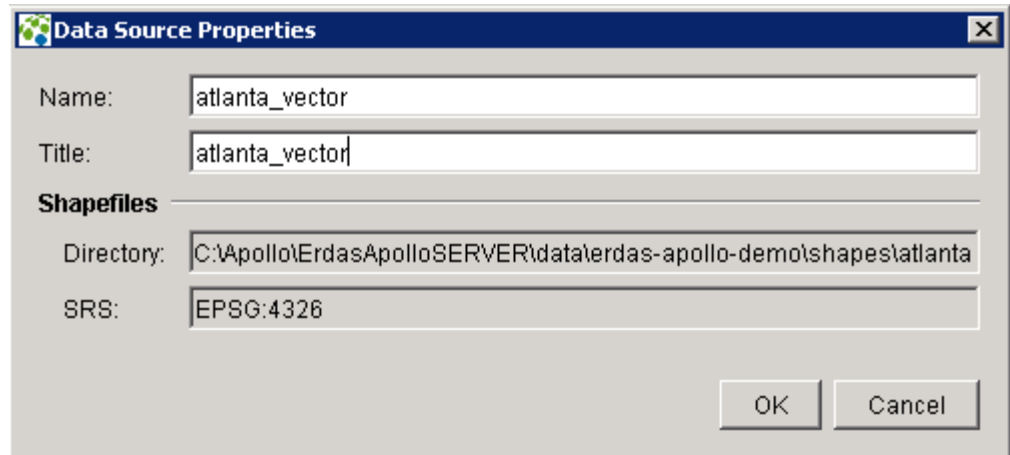
To browse a specific data source, double-click the data source name or click its expand icon. Each data source lists its containing elements. The list of elements is specific to each data source. For instance, a WFS will list feature types while a WMS will list layers.

More information on the data source as well as type specific actions can be obtained in the contextual menu, by right-clicking in the data source label.

Figure 38: Data Source Properties Item



Figure 39: Data Source Properties Window



The Forgiving Checkbox

When trying to add to the preview panel a WFS style based on a non-conforming geometry, if the actual geometry does not match the type defined in the feature schema, an error will be output alerting you of this situation. This is a sign of server misconfiguration that should be fixed in order to maintain the server compliant with the WFS specification.

However, in some circumstances, in order to be able to use the information despite that mismatch, you may want to allow the introduction of non-conforming geometry layers in your project. To do this, access the data source properties menu, by right-clicking it in the Styles panel and selecting **properties**. If the data source type is a WFS, you will see a checkbox named **Forgiving**. If activated, this property makes ERDAS APOLLO Style Editor behave less restrictively and accept the layer.

Removing Data Sources

The data sources you added to your project can be easily removed directly from the Styles panel.

Procedure Removing a Data Source

1. In the Styles panel, right-click on the data source you wish to remove.
2. A contextual menu appears, select **Remove Data Source**.

Figure 40: Remove Data Source Option



3. In the confirmation box, select **Yes**.



This action is irreversible. It will destroy all the styles created from the selected source, even if they are currently being used.

Data Source's History List

Each time you add a valid web service URL it is stored in the data source's history. This history is conveniently shown to you when trying to add data sources, so you don't need to remember or retype the same URL multiple times.

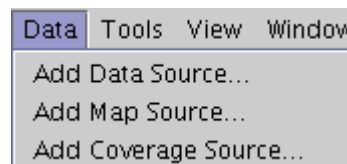
The data source's history is not shown necessarily in a chronological order, since you can manipulate it in order to obtain the most benefit. This includes the possibility of removing the stored URLs you don't want to keep, as the possibility to change their order of appearance.

By selecting the displayed position of the URLs, you can organize them by subject, by keeping the most used URLs on top of the history list, or in any other way you'll find suitable for your needs.

Procedure Manipulating the Data Source's History List

1. Select **Data** from the top menu.
2. Select to **add** one of the following data types: **Data**, **Map** or **Coverage**.

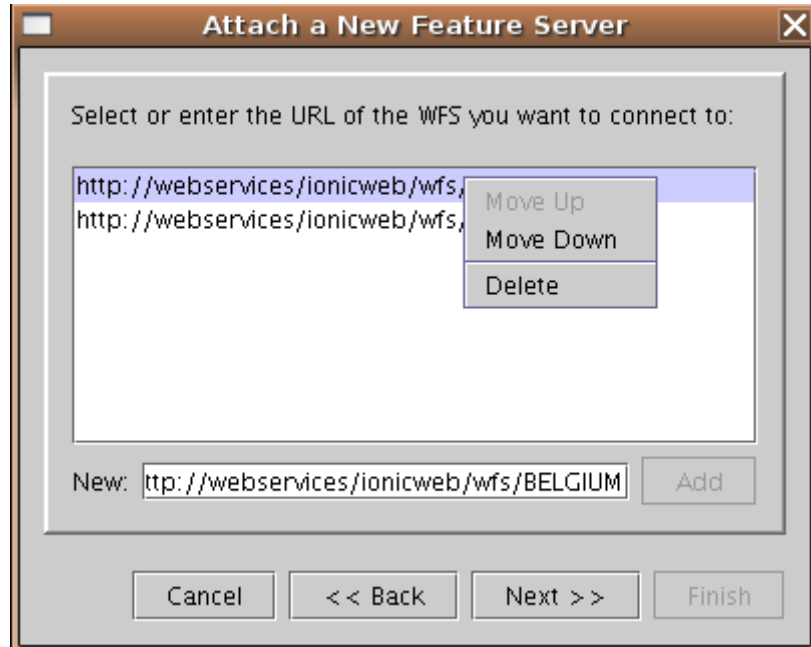
Figure 41: Add Data Menu



3. Select the locality of the data source. (E.g: It can be a local resource or a remote server).

4. A list with the previous inserted resources will appear. Right-click in one of the entries to obtain its contextual menu.
5. From the contextual menu, select the action you want to perform:

Figure 42: History List options



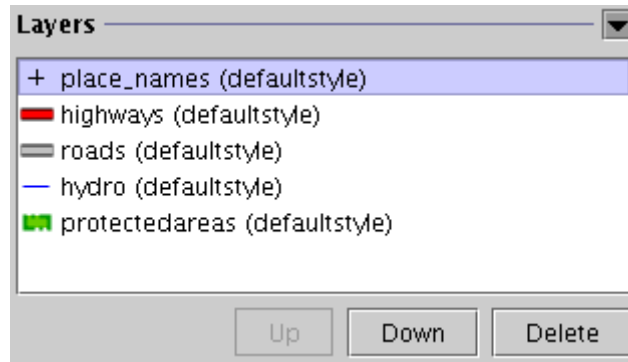
- Select **Move Up** to bring the data source's address up by one position.
- Select **Move Down** to bring the data source's address down by one position.
- Select **Delete** if you want the URL to disappear from the history list.

Layers

The Layers Panel

When rendering a map, the ERDAS APOLLO Style Editor will overlay different layers to form a single image. The list of layers that the ERDAS APOLLO Style Editor considers when composing its map are taken from the layers panel.

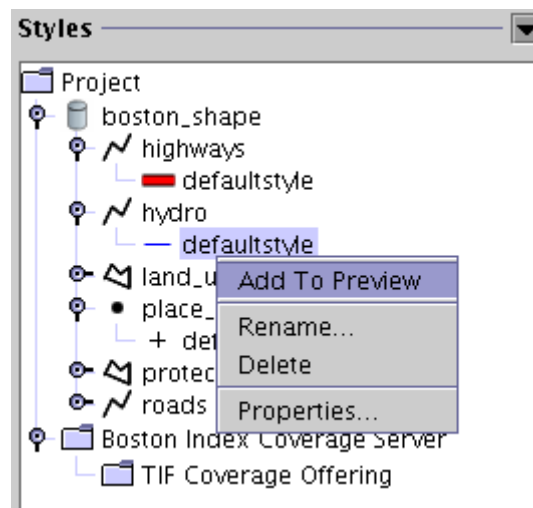
Figure 43: Layers Panel



Adding a Layer to the Map

To add a layer to the map, in the data source panel, right-click on one of the sub-elements of a data source. Select the **Add To Preview** menu entry. The selected layer will then appear in the layers panel and the map will automatically be refreshed.

Figure 44: Add Layer



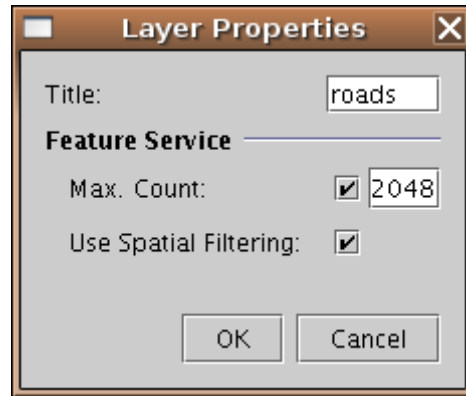
Renaming a Layer

The layers you add to the map are created, by default, with the same name as its corresponding data source element, allowing you to quickly start using the layer without further delay. ERDAS APOLLO Style Editor gives you the possibility to rename the title of each individual layer, thus, allowing you to choose the best suited name in the context of your own project.

Procedure Rename a layer

1. In the layer panel, right-click on the layer you wish to rename. Its contextual menu will appear.
2. In the contextual menu, select **Layer Properties**.

Figure 45: Layer Properties

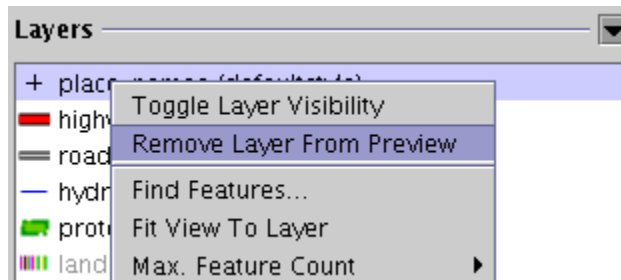


3. Replace the current name in the **Title** field with the new label title.
4. Select **OK**.

Removing a Layer from the Map

To remove a layer from the rendered map, right-click on a layer in the layers panel and select the **Remove Layer from Preview** menu entry. The map will refresh itself and the layer will be removed from the layers panel.

Figure 46: Remove Layer



Hiding a Layer from the Map

To keep a layer in the Layers Panel but force it not to be rendered in the map, right-click on the layer in the Layers Panel and select the **Toggle Layer Visibility** menu entry. The layer will then be hidden from the map and will appear as grayed in the layers panel.

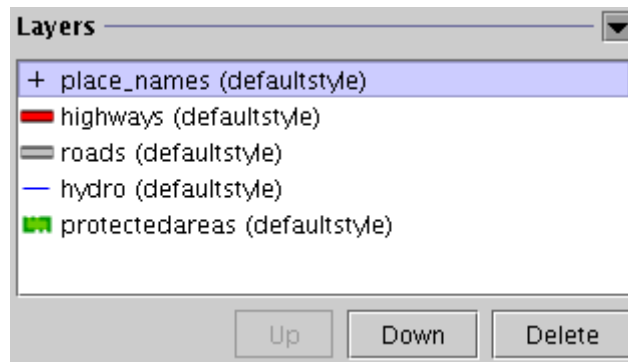
To make the layer visible again, repeat the above operation.

Ordering Layers

The order in which layers compose the map can be changed. To make a layer appear above another layer, click the layer in the Layers Panel and select the **Up** button. The map will automatically reload to reflect this change. The **Down** button can also be used to place a layer below another layer.

The **Up** and **Down** buttons are illustrated in the following figure:

Figure 47: Ordering Layers



The Max Count Option

This option allows you to limit the number of results obtained. This is especially useful in situations where a too generic request retrieves a data amount which is far greater than you anticipated, resulting in huge transfer delays.

As example, imagine a server with detailed information on all of the streets in the United States. You may need only a specific regional part of this information, but if you start with the U.S. map on your screen, before zooming in, a request demanding all the roads will be sent blocking navigation. If a limit is set, only the specified amount will be drawn allowing you to proceed with the navigation or perhaps analyze the retrieved data in order to understand which data can be filtered out.

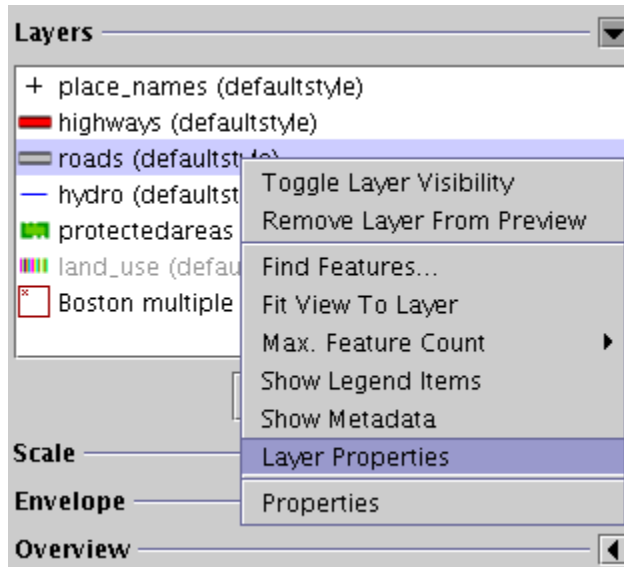
This option only applies to WFS servers. Other services, such as WMS or WCS have their own specific filters.

Procedure Configuring the Max Count property

1. In the Style panel, **right-click in a feature layer**. Its contextual menu will appear.

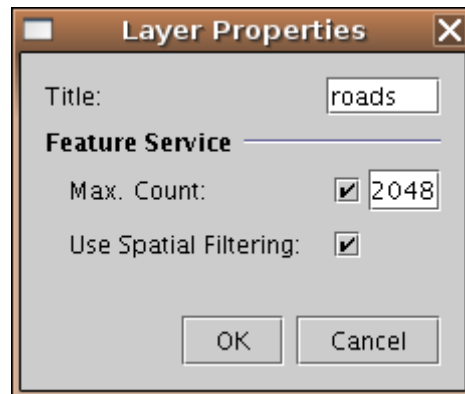
2. Select the **Layer Properties** menu item.

Figure 48: Layer Properties Menu Item



3. In the Layer Properties window:
 - Toggle the **Max Count Checkbox** to activate or deactivate it.
 - Edit the **Max Count Text Field** to specify the retrieved feature limit.

Figure 49: Max Count in Layer Properties Window



4. Select **OK**.

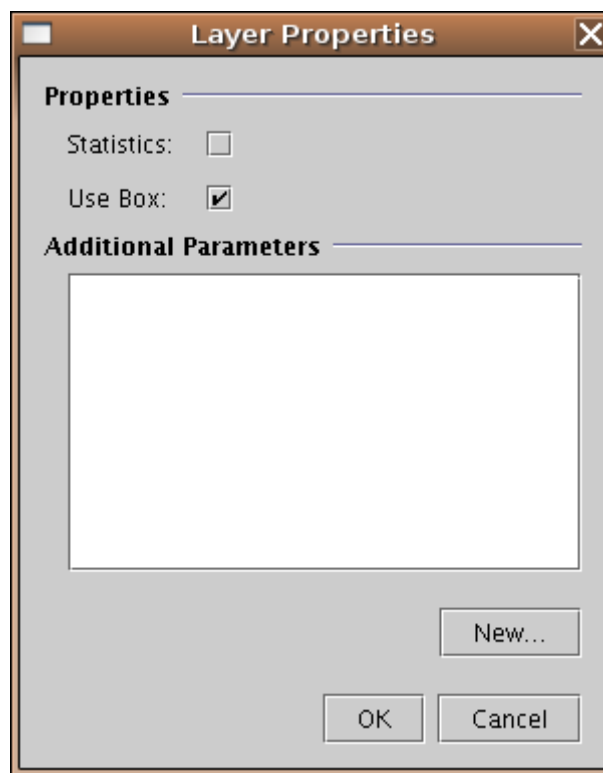
Spatial Filtering

Spatial filtering is used to limit the retrieved information to the area being currently displayed on your screen. This may be used both with WFS and WMS servers.

Procedure Configuring the Spatial Filter

1. In the Style panel, **right-click in a feature or map layer**. Its contextual menu will appear.
2. Select the **Layer Properties** menu item.
3. In the Layer Properties window:
 - If configuring a feature layer, toggle the **Spatial Filtering** Checkbox.
 - Else, if configuring a map layer, toggle the **Use Box** Checkbox.

Figure 50: Use Box in Layer Properties Window



4. Select **OK**.

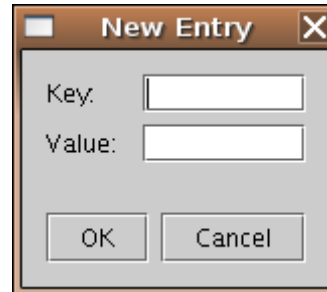
Additional Parameters

Some servers accept proprietary parameters in their requests (ServiceName, Quality, ...). Since this is not standard regulated, there is no way to anticipate which parameters those servers use and its functionality. Nevertheless, in order to provide our clients with the up most interoperability, ERDAS APOLLO Style Editor allows you to specify additional parameters to the server requests.

Procedure Adding additional Parameters

1. In the Style panel, **right-click in a map layer**. Its contextual menu will appear.
2. Select the **Layer Properties** item.
3. Select **New** in the Layer Properties window.
4. In the New Entry window, insert the server specific **Key** and **Value**.

Figure 51: Additional Parameter New Entry Window



5. Select **OK**.

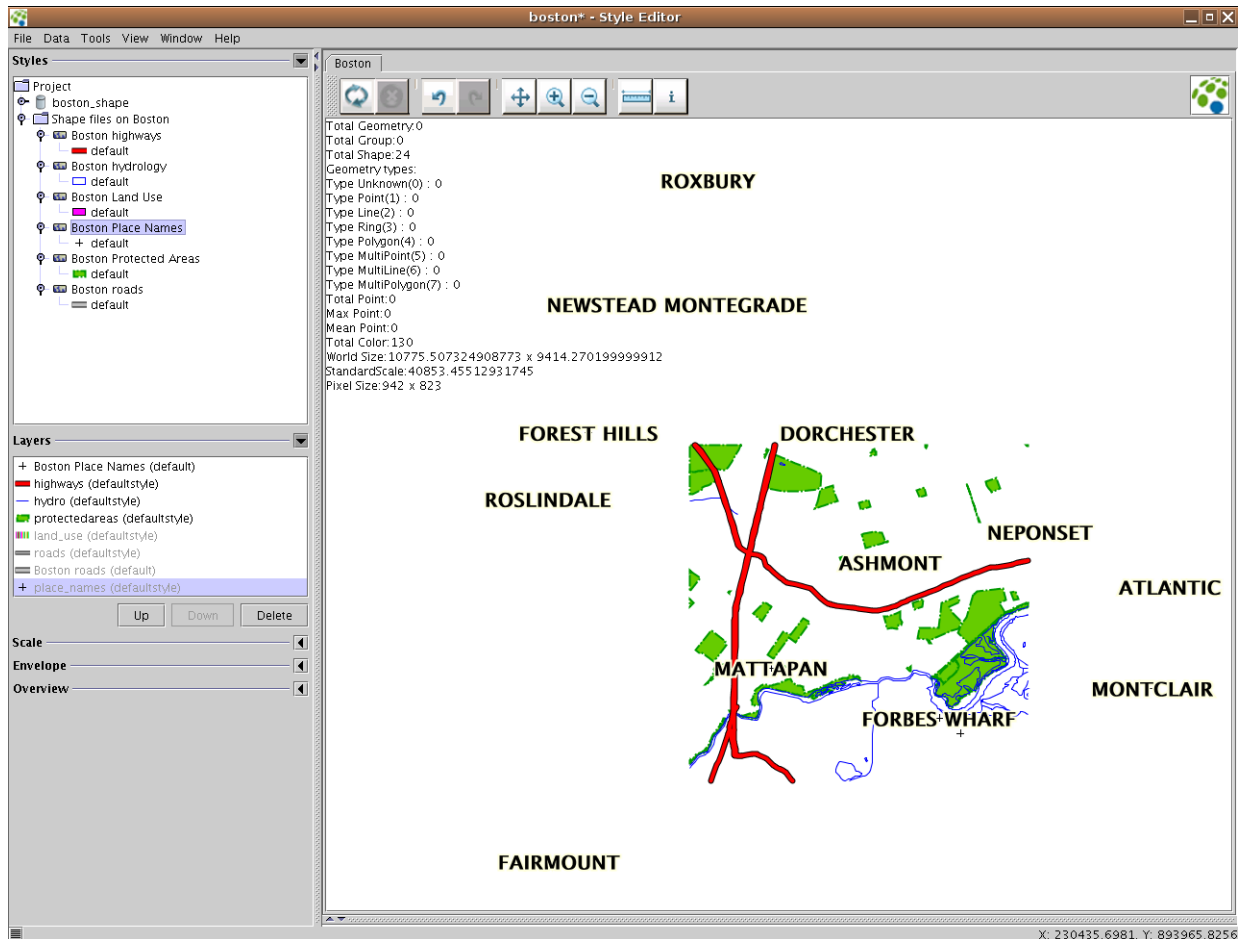
Layer Statistics

The statistics option allows you to obtain detailed information on the elements contained in WMS layer built over a WFS. This information includes the number of retrieved geometries, the type of geometries... This is a feature only available with erdas-apollo servers.

Procedure Activating Layer Statistics

1. In the Style panel, **right-click in a map layer**. Its contextual menu will appear.
2. Select the **Layer Properties** item.
3. Toggle the **Statistics** checkbox to activate/deactivate this option.
4. Select **OK**.

Figure 52: Layer Statistics with boston_shape



Map Navigation

This section presents the ERDAS APOLLO Style Editor navigation tools that allow you to change the current view of the map by zooming, panning or quickly jumping to specific locations.

Zoom and Pan

This section describes the zoom and pan tools provided by the ERDAS APOLLO Style Editor. Note that these tools remain selected until you choose another one.

Zoom Tools

Three zooming tools are available through the use of two icons:

- The tool allows you to zoom in by either one point (a single click) or two points (by drawing a box).
- The tool allows you to zoom out by one point (a single click).

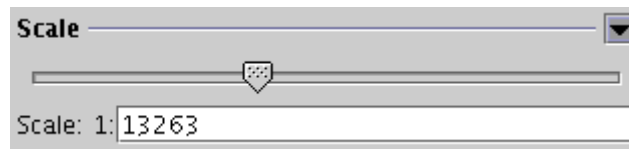
Pan Tools

Panning can be achieved by using the icon in the toolbar. This tool allows you to recenter the view on a point indicated by a single click onto the map.

Changing Scale

You can also use the Scale panel to change the current scale, which is also a kind of zooming tool. The Scale Panel allows you to change the scale using the slider or to enter a scale value in the text field.

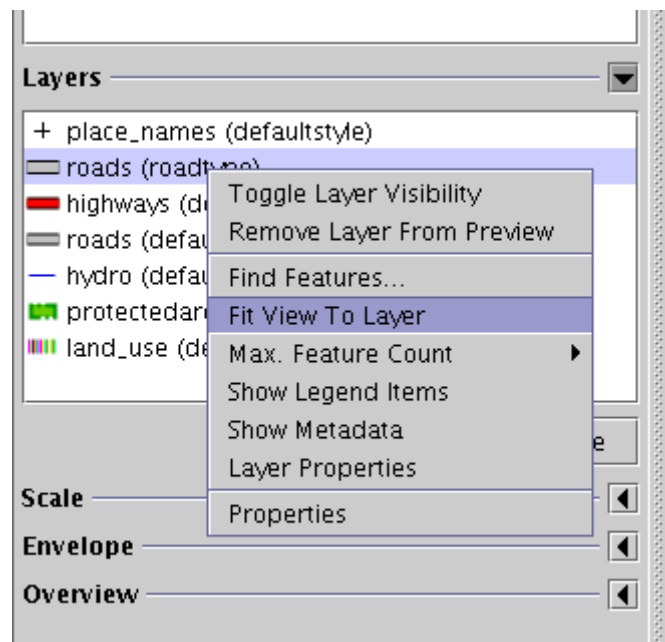
Figure 53: Change Scale



Fit Envelope to Layer(s)

This is the quickest way to view an entire layer content. In the Layers Panel, select the layer to extend to by right-clicking it and choosing **Fit View To Layer** in the menu, as shown below:

Figure 54: Fit To Layer



When executed on a single layer, the current map envelope is adapted to the chosen layer envelope. If you choose several layers, then the resulting envelope is the smallest envelope that surrounds every layer envelopes.

Switching Map Extents

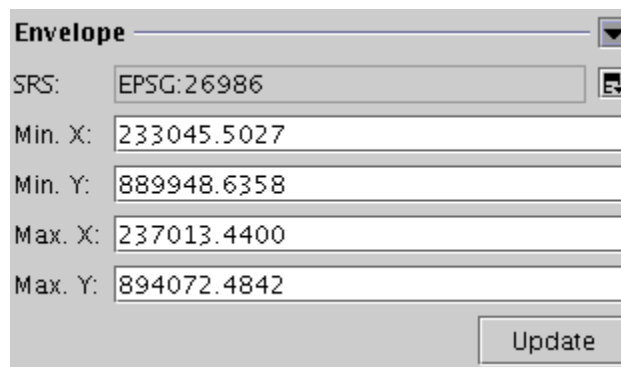
You can quickly navigate to a previous extent of the map, i.e. the area you were previously on the map before navigating to some other area, by using the icon in the toolbar.

Clicking the icon will take you back to a more recent map extent.

Envelope Manipulation

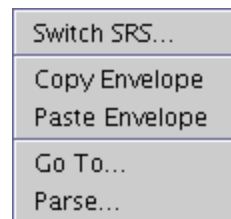
Using the Envelope Panel, you can view and change the coordinates of the current view box of the map, as well as the Spatial Reference System.

Figure 55: Envelope Panel



The Envelope Panel also offers additional features such as parsing of bounding box, copy and paste of envelopes, etc. To start using these features, click the icon and select the appropriate menu entry:

Figure 56: Envelope Menu

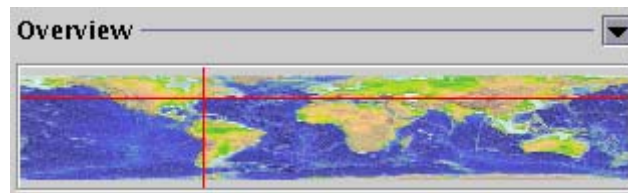


Map Overview

ERDAS APOLLO Style Editor offers, as a collapsible panel, an overview map, or index map. The overview area gives an indication of where the current map view is situated within a broader context, e.g the world.

The box of the active map will be represented on the overview map as either a red cross or a yellow rectangle area, depending on whether the box of the active map can be drawn meaningfully as a rectangle area on the overview map.

Figure 57: Overview Panel



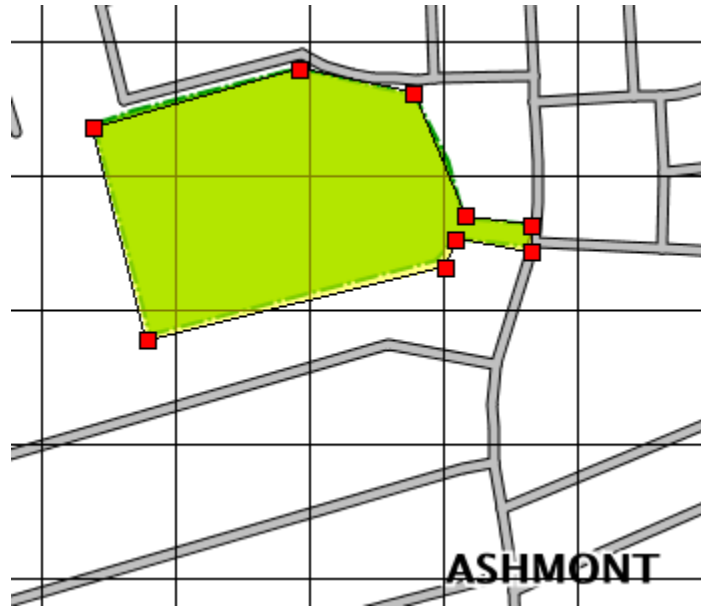
Miscellaneous Tools

This section presents the various ERDAS APOLLO Style Editor tools related to map navigation.

Compute Distances and Areas

The Compute Distance and Area tool uses a *Geometry Editor*. This Geometry Editor will let you draw polygons while displaying, in the status bar, the distance and area of the currently drawn geometry. To use the Geometry Editor, please click on the icon.

Figure 58: The Geometry Editor



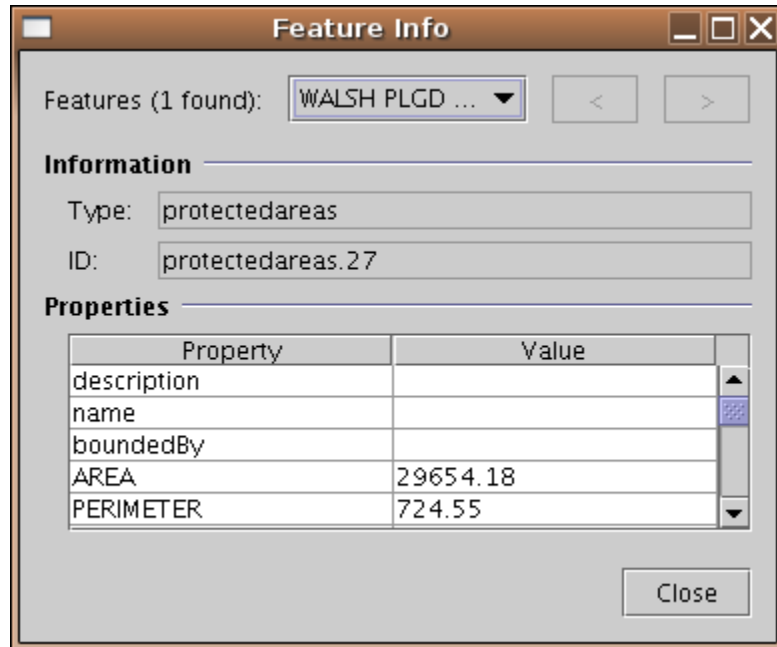
You can add points using the left mouse button and remove a point with right-click.

Simply use any other tool to quit the Geometry Editor.

View Feature Properties

Using the icon, you can obtain information about features that are located in a given box. The kind of information you will obtain is dependent on the type of the feature. To use this function, click the above icon then drag a rectangle on the map. A window, like the one below, will appear:

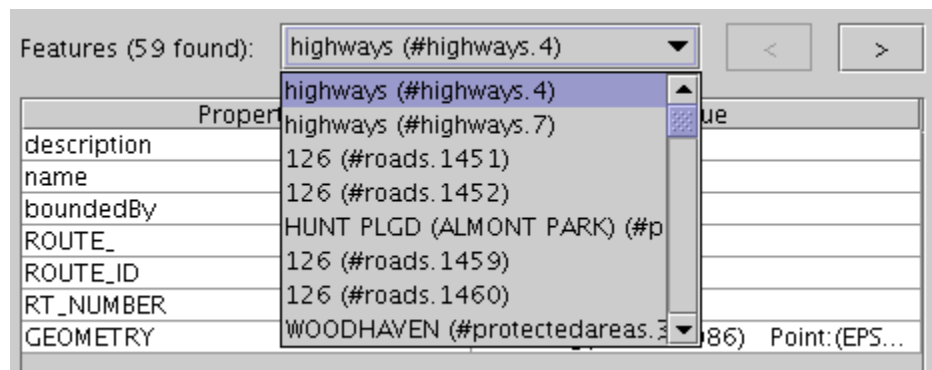
Figure 59: Feature Info



All properties of the selected feature are listed in the two-columns table. Note that, in the example above, the selected feature is of type *protectedareas* with ID *protectedareas.27*.

The drop-down list contains all the features that were found in the selected box. To view the properties of another feature, simply select it from the drop-down list or click the left and right arrows to navigate in the list of features, as shown below.

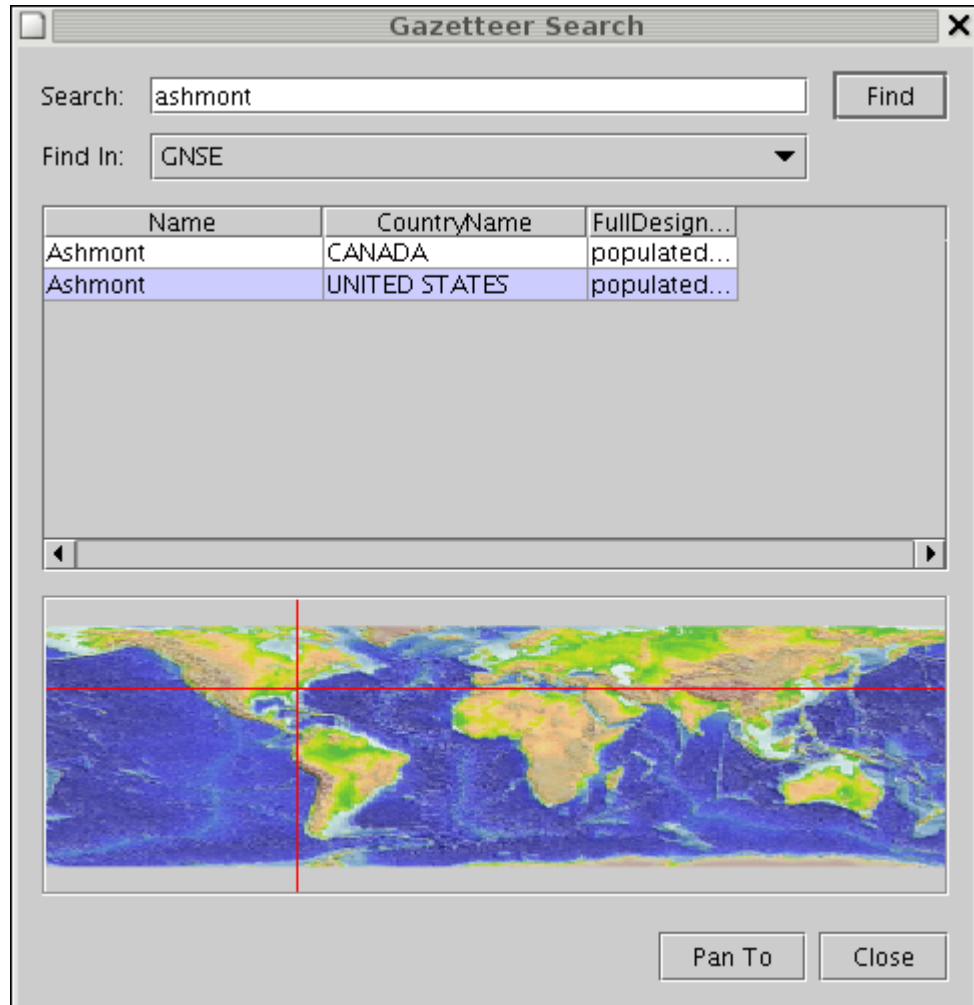
Figure 60: List of Features



Gazetteer

You can now use ERDAS public gazetteer service to quickly navigate to well known places. Use the Tools/Gazetteer menu entry to view the Gazetteer panel, as shown below:

Figure 61: Gazetteer



Enter your search criteria (% wildcard allowed) into the Search text field then click on Search. All the results are presented in the table, you can click on each column header to sort the rows. Choose a location then click on Pan To to center the map on this location. Press the Close button to exit.

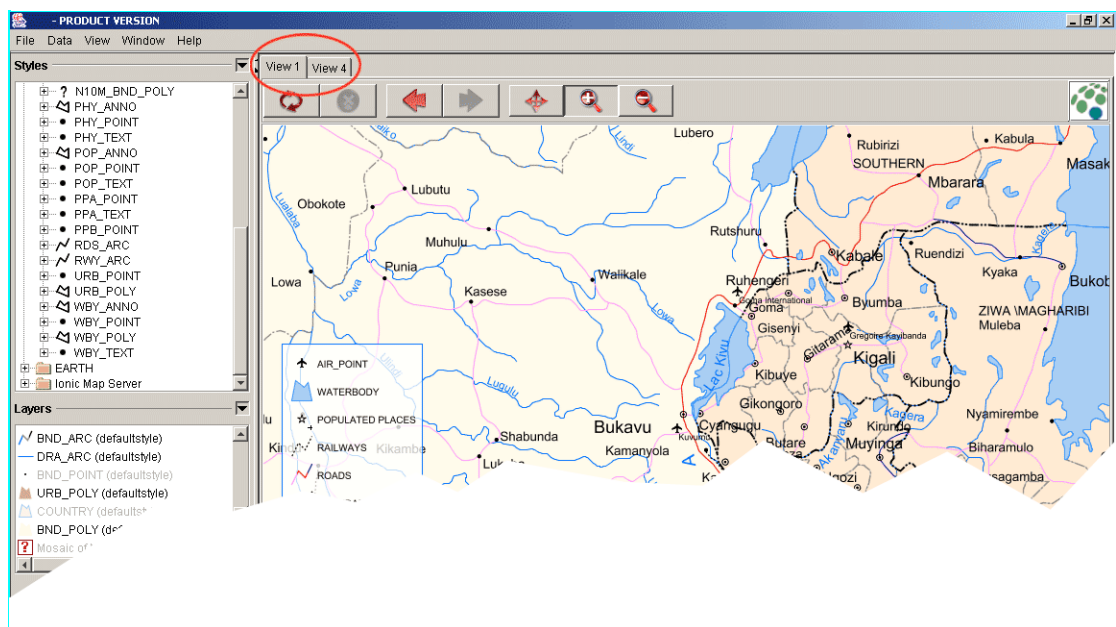
Views

The ERDAS APOLLO Style Editor uses the concept of *views* to display maps. A view is an interactive map that lets you display, explore, query, analyze and style geographic data in the ERDAS APOLLO Style Editor. Views are saved in the ERDAS APOLLO Style Editor project you are currently working with.

A view defines the geographic data that will be used and how it will be displayed, but it does not contain the geographic data sources or map servers themselves. Instead, a view references these data sources. This means that a view is dynamic, because it reflects the current status of the data source. If the data source changes, a view that uses this data will automatically reflect the change the next time the view is drawn. This also means that the same data can be displayed on more than one view. Another powerful functionality of the ERDAS APOLLO Style Editor is that it allows you to share styles between different views. Each style created in the scope of a project can be used to render data in any view of that project!

With this new feature, in no time you'll be working with your data in a completely new way. You will be able to easily compare maps without having different instances of the ERDAS APOLLO Style Editor open, you will be able to preview your maps at different scales, and see how your maps look like on a specific device. Additionally, you will be able to export your view as an OpenGIS Context!

Figure 62: Views



Creating a View

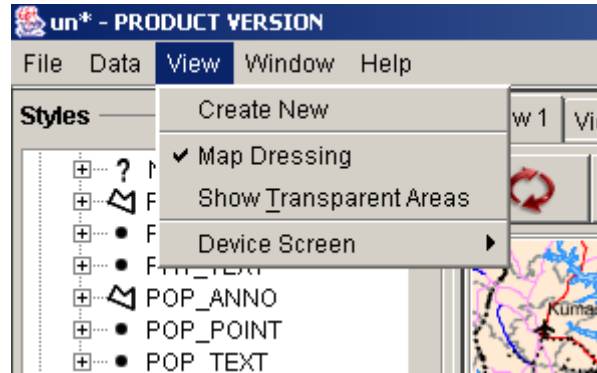
A default view, called View 1 will be created when you initiate a new project.

If you want to create a new view in your project, you can do it in two ways:

Procedure Create a New View

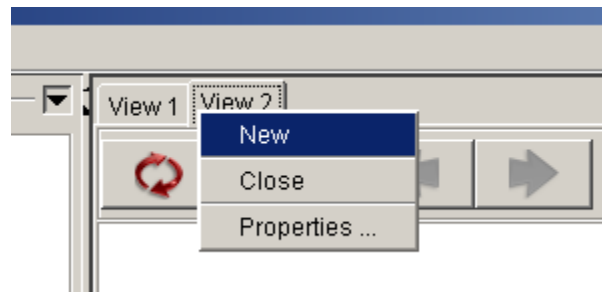
1. In the View Menu, select **Create New** a new view (2, 3, 4,... depending on the number of views you have already created) will be displayed.

Figure 63: Create a New View - Method 1



2. Or you can right click on a existing view and hit **New**

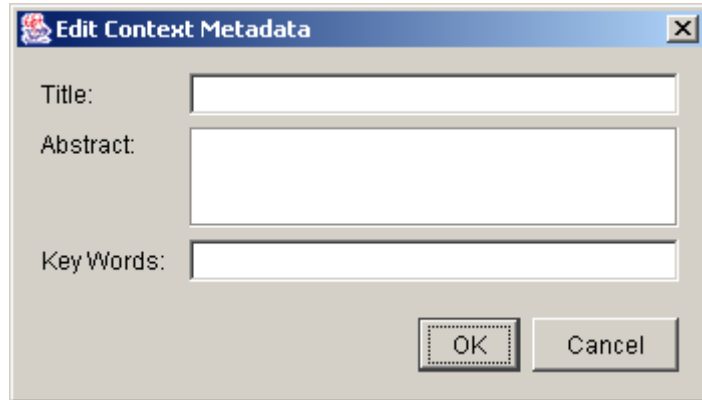
Figure 64: Create a New View - Method 2



View Properties

Since a view can be exported as an OpenGIS Context, the ERDAS APOLLO Style Editor offers you the possibility to set some OpenGIS Context attributes such as the Context Title, Abstract and Keywords.

Figure 65: View Properties

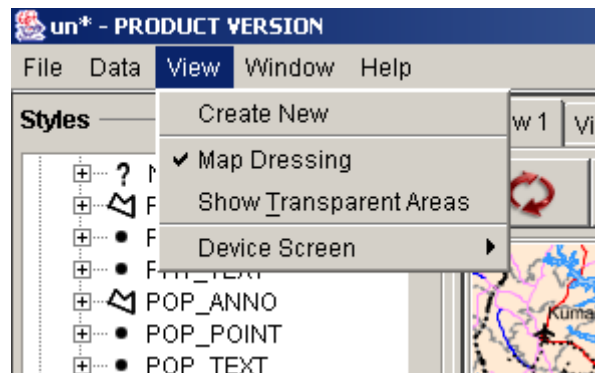


Configuring a View

Enabling Map Dressing

Map dressing is the option to add a legend, a scale bar, or a grid to the map. You can enable or disable Map Dressing by selecting or un-selecting the **Map Dressing** entry in the View Menu.

Figure 66: Enabling Map Dressing



See below for a map with map dressing enabled.

Figure 67: View with Map Dressing Enabled



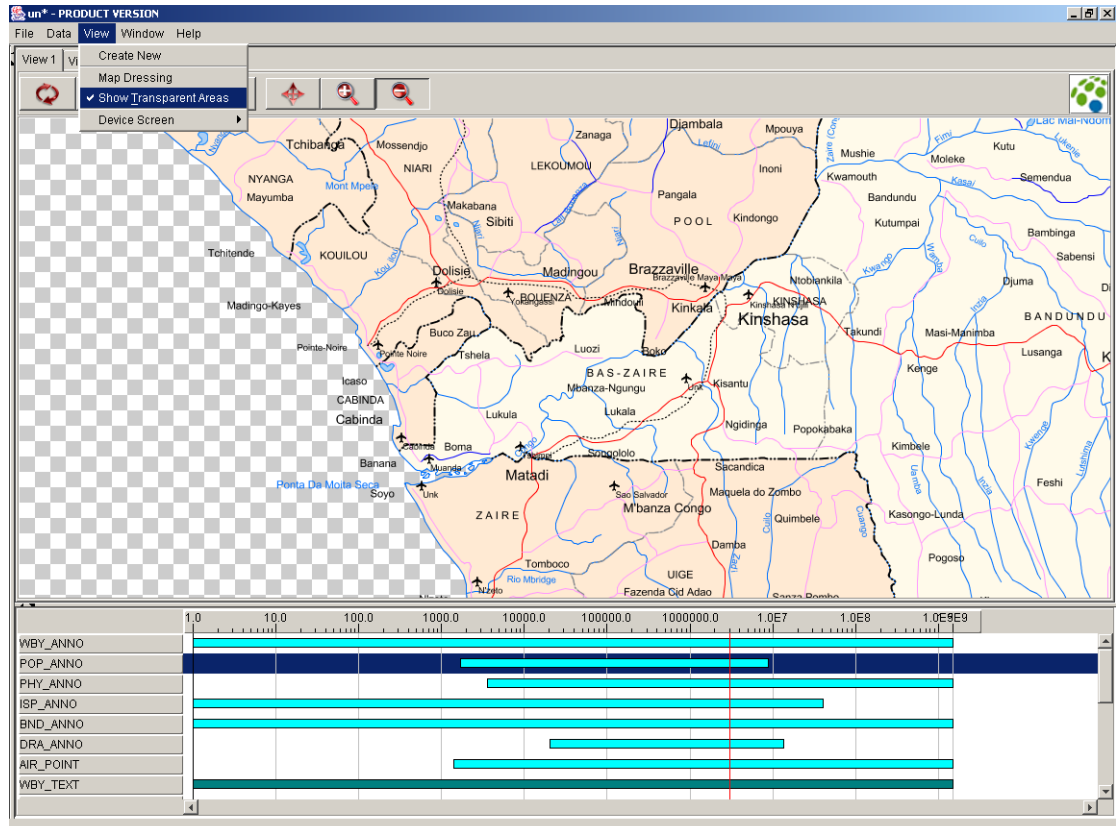
*If map dressing hasn't been configured before being enabled, you will not see any result from the **Map Dressing** menu command. Map dressing configuration will allow you to select what kinds of dressing elements you want to add to your map, such as a legend, a grid, a scale bar, etc.*

To configure map dressing, right-click on the data source to configure in the data source panel and select the **Edit Dressing Style...** menu entry.

Show Transparent Areas

To display the transparent areas, you need to hit the **Show Transparent Areas** entry in the View Menu

Figure 68: View with Transparent Areas



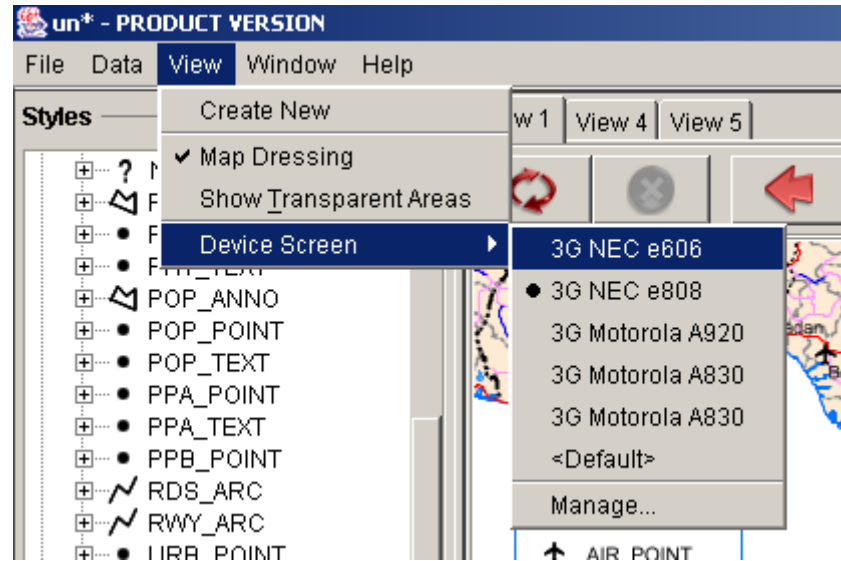
Configuring a Device

The view menu contains an option referred to as **Device Screen**. You can use this option to display a sub-menu with pre-defined devices.



By "Device" we mean a real live tool in which we can display maps such as a 3G mobile phone or a palm. The ERDAS APOLLO Style Editor can help you to design maps for these devices with the right settings (color depth, size), so that you can preview them as they will be displayed on the device.

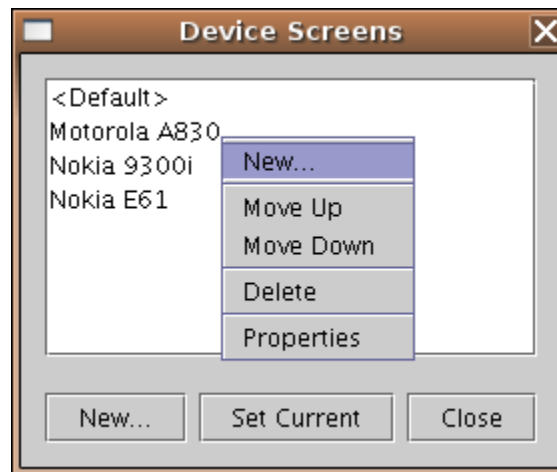
Figure 69: Select a Device



The last item of the Device Screen sub-menu is the **Manage** option. You can use this option to reference the list of pre-defined devices, or to add, remove or define new specific devices from the **Device Screen** sub-menu.

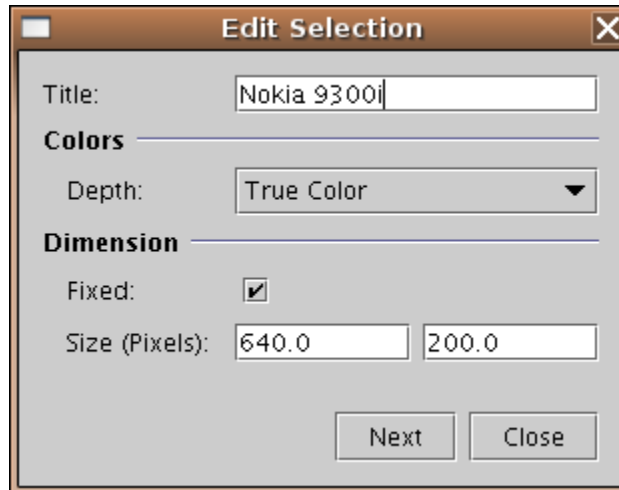
To add a new device you will need to right click in the Device Screen window and then hit the **New** option.

Figure 70: New Device



This screen will allow you to re-order the list of devices using the **Move Down Move Up** items. You can also easily delete a device or edit the device's properties by double clicking on the device name or using the **Properties** option in the pop-up menu.

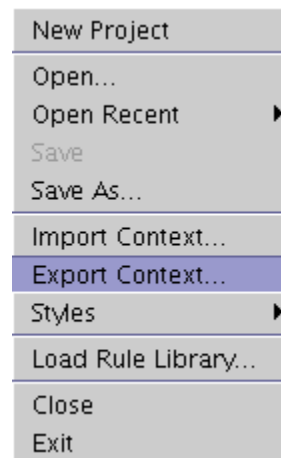
Figure 71: Configure Device



Saving a View as a Context

The current view can be saved as an OpenGIS context. Saving the context means to serialize the current view and its associated data sources to an XML file on your hard disk. To save the current view as a context, select the **Export Context...** from the File menu.

Figure 72: Export Context



You will then need to locally save the context XML file using the standard **Save As...** dialog. The XML context file can be later re-used by importing it into ERDAS APOLLO Style Editor.



When exporting a context, if some of the data are local shapefiles, a messages warns that those layers will not be exported and proposes to proceed with the export or to abort it.

Styling Data

Brief Introduction to Styling

What is Styling

Styling is the process of creating stylesheets, which are sets of parameters to be applied when portraying a particular feature type. Please refer to [Portrayal Capabilities](#) for a thorough discussion of the portrayal service.

Styles and Rules

Each style produced by the ERDAS APOLLO Style Editor is based on a particular portrayal rule.

Portrayal rules and styles are two separate concepts. You need both rules and styles to portray data, but each entity provides a different level of service. Rules are pieces of program code providing a certain way for portraying data such as classification upon numeric value. Styles are text files that contain parameters defining how to portray a dedicated set of data. For example, a style will define which field the portrayal engine should classify upon and what fill color and stroke width are to be used.

The ERDAS APOLLO Style Editor Styling Architecture

One of the important benefits of using the ERDAS APOLLO Style Editor tool to prepare styles is the possibility to visualize their effect while fiddling with parameters, before server-side publication. To accomplish this, the tool embeds the following components:

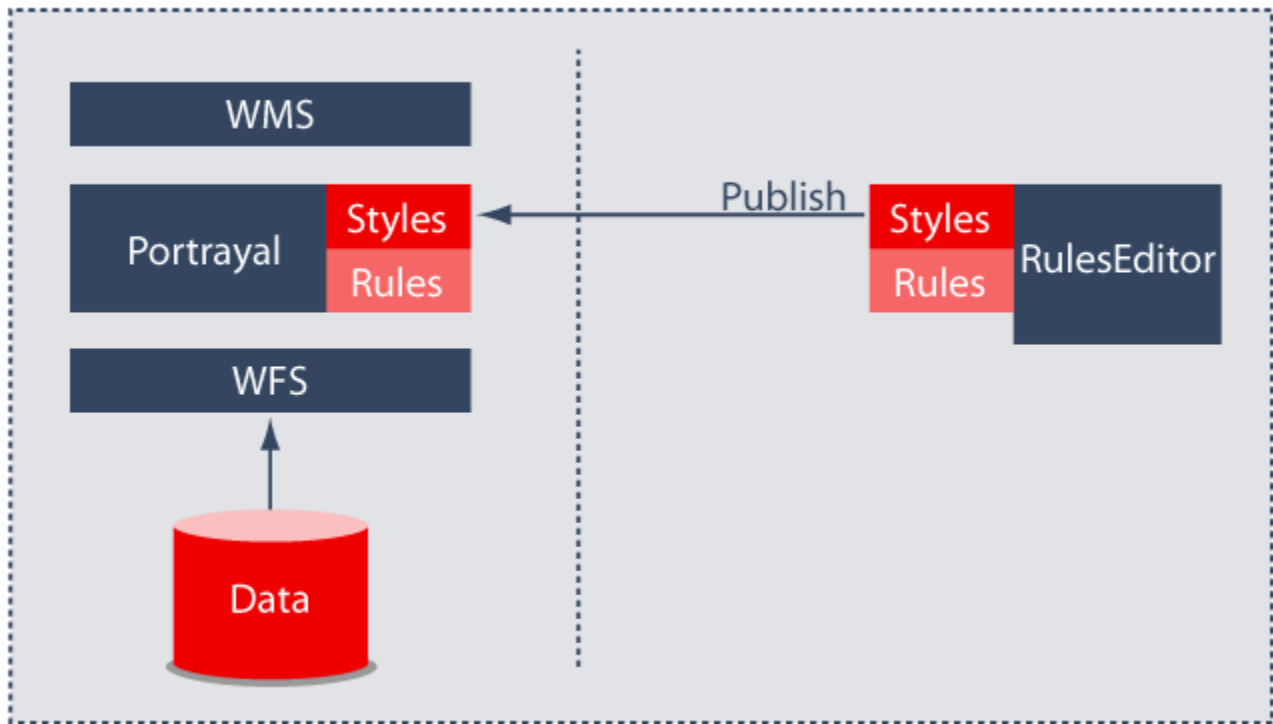
- A "client" version of ERDAS's versatile portrayal engine, with support for plug-in styling rules.
- The same bundle of styling rules that is available on the server side, with additional configuration/user interface descriptors.

The rules are used in three distinct ways by the ERDAS APOLLO Style Editor tool:

- To create and present you with an user-friendly user interface that allows intuitive parameter editing/selection.
- To generate preview renderings of the style against sample feature sets extracted from your real-world data.

- To create style packages that will later be published on the server side. The published styles will then be available via the WMS interface of the service.

Figure 73: The ERDAS APOLLO Style Editor Architecture



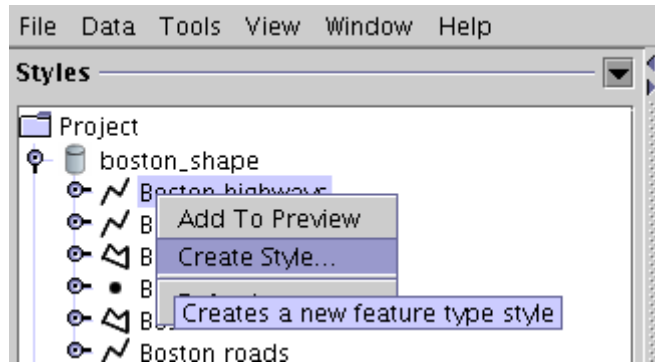
The above picture shows how the rule bundle is always available on both the client and server side, while the style packages produced by the ERDAS APOLLO Style Editor tool are published on the server.

Managing Styles

Creating Styles

You can create a new style by executing the Style Wizard, which is accessible by selecting **Create Style...** from the contextual menu of any feature type:

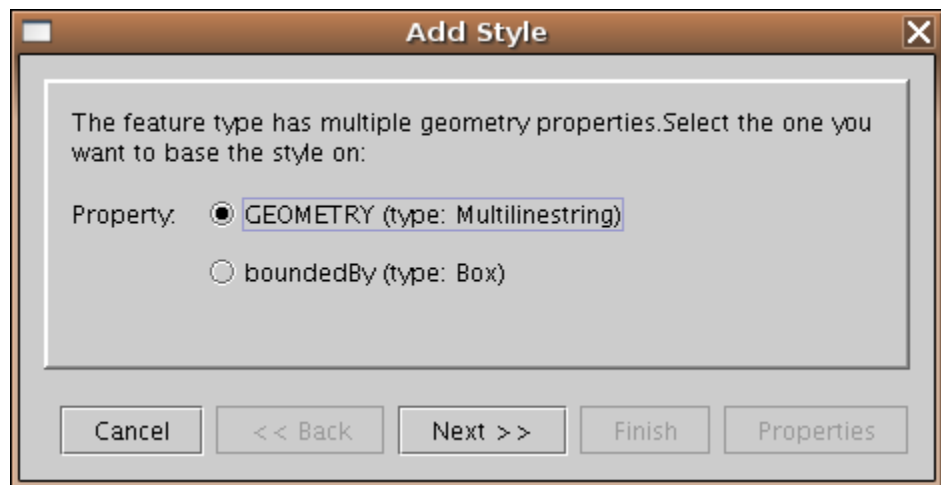
Figure 74: Create Style Menu



The Style Wizard guides you through the following steps:

1. Right-click the source you want to use to obtain its contextual menu.
2. Select the option Create Style.
3. Select the geometry type you want to use.

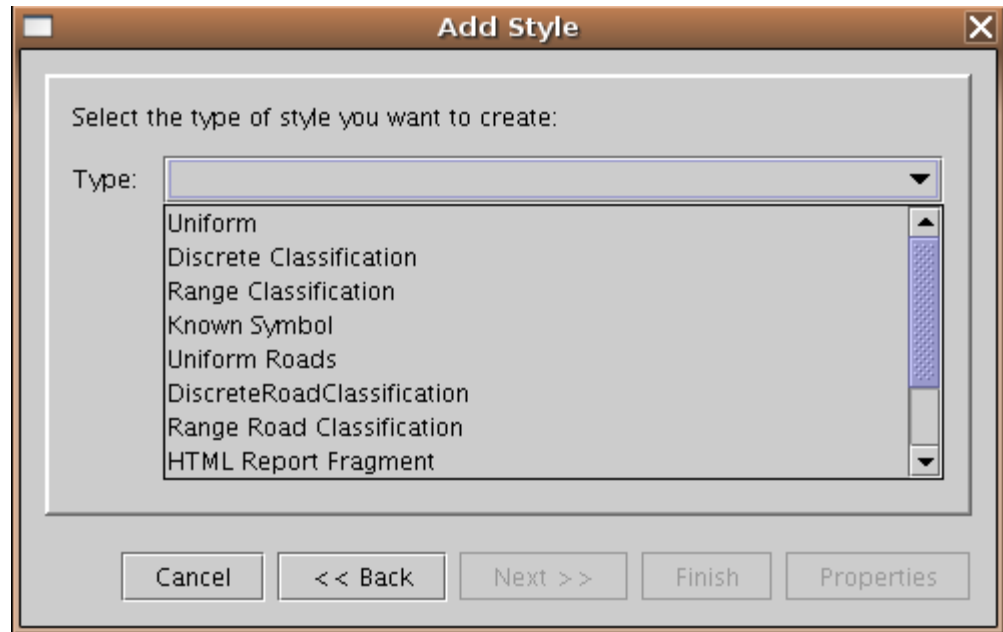
Figure 75: Geometry Property Selection



This step will only appear for sources with more than one geometry type defined.

4. Choose if you prefer to create new style or use one of the predefined styles, then click **Next**.
5. Select the styling rule your style will be based on from the drop-down menu (a small description of the rule behavior will appear under the drop-down control when available), then click **Next**.

Figure 76: Styling Rule Selection

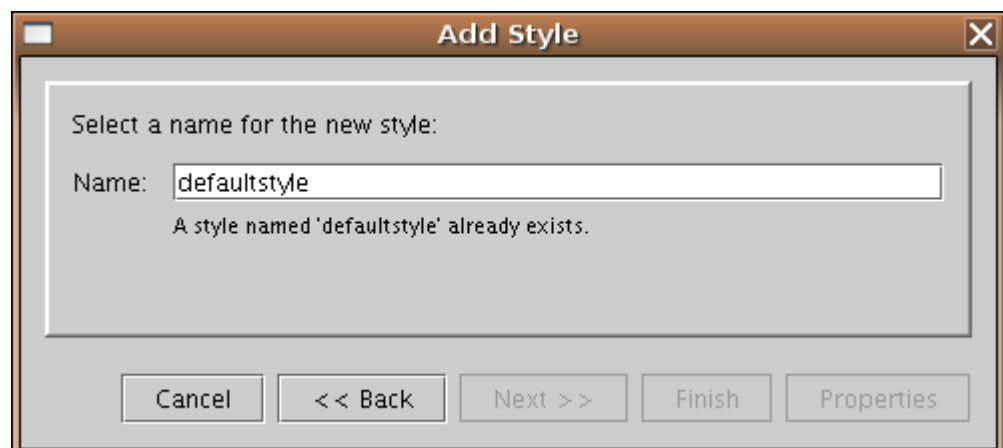


6. Enter the name that will be used to reference the style once it is deployed on the server, then click **Finish** or **Properties...** to create the style and insert it in the database.

Since style names have to conform to the WMS naming conventions, the **Finish** and **Properties...** are enabled only when the name is valid. While the name is invalid, a descriptive tip explaining the cause should appear under the control.

The **Properties...** button performs the same action as **Finish**, and then invokes the inspector for you to edit the style parameters.

Figure 77: Name Selection and Validation



Deleting Styles

You can delete an existing style by selecting **Delete** from the contextual menu of the style.

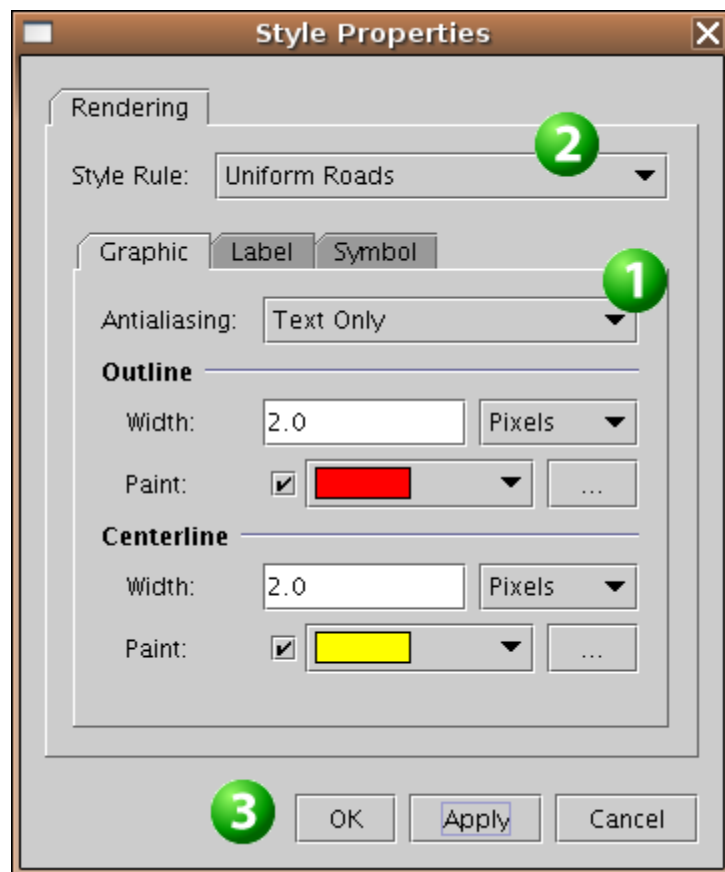
If you confirm the deletion, the style is removed from the database and every preview layer that depends on it is also deleted.

Modifying Styles

You can launch the style editing dialog by double-clicking on the layer in the preview panel or by highlighting the layer in the panel, right-clicking, and selecting the properties item from the pull-down menu.

The style editing dialog is composed of the following parts:

Figure 78: Style Editing Dialog



- **1** the central part of the dialog embeds an inspector for the currently selected styling rule. You will find the description of these inspectors in [Rules Reference Guide](#).

- **2** selects the rule the current style is based on. If you select another rule, the ERDAS APOLLO Style Editor tool will try to migrate your current style settings to the new rule (when applicable).
- **3** the action area contains the various style application commands which are detailed below.

The following commands are available in the action area:

- **OK:** Applies the current settings to the style, commit them to the style database, and closes the editing dialog.
- **Apply:** Temporarily applies the current settings to the underlying style, and causes the views that depend on it to refresh themselves. The **Apply** button does not close the dialog, and can be clicked more than once.
- **Cancel:** Undoes any temporary changes made to the underlying style and then closes the editing dialog.

Rules Summarized

The styling rules provided with the ERDAS APOLLO Style Editor are:

- **Uniform** - Uniformly applies a simple style to every feature. The stroke, fill and symbol to use can be configured for the whole feature collection, and a property of the feature can be used for labeling.
- **Discrete Classification** - This style rule should be used for displaying categorical data (data values such that the symbol for one value is no more or less prominent than the symbols for any other value). It also handles lines' and polygons' stroke and fill color variations as well as line/outline width.
- **Range Classification** - This style is used to classify raw data (such as population counts), ranked data (to show a progression of values such as best to worst scholastic scores in a given region) or to represent percentages (such as percentage of given area that are affected by pollution).
- **Known Symbol** - Applies a fast-to-render marker (from a fixed, predefined set) at the centroid of each feature. A property of the feature can be used for labeling.

- **HTML Report Fragment** - This Style Type allows you to render a feature collection into a HTML fragment. You may set a sub-title that will appear in the output for this feature type. Please refer to Chapter 10 for the HTML Report layout construction.
- **Uniform Roads** - This is a style type dedicated to the display and portrayal of various types of roads. It allows you to configure an outline, fill color, and center line to line or polyline geometry; it can also label the road with a property of the feature.
- **Discrete Road Classification** - This style type is used to render roads with a discrete classification that affects outline and center line colors.
- **Range Road Classification** - This style type is used to render roads with a range classification that affects outline and center line colors.
- **Variable Markers** - This styling rule marks features with scaled and (optionally) rotated symbols. The size and orientation are determined from one of the properties of the feature.
- **Patterner** - This styling rule fills polygons with patterned backgrounds.
- **Numberer** - This styling rule marks the features that are the nearest from the map center with sequential numbers.
- **Symbol Roller** - This styling rule renders linear geometries by stamping a list of symbols along the curve in a cyclic manner.

Styling Various Types of Geometries

Styling is specific to geometry contained by the selected layer.

Table 13: Graphic Options According to Geometries

Points	You can select the specific marker desired and set display properties, such as color, size and labels.
Lines	You can set a color, width, and dash type for lines as well as determine linear capping and join parameters.
Polygons	You have the same styling options as available for Lines, as well as the ability to add a fill color.

Deploying Styles

The easiest (and recommended) way to deploy styles is to package them in a `GAR` archive which will be subsequently dropped in the rendering directory of the portrayal service. Please refer to the *Administrator's Guide* for more information about the rendering directory organization.

If you are saving your styling project in a `GAR` package (which is the default format in this version of the ERDAS APOLLO Style Editor tool), no additional steps are necessary: you can just drop your project file as-is in the rendering directory.

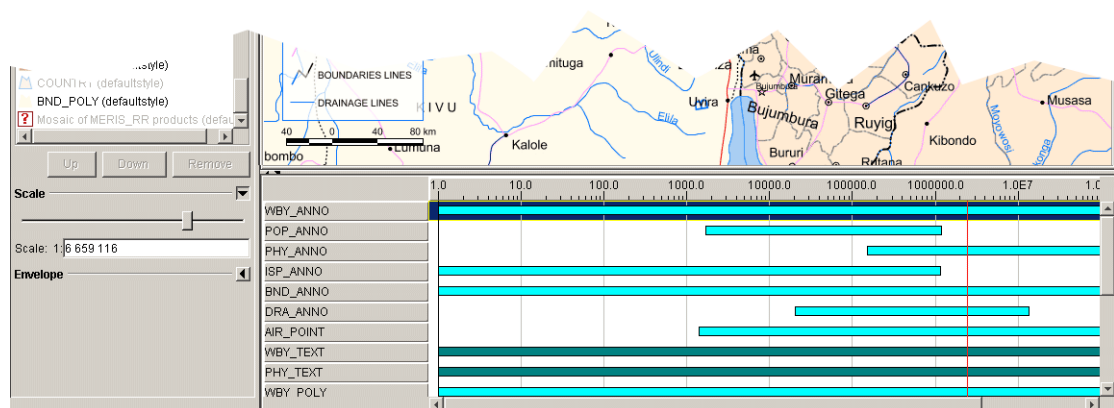
If you are using an old-style `.styler` archive, you can generate a `GAR` archive using the **File/Styles/Create Bundle...** command. This command generates the same metadata-augmented archive format than **Save As...**, but does not change the filename and package format of the current project.

Scale Range Management

What is Scaling

Scaling is an important notion in GIS. When manipulating data, one needs to be able to determine and decide at which scale to do rendering and to display layers. It is now possible to easily deal with scaling in the ERDAS APOLLO Style Editor. A new scale panel can help you to specify which layer should be displayed at any particular scale.

Figure 79: Scale View



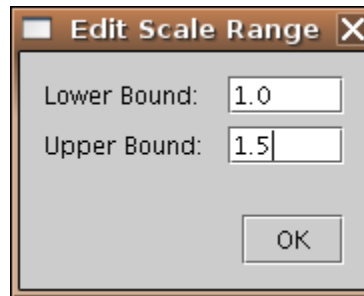
Details of the Scale Panel

The scale panel contains the list of the layers selected in a particular view. The scale panel can be activated by dragging the icon located in the bottom left corner just below the map. The map can be viewed at different scales by moving the red scale cursor to different points along the scale line.

Changing the scale line maximum and minimum values, will result in a different mix of the features being displayed, depending on which feature's range bars are dissected by the scale cursor. This change can be done by three different ways. You can either:

1. Double click on the scale bar, directly accessing the Edit Scale Range Window.
2. Right-click in the scale bar, and select Properties in the contextual menu.
3. Drag one of the extremities of the bar with the mouse pointer.

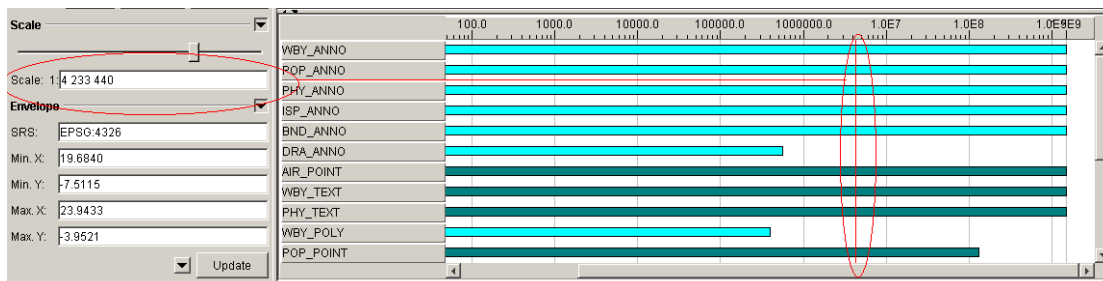
Figure 80: Edit Scale Range Window



Feature types not toggled in the layer visibility will be highlighted in dark green on the Feature Type list and will not be affected by the scale line placement.

Changing the scale by moving the red scale cursor to different points will automatically update the view's scale box and of course the current bounding box.

Figure 81: Changing the Scale



In order to configure a WMS server with the scale configuration you designed, you have to export your project as a context, and based on it, build a ContextProvider. More details on how to export a context in [Saving a View as a Context](#).



When the WMS server you access publishes a "Scale Hint" information for a layer, this hint is converted to the Scale Range for this layer in the ERDAS APOLLO Style Editor. Further changes in the Scale Range panel will override the Scale Hint.

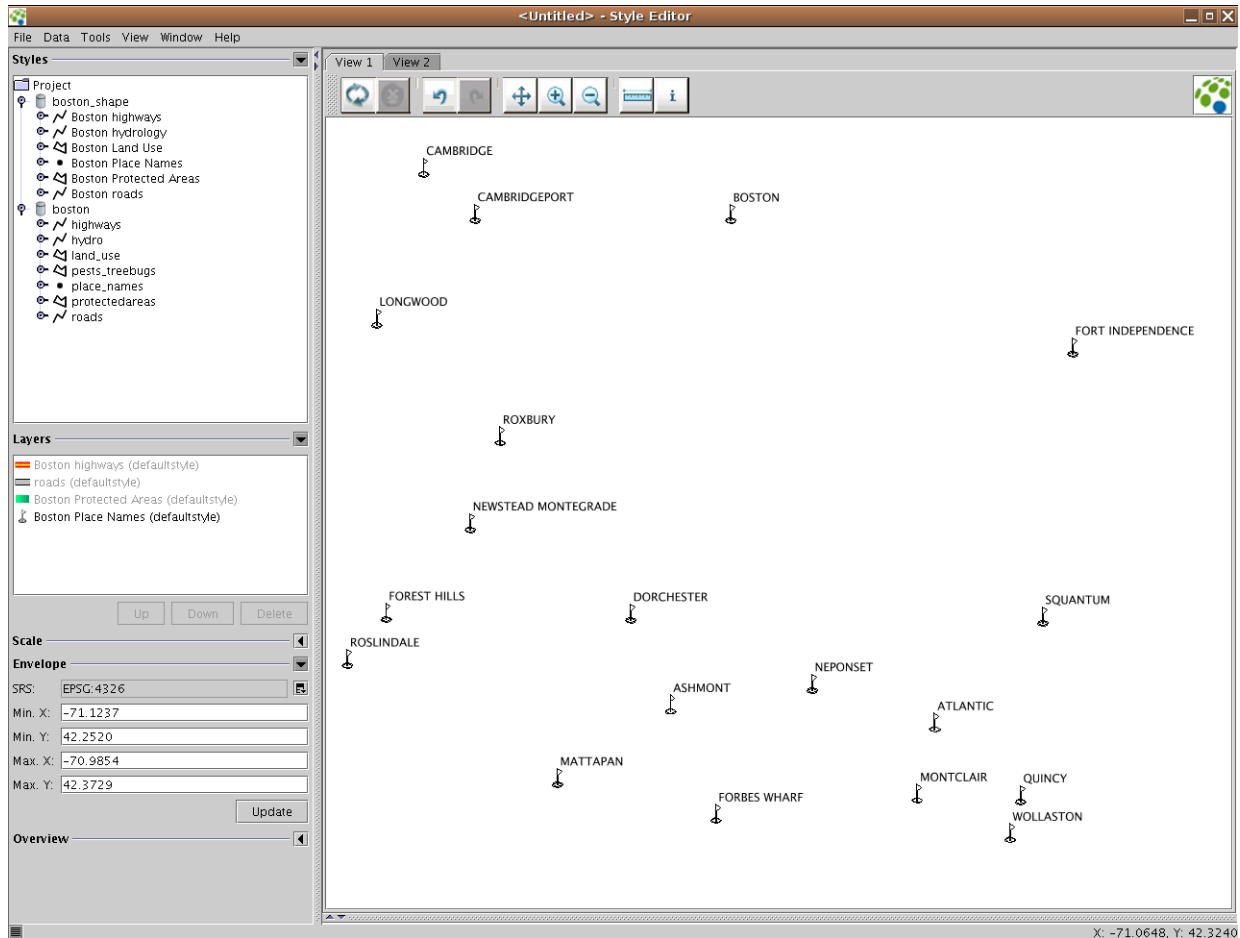
Rules Reference Guide

"Uniform" Rule

The "Uniform" styling rule is valid for simply applying the same default style (colors and symbols) to all of the features types in the layer. This is useful for simply demonstrating where the features of the layer are located.

Whether you want to portray points, lines or polygons, this basic rule will enable you to control the various parameters relevant to the chosen geometry.

Figure 82: Point Style Example



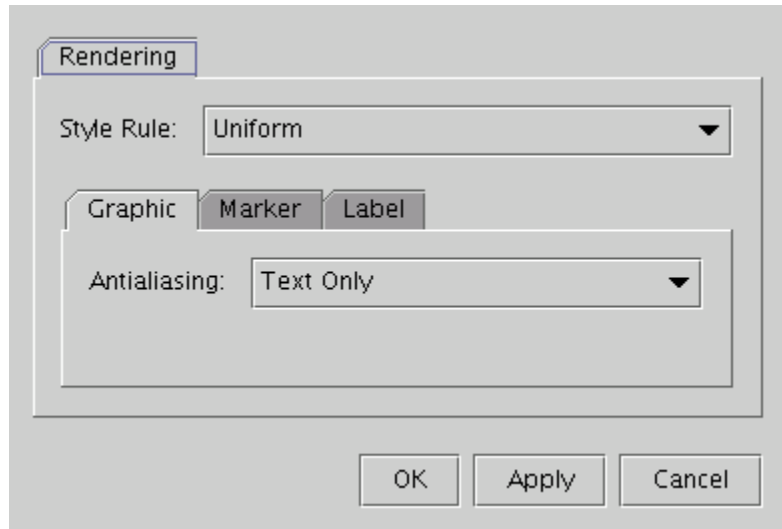
Styling Points

The "Uniform" styling rule, like the other rules provided within the ERDAS APOLLO Style Editor, is customizable through a popup window composed of a set of panels. Parameters are organized by panels according to their geometry. There are three panels available for styling point layers.

- The Graphic Panel: This panel (for points) allows you to select the antialiasing option.
- The Marker panel: This panel allows you to select options about the desired marker image, its size and color.
- The Label panel: This panel allows you to control your label options.

The Graphic Panel

Figure 83: Uniform - Graphics Panel (Point Mode)



- **Antialiasing** - Antialiasing is a computer graphics technique used for smoothing jagged edges in text characters and line segments. Enabling it may dramatically improve readability, but may affect the ERDAS APOLLO Style Editor's performance.

The following anti-aliasing options are available to you from the drop-down menu:

Table 14: Antialiasing Options

None	Antialiasing is not applied at all, resulting in faster performance but jagged edges for text characters and line segments.
Shapes Only	Antialiasing is applied only to line segments. Text at small sizes may appear jagged.
Text Only	Antialiasing is applied to text only, but line segments are not smoothed.
Full	This is the default value for this option. Antialiasing is applied to both text and line segments. This is the smoother and the slower rendering mode.



Please read the "Common" chapter for more information about the antialiasing process within the ERDAS APOLLO Style Editor.

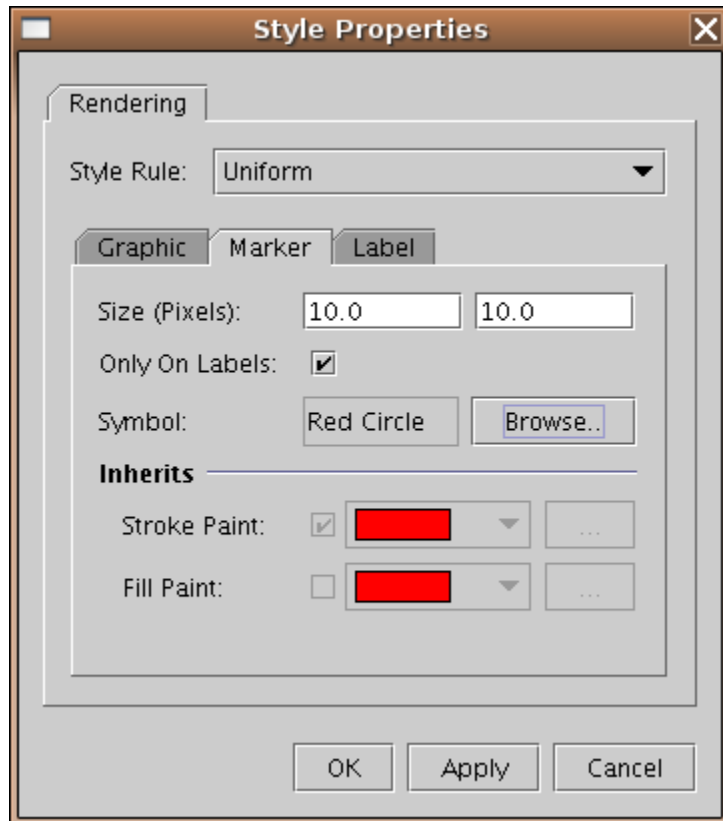
The Marker Panel

The Marker Panel allows you to pick the type of marker to portray your point symbols. Additionally, you may customize additional parameters to modify the marker's appearance such as the marker size, labeling properties, symbol type, color and fill.



This option is only present with the point geometry type.

Figure 84: Uniform - Marker Panel



- **Size** - Two values, expressed in pixels, define the width and height of the marker.
- **Only On Labels** - This option defines whether the marker will be rendered for each instance of a point feature or for each attached label. When markers are tied to the label, they are rendered after the clash management process. This renders an easier-to-read map with properly placed labels.

Table 15: Uniform - Only On Labels options

Selected	Render a marker for each label generated. This takes place after the Clash Management process, meaning some labels and symbols may have been removed for ease of map viewing.
Unselected	One marker will be rendered for each geometry, regardless of the Clash Management process

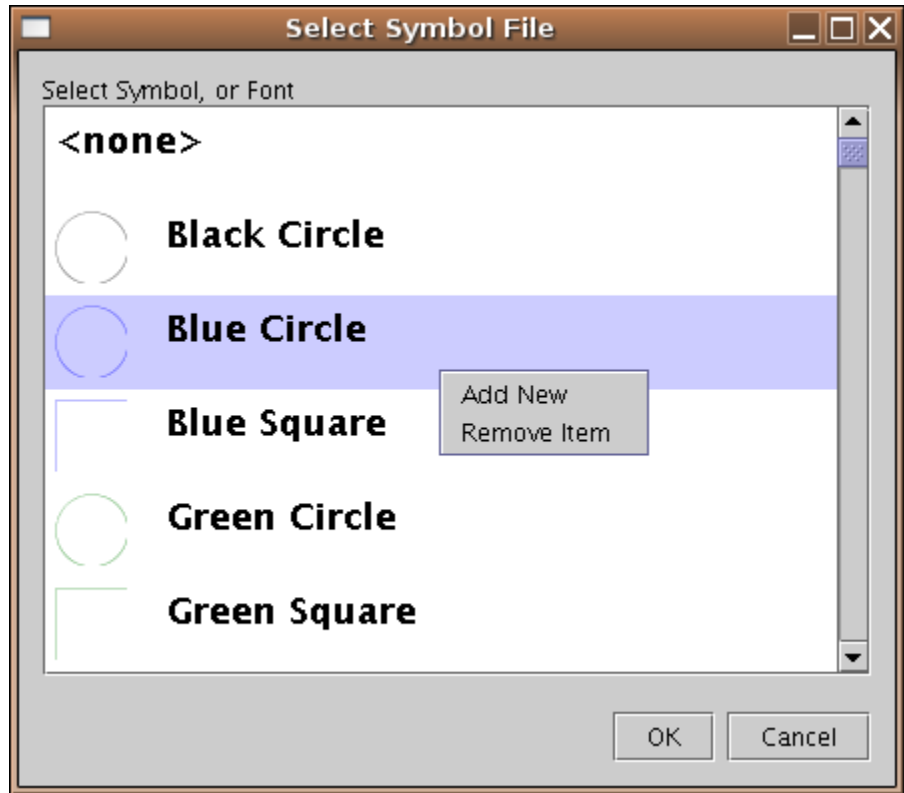


Read more information about Clash Management in the [Common Elements](#).

- **Symbol** - This option lets you choose a symbol in the drop down list of standard symbols or by selecting an SVG, PNG, GIF or TrueType font file on your local computer or mounted network disks. The chosen symbol will be resized according to the size expressed in the "size" option.

While browsing the existing symbols, you have the possibility to add your own symbols and remove the existing ones. This is achieved through the contextual menu obtained by right-clicking a symbol.

Figure 85: Uniform - Select Symbol Window



There are two kinds of symbols. Some symbols enclose the outline and fill color, while others can be customized using the "Inherits" section of this panel.

Table 16: Symbols

Blue Circle, Blue Square, Green Circle, Green Square, Red Circle, Red Square	Draws the symbol selected from the default symbol menu.
Inherited Style Circle or Square	Renders a Circle (or Square) whose colors are defined in the "Inherits" section of the Marker panel.

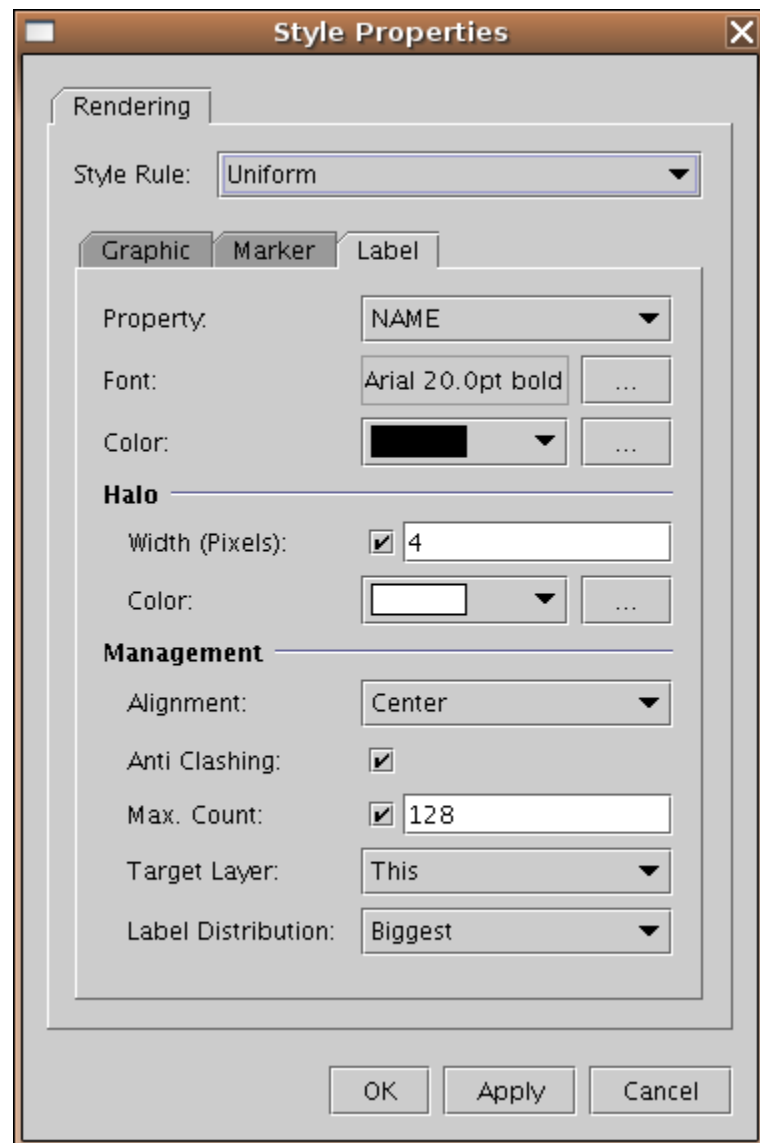
- **Inherit Section** - This section is used to define the outline and fill colors for an "Inherited Style" SVG symbol.
- When correctly encoded to allow inheritance, a SVG symbol can be modified by the ERDAS APOLLO Style Editor. This means you will be able to change its color properties for both the outline & fill of the symbol.

- **Stroke Paint** - Defines the outline color for the selected "Inherited Style" symbol. Colors can be chosen from the drop down color list or using the color palette by clicking the "...". The stroke color can be set to none by un-checking the box to the left of the colors drop down list.
- **Fill Paint** - Defines the fill color for the selected "Inherited Style" symbol. Color can be chosen from the drop down color list or using the color palette by clicking the "...". The fill color can be set to none by un-checking the box to the left of the colors drop down list.

When the chosen symbol is a TrueType font (generally a `.ttf` file), an additional panel appears to let you choose the symbol (glyph) from the list.

The Label Panel

Figure 86: Uniform - Label Panel



- **Property** - This drop down lists all properties exposed by the data source (either WFS or local shapefile) and allows you pick the one you want to use as the labeling property.
- **Font** - Allows you to choose the font to be used for text labels through a standard font chooser.



The fonts available in the font chooser are the ones installed on your local computer. Ensure the font you've chosen is available on your remote server. If the selected font is not available on the remote server, font substitution will take place and the produced map may be slightly different from the preview in the ERDAS APOLLO Style Editor.

- **Color** - Defines the text color for the label. A color can be chosen from the drop down color list or using the color palette by clicking the "... " button.
- **Halo** - You also have the option to add a "Halo" to your labels. The Halo parameter is essentially a background color for your labels. It can be used to draw attention to the labels that you have created. You can pick a desired color and width in pixels for the halo of your text labels. This functionality is useful if you want to be sure your text can be read when used in every kind of map, regardless of the color used in the other layers. For example, you may want to set a black text and a white surrounding halo. In this way your text can be read regardless of the underlying layer colors.
 - **Width** - Sets the width of the halo in pixels. You can deactivate the halo by unselecting the check box in front of the size entry box.
 - **Color** - Defines the halo color for the label. A color can be chosen from the drop down color list or using the color palette by clicking the "... " button.
- **Management** - You may choose to align your feature labels in the same location with respect to the feature being labeled. Choosing the alignment management option allows you to pick the location in which the label should be located in terms of the point feature.
 - **Alignment** - Allows you to select where to render the label relative to the point (or geometry). There are nine values for label alignment:

Table 17: Alignment options

Top Left	Top	Top Right
Left	Center	Right
Bottom Left	Bottom	Bottom Right

- **Anti Clashing** - You can choose to avoid label text that "runs-into" other label text on your display. The Anti-Clashing option activates Clash Management whose goal is to avoid label overlapping at rendering time.



Read more information about Clash Management in the "Common" chapter.

- **Max. Count** - This option sets the maximum number of labels to be kept after Clash Management. For example, if you have a layer with 1000 points, you may ask the Clash Management to:
 1. Remove any overlapping labels.
 2. Only keep a maximum number or a given number of labels (100, for example) to insure map readability.
- **Target Layer** - Here you can specify whether you want the labels to be drawn on the current layer or to be drawn on the Map Dressing layer.

Table 18: Target Layer options

This	The labels are drawn on the current layer.
Dressing	The labels are drawn on the Map Dressing layer. Using this method, the labels will be on the upper most layer for this provider. This ensures better readability of the produced map.

Styling Lines

When you have linear data, you can access appropriate styling rules by double-clicking on the line layer in the preview panel or by highlighting the layer in the panel, right-clicking, and selecting the properties item from the pull-down menu.

When styling lines, the "Uniform" styling rule offers the ability to define how to render and label the line.

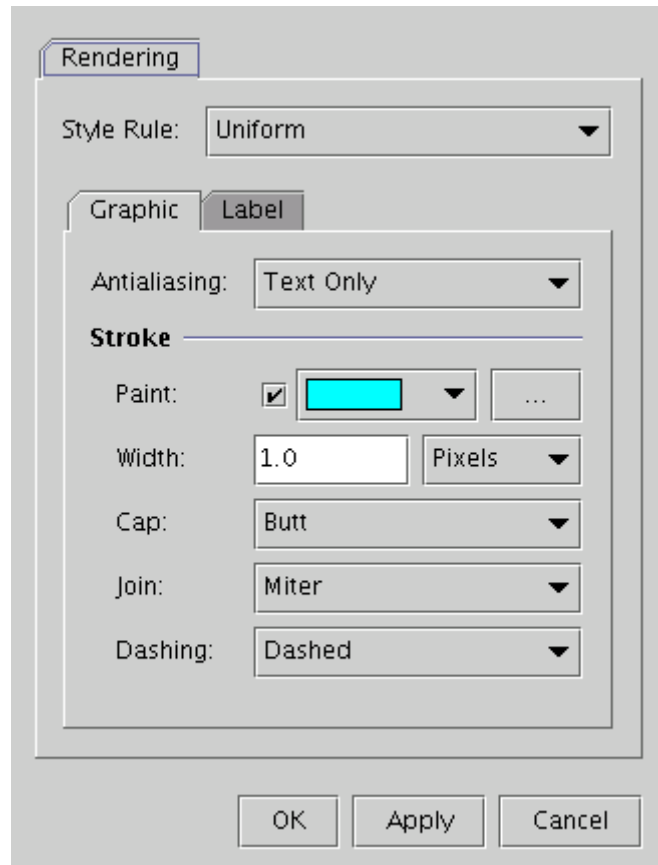
These options are grouped in two panels:

- The Graphic panel: Like the point Graphics panel, you may configure the options for styling linear geometry in terms of color, line types, end points and joining.

- The Label panel: This panel defines your labeling options. It is the same panel as for point geometries. Please refer to [The Label Panel](#) for more information.

The Graphic Panel

Figure 87: Uniform - Graphic Panel (Line Mode)



- **Antialiasing** - Activates the antialiasing option. Please refer to the Labeled Basic Style Graphic Panel explanation for points geometry for more information.
- **Stroke** - This set of options defines how the line geometry will be portrayed with options such as stroke width and stroke color.
 - **Paint** - Selects a paint color for the line geometry.
 - This color can be picked from the drop down color list or using the popup color palette by clicking on the "..." button.
 - The Paint option can be deactivated by unchecking the box next to the color drop down option.

- **Width** - Defines the line's width.
 - This value is expressed in the unit chosen from the drop down unit selector. See the table below for units explanation

Table 19: Stroke Width Units

SRS Units	The width value is a number of units in the result image SRS. These units may be degrees, meters, inches or any other linear measure depending on the SRS used. This way width is strongly tied to the image scale.
Pixels	The value will be expressed in pixels in the resulting image, regardless of its scale.
Meters	The width value will be converted into meters in the resulting image SRS. This is somewhat the same behavior as the SRS Units option.
Percents	The value is expressed in percents of the result envelope.

Please refer to the [SVG units definition](#) for more information.

- **Cap** -The ERDAS APOLLO Style Editor allows you choose the type of end caps for your linear features. End caps specify the shape of the endpoints of an open path.

Table 20: End Cap Parameters

Butt	Ends unclosed subpaths and dash segments with no added decoration. By default, end caps use this setting.
Round	Ends unclosed subpaths and dash segments with a round decoration that has a radius equal to half of the width of the line stroke.
Square	Ends unclosed subpaths and dash segments with a square projection that extends beyond the end of the segment to a distance equal to half of the line width stroke.

- **Join** - This parameter sets how you want your lines to be joined together:

Table 21: Join Parameters

Bevel	Joins path segments by connecting the outer corners of their wide outlines with a straight segment. A beveled joint will display corners as squared-off path segment corners, so that the joint appears flat rather than rounded or pointed.
Miter	Joins path segments by extending their outside edges until they meet. This is the default option. Miter joins path segments with sharp corners that extend to a single point.
Round	Joins path segments by rounding off the corner at a radius of half the line width.

- **Dashing** - There are several different dash patterns available to you:

Table 22: Dashing Patterns

Continuous	_____
Dotted
Dashed	-----
Dash Dot	-.-.-.-.-.
Dash Dot Dot	-.-.-.-.-..

Styling Polygons

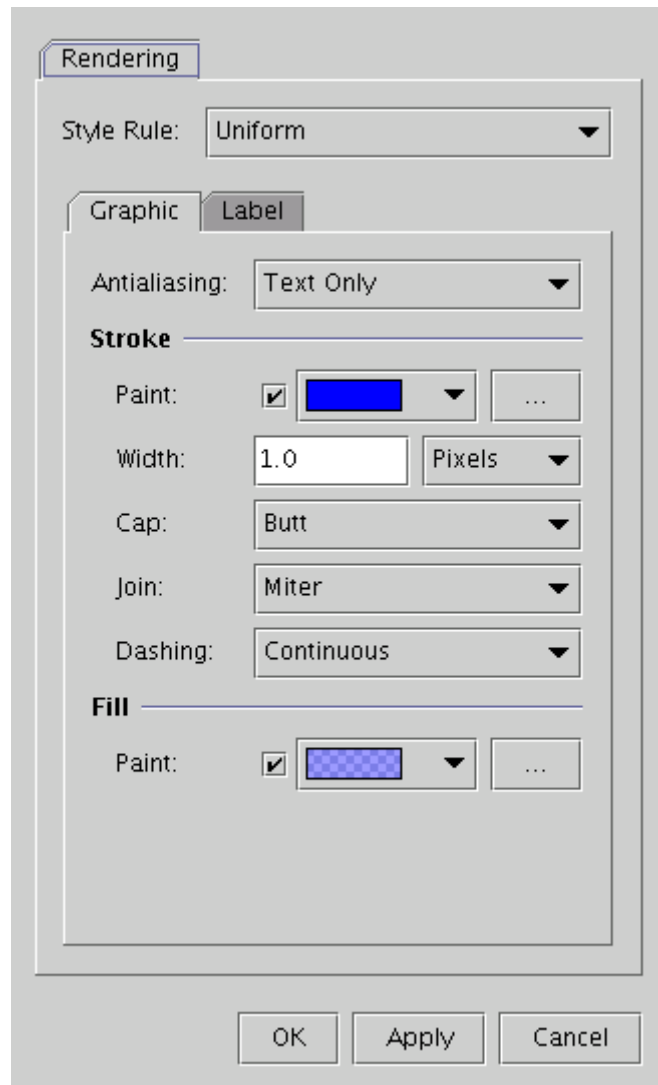
Styling polygons or polylines is basically the same as styling lines except you may define a fill color for the geometry's area.

As for line geometries, there are two sets of options:

- The Graphic panel: This panel is the same as for styling a default line except now you have the ability to define a fill color.
- The Label panel: This panel defines your labeling options.

The Graphic Panel

Figure 88: Uniform - Graphic Panel (Polygon Mode)



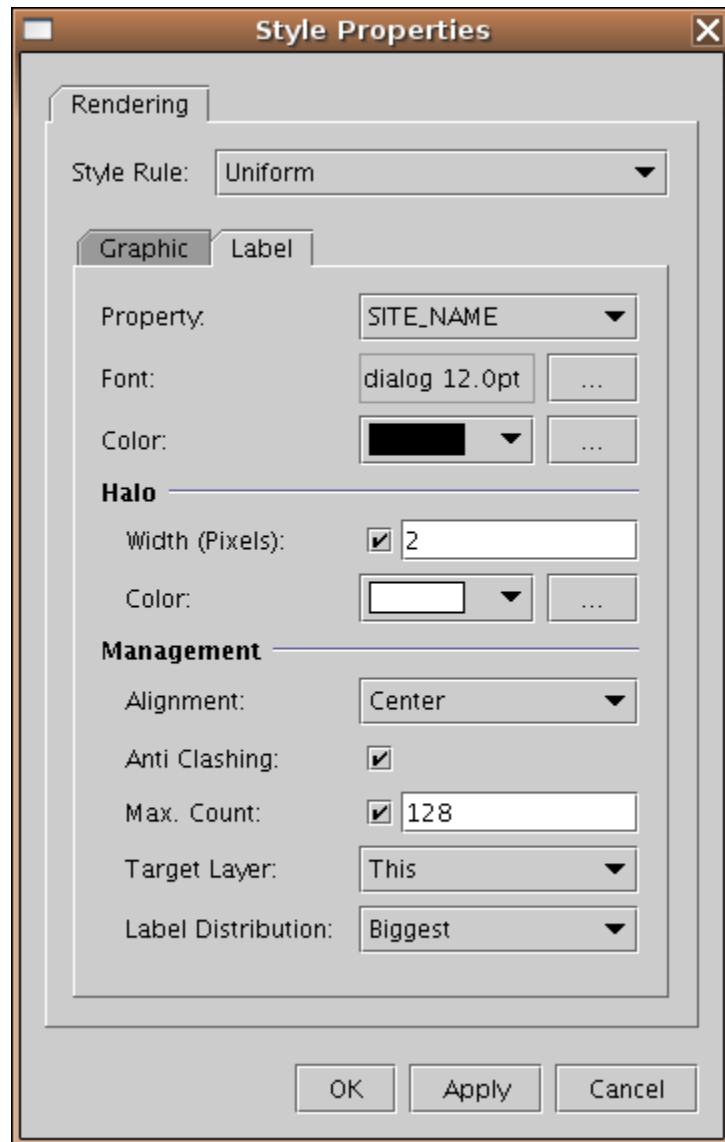
- **Antialiasing** - Activates the antialiasing option. Please refer to the Labeled Basic Style Graphic Panel explanation for points geometry for more information.
- **Stroke** - This set of options defines how the polyline or polygon outline geometry will be portrayed with options such as outline size or width.
- **Fill** - This panel section only contains one option: the paint color to use for filling the area of a polygon.
 - You can choose a color by accessing the drop down color list or by using the color palette by clicking the "..."

The Label Panel

You will find the majority of the options here present to be common with those of point geometries. This section will focus only on the polygon specific options. Please refer to [The Label Panel](#) for more information.

Labels within a polygon will always be sized to fit within the polygon. If you zoom out and the polygon is very small, the label will not display.

Figure 89: Uniform - Label Panel



- **Label distribution** - You can label multi-polygons in three ways: all polygons of the collection, the first one or the biggest one. The style panel gives you the choice, as a drop down list.

Sample Styles

Here is a sample map, composed of three layers and produced by the ERDAS APOLLO Style Editor using Labeled Basic Styles rule:

Figure 90: Uniform - Sample Styles

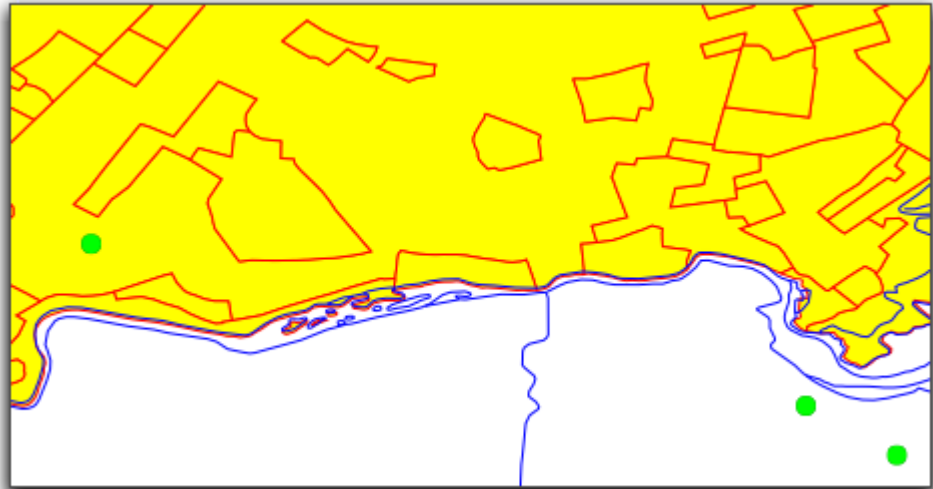


Table 23: Uniform - Sample Styles

Placename Layer	Is composed of 3 features with points geometry. They are displayed in green with a fill color. No labels have been drawn for these points.
Rivers	Are composed of lines and are rendered in blue with a 1 pixel width
Land Use	Is composed of polygons rendered in yellow with a red outline of 1 pixel width.

Classifications

Classifying data is the process of grouping your data together into classes that have similar values. One classifies data in order to make the different data in your map easier to understand and manage. You can also classify your data to discover and expose spatial patterns in your geodata that are not obvious.

The ERDAS APOLLO Style Editor includes two different methods to expose your data. The type of classification that you choose should depend on the type of data you intend to classify.

- **Discrete Classification** - If your data are categorical, such as unique names for landuse parcels, or types of roads, you should choose the discrete classification type.

- **Range Classification** - If your data are raw (such as population counts), ranked to show a progression of values (such as best to worst scholastic scores in a given region) or represents percentages (such as percentage of given area that are affected by pollution), you should use the range classification type.

You can apply classification on any valid data type: points, lines or polygons. Simply invoke the properties of the layer you wish to classify by right-clicking on the layer or double-click the layer itself.

To help you create styles for each value or range of values of your classification, we include a tool called the Populator in the ERDAS APOLLO Style Editor. This tool will create a collection of styles by allowing you to modify their parameters according to a specified color palette or color gradient. You can also edit or create additional styles manually to append additional values to your classification or even create a brand new classification.

Discrete Classification

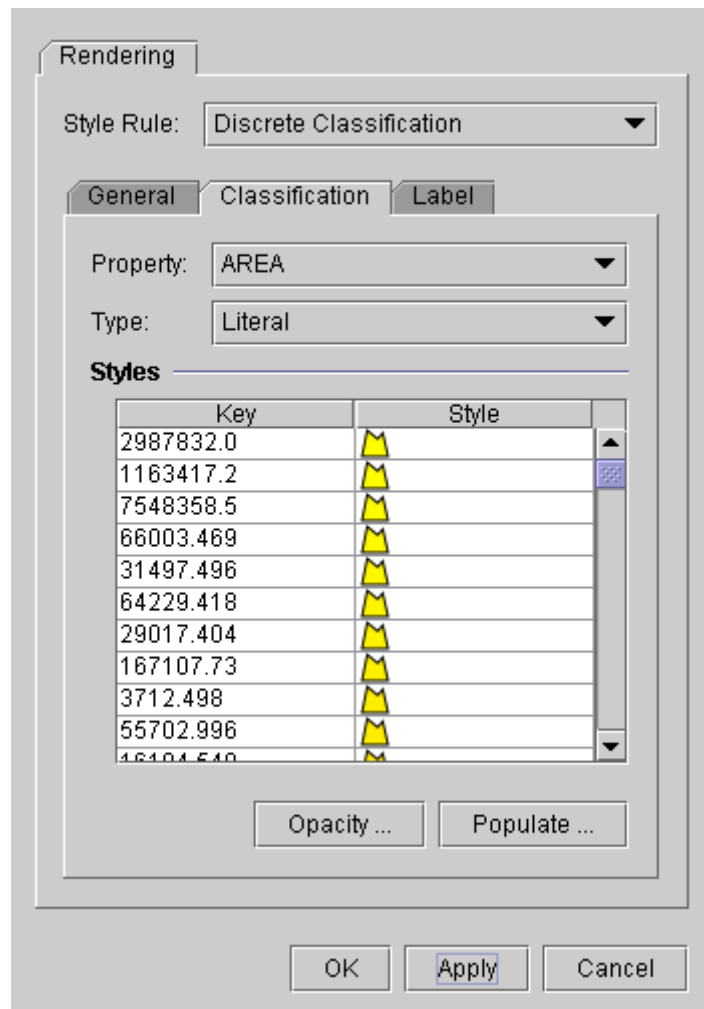
After you invoke the styling properties of a layer, pick the Discrete Classification option.

The Discrete Classification style presents three panels:

- The General panel: This panel provides a set of options to configure a display for values for your specified classes. This panel offers the same options as the "Uniform" styling rule's Graphic panel. It will change according to the geometry you wish to render. Please refer to **"Uniform" Rule** for more information about the options in the panel.
- The Classification panel: This panel allows you to build your classification's classes.
- The Label panel: This panel defines your labeling options. It is the same panel as the one for point geometries. Please refer to **The Label Panel** for more information.

The Classification Panel

Figure 91: Discrete Classification - Classification Panel



- **Property** - In order to classify your data, you must choose a field from your geographic data that contains valid information for classification. If your layer references a valid table or object database, you may click on the pull-down menu on the Property box. The pull-down menu will expose the fields in your table or object database that are valid for creating classifications for the selected layer's features.
- **Type** - Defines the type of data you build your classification upon.

Table 24: Discrete Classification - Data Types

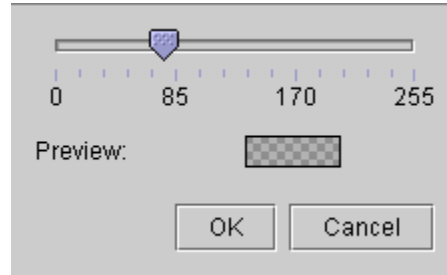
Literal	Matching is done on a character-by-character basis between the key and the string value of the data field. This is the default matching type and corresponds to the most common usage.
Integer	Matching is done on an integer basis. This may be useful when the classification keys are logically integers, but the feature data field has a real number type (values are rounded to the nearest integer before matching).
Real Number	Matching is done on a real number basis, with a limited precision of 1/1000th. This ensures that the right classification styles will be chosen when using real number data, including between different computer architectures (values are rounded a bit before matching).

- **Styles** - In this section, you will define all your classification's classes either manually or by using the Classes Populator.
- This table lists the already defined classes by their key value. There is an overview of the style associated with each class.
 - **Key:** This is the value to which the selected feature's property should be rendered as defined in the associated style.
 - **Style:** This area will display an icon previewing the associated style (mainly line and fill color).

Please read section [Creating Classes Manually](#) for information on how to create and edit classes manually or the section [Creating Classes with the Classes Populator](#) to create classes with the Classes Populator.

- **Opacity** - Displays a pop-up window to control the opacity level for the entire layer. Opacity ranges from 0 to 255 and is set once for the layer. You can also define an opacity level for one class by setting the "opacity" property of its associated style fill color.

Figure 92: Discrete Classification - Opacity



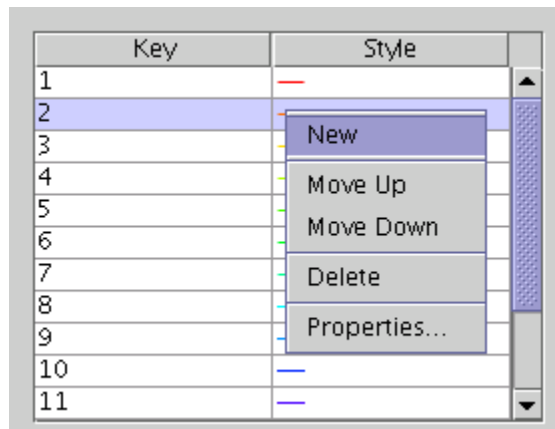
- **Populate** - Start the Classes Populator. Please read [Creating Classes with the Classes Populator](#) to learn how to use this tool.

Creating Classes Manually

Classes can be created manually. You must first create a new entry in the classes table, assign it a value and define a style for it based on the Labeled Basic Style type.

To create a new class:

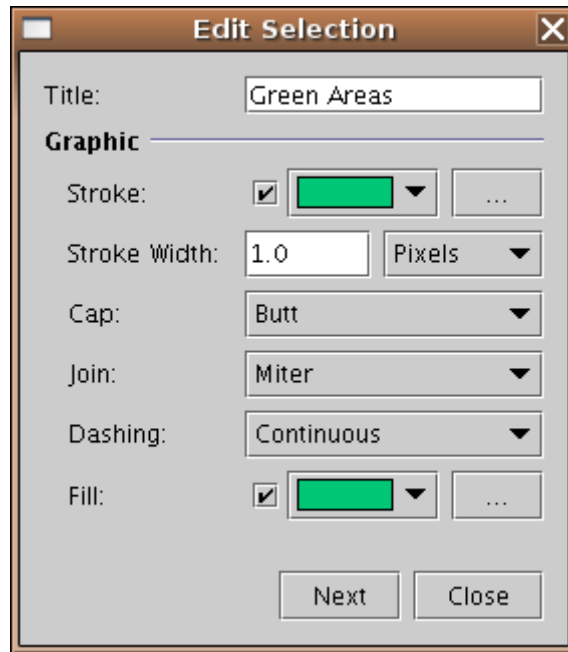
Figure 93: Discrete Classification - Styles Table



1. Right-click on any class and select "Insert". A new line is added to the Styles Table.
2. Enter a key value for your class. Right-click on this class and select "edit". The Edit pop-up window will appear.
3. Set the style options to reflect the display you want and then click on the "Close" button to close the window and save your changes. Click on the "Next" button to edit the following class (according to the Styles Table order).

Let's walk through the "Edit" window options to define a style for a specific class:

Figure 94: Discrete Classification - Styles Table



- **Title** - The title field allows you to define a title for your classification. This title will be used when displaying a legend for your classification. If no values are entered here, the legend will be generated with the key value as a title for the class.
- **Graphic** - This section's parameters define the display of the feature classification. The options are the same as for the Graphic panel of the Labeled Basic Style rule.
 - This window offers the same options as the "Uniform" rule's Graphic panel, according to the geometry you wish to render. Please refer to **"Uniform" Rule** for more information about the options in this panel.
- **Next button** - Saves your modifications and displays the styles parameters for the next class according to the order of the layers.
- **Close button** - Saves your modifications and closes the window.

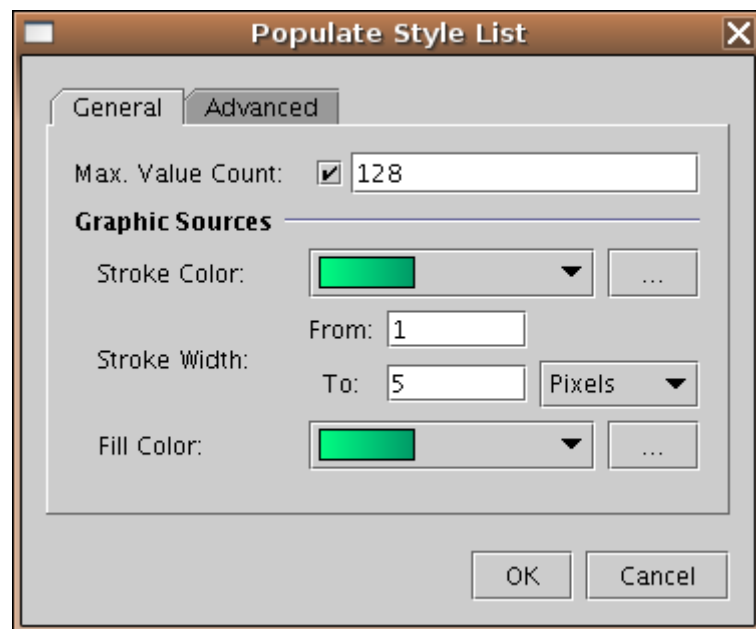
Creating Classes with the Classes Populator

The Classes Populator is a convenient tool that allows you to easily populate classifications of your data. It provides an easy way to build classifications based on a color palette, color gradient, or a two color interpolation. The tool also lets you specify a fixed number of values (or value ranges when used from a Range Classification).

Creating classes with the Populator can be achieved this way:

1. Create a new Discrete Classification rule. Open the style properties and select the "Classification" panel.
2. Select a feature property to classify upon.
3. Select a type for the feature's property.
4. Click on the "Populate" button. This will invoke the Populator window.
5. Set the graphic parameters for the style of the generated classes.
6. Click OK for the classes to be generated and added to the Styles Table.

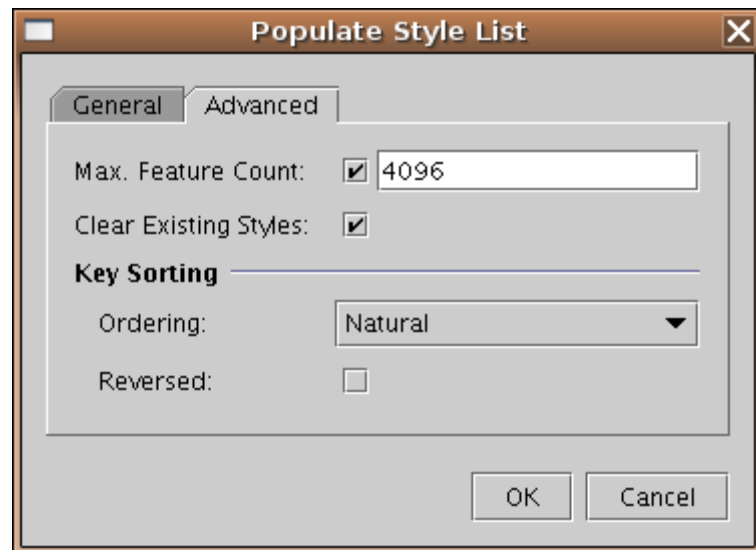
Figure 95: Classes Populator - General Panel



- **Max. Value Count** - This option defines the maximum number of different values to be kept when generating classes.

- When used with a Discrete Classification, the n -being the maximum number of values- first values coming from the features are kept whilst all other values are displayed using the default style as defined in the General panel of the Discrete Classification.
- **Graphic Sources** - These are the same options as for the Graphic panel of the Labeled Basic Style, depending on the geometry. Please refer to “**Uniform**” **Rule** for more information on these options.

Figure 96: Classes Populator - Advanced Panel



- **Max. Feature Count** - Defines the maximum number of features to be retrieved from the data source, either local or remote, to build the classes. The check box in front of the input field allows you to deactivate this feature.
 - Reducing the maximum number of features to be retrieved may significantly improve classes building time if the data source holds a large amount of features.



When used, the classes values are computed over the reduced set of data fetched. This means some values may not be taken into account when viewing the complete data set.

- **Clear Existing Styles** - This option determines whether the existing classes should be erased when generating new classifications.

- If you want to create a brand new set of classification classes, choose "True". If you want the newly generated classes to be appended to the existing ones, choose "False".
- **Key Sorting** - This section handles the generated classes order. This is important if you style your classification with the color gradient or color palette and you want the colors to reflect a data value.
 - **Ordering** - Even when doing a discrete classification, it is better to have values ordered in a logical way. Note that this setting has no effect on rendering, but may ease the editing of generated classes lists.

Table 25: Class Populator - Key Ordering

None	The classified values are not sorted. This is the fastest option.
Natural	The classified values are sorted lexicographically. This is the default option.
Numeric	The classified values are sorted in a pseudo-numeric order.

- **Reversed** - When an ordering has been set on classification keys, you may reverse the resulting list.

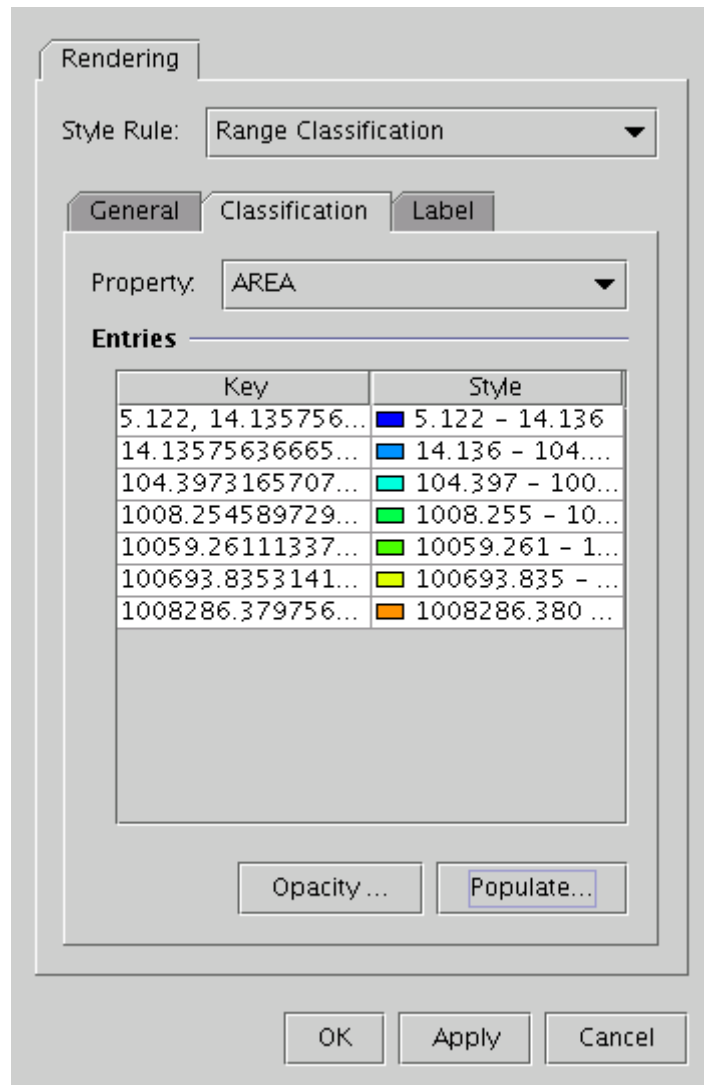
Range Classification

After you invoke the styling properties of a layer and pick the Range Classification option, the Range Classification Rule presents three panels:

- The General panel: This one is the same as for Discrete Classification. Please refer to [Discrete Classification](#) for more information.
- The Classification panel: This panel is the same as for the Discrete Classification panel except for the lack of the feature's property type selection. Range classification always assumes the property type is numeric.
- The Label panel: This panel defines your labeling options. It is the same panel as for point geometries. Please refer to [The Label Panel](#) for more information.

The Classification Panel

Figure 97: Range Classification - Classification Panel



- **Property** - Sets the feature's property upon which you build your classification. The drop down list shows the complete list of available properties for the selected layer's features.



Range classification can only be made upon numeric values. This is why you cannot select a property type. The Discrete Classification "Classification" panel allows this type of behavior.

- **Styles** - This table shows the same information and has the same behavior as for Discrete Classification.



The key values here are composed of two comma separated numeric values to represent range.

Creating Classes Manually

Classes are created the exact same way as for discrete classification. Please refer to [Creating Classes Manually](#) for Discrete Classification.

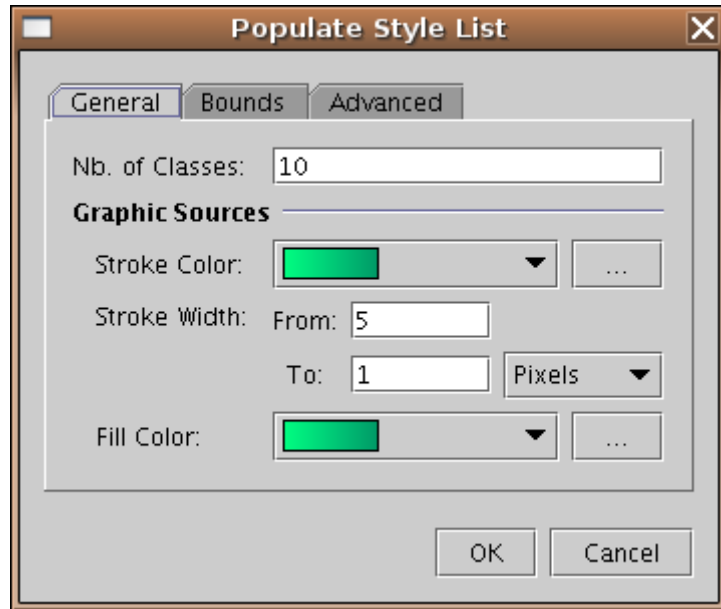
Creating Classes with the Classes Populator

The Classes Populator tool allows you to easily create all the classes of a classification. However, the panels slightly change in terms of handling numeric values as opposed to key values.

Creating classes with the Populator can be achieved this way:

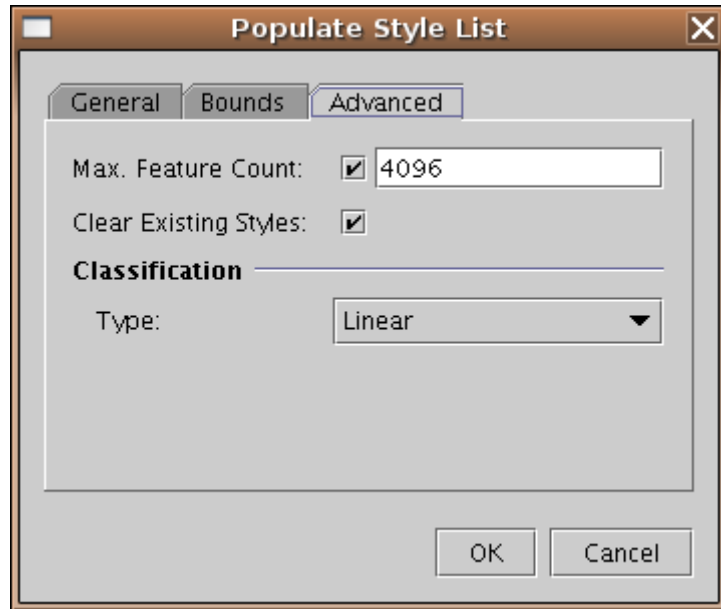
1. Create a new Range Classification rule. Open the style's properties and select the "Classification" panel.
2. Select a feature property to classify upon.
3. Select a type for the feature's property.
4. Click on the "Populate" button. This invokes the Populator pop-up window.
5. Set the parameters for the generated classes style.
6. Click OK for the classes to be generated and added to the Styles Table.

Figure 98: Classes Populator - General Panel



- **Steps** - This option defines the number of classes to be generated.
 - You can query data from the data source, check the selected property's range values and then split it in "n" sub ranges (n being the number of requested steps).
- **Graphic Sources** - This panel provides a set of options to configure a display for values for your specified classes. This panel offers the same options as the Labeled Basic Style's Graphic panel. It will change according to the geometry you wish to render. Please refer the **"Uniform" Rule** for more information about the options in the panel.

Figure 99: Classes Populator - Advanced Panel



- **Max. Feature Count** - Defines the maximum number of features to be retrieved from the data source, either local or remote, to build the classes. The check box in front of the input field allows you to deactivate this feature.
 - Reducing the maximum number of features to be retrieved may highly improve classes building time if the data source holds a large amount of features.



When used, the classes values are computed over the reduced set of data retrieved. This means some values may not be taken into account when viewing the complete data set.

- **Clear Existing Styles** - This option tells whether or not the already existing classes should be erased when generating new ones.
 - If you want to create a brand new set of classification classes, choose "True". If you want the newly generated classes to append the existing ones, choose "False".
- **Classification - Type** - You can pick from three different classification types: Linear, Logarithmic, and Quantile to compute the bounds of the intervals. You will first choose the number of classes you desire and the classification types. For example, you have a population property for 20 regions in a country. The value range is from 0 -20,000. You have chosen to display only five classes or intervals.

Table 26: Range Classification Types

Linear	Linear classification will divide the value range into five equivalent intervals. For the example, then, the first class will display values from 0-4000 and the second will display from 4000-8000. Each region that has less than a population of 4000 will be drawn using the geometry properties assigned to the first classification interval.
Logarithmic	Logarithmic classification will divide the value range in five intervals using a logarithmic progression to compute each interval range. Therefore, in the example, the first class will be 0 to 10.89 and the second will be 10.89 to 140.42. The interval sizes actually grow in an exponential way.
Quantile	In the quantile classification method, each class contains the same number of features. Population counts (as opposed to density or percentage), for example, are usually not suitable for quantile classification because only a few places are highly populated. Quantiles are best suited for data that is linearly distributed; in other words, data that does not have disproportionate numbers of features with similar values.

Sample Styles

Figure 100: Classifications - Sample Result



Table 27: Description of the Map

Protected Areas	A ranged classification with a style variation on the fill opacity (from 85 to 175).
Land Use	A discrete classification upon land usage code styled with a dedicated color palette.

“Uniform Roads” Rule

This rule is meant for rendering data representing various types of roads. Even if the parameters seem to be the same as for basic line rendering, the rendering process has been updated to enable users to portray roads and line segment junctions efficiently. The labeling contained in this rule also allows road and river specific behaviors, such as spline labeling.

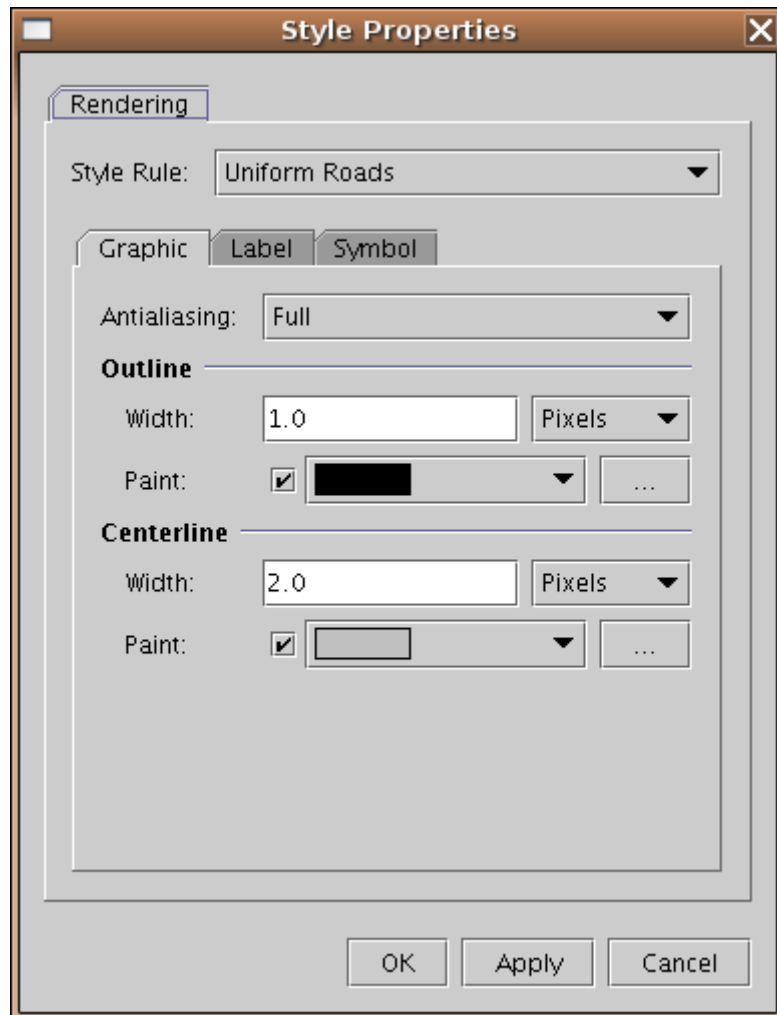
This rule can be used with line geometry only.

The rule configuration is composed of three panels:

- The Graphic panel: As with the Lines Graphics panel, you may configure the options for styling lines and center lines in terms of color, line types, end points and joining.
- The Label panel: This panel defines your labeling options. It is the same panel as for point geometries. Please refer to [The Label Panel](#) for more information.
- The Symbol panel: Here, you can apply a symbol to the feature being drawn, either from those already supplied with ERDAS APOLLO Style Editor or by importing one of your own collection.

The Graphic Panel

Figure 101: Uniform Roads - Graphic Panel

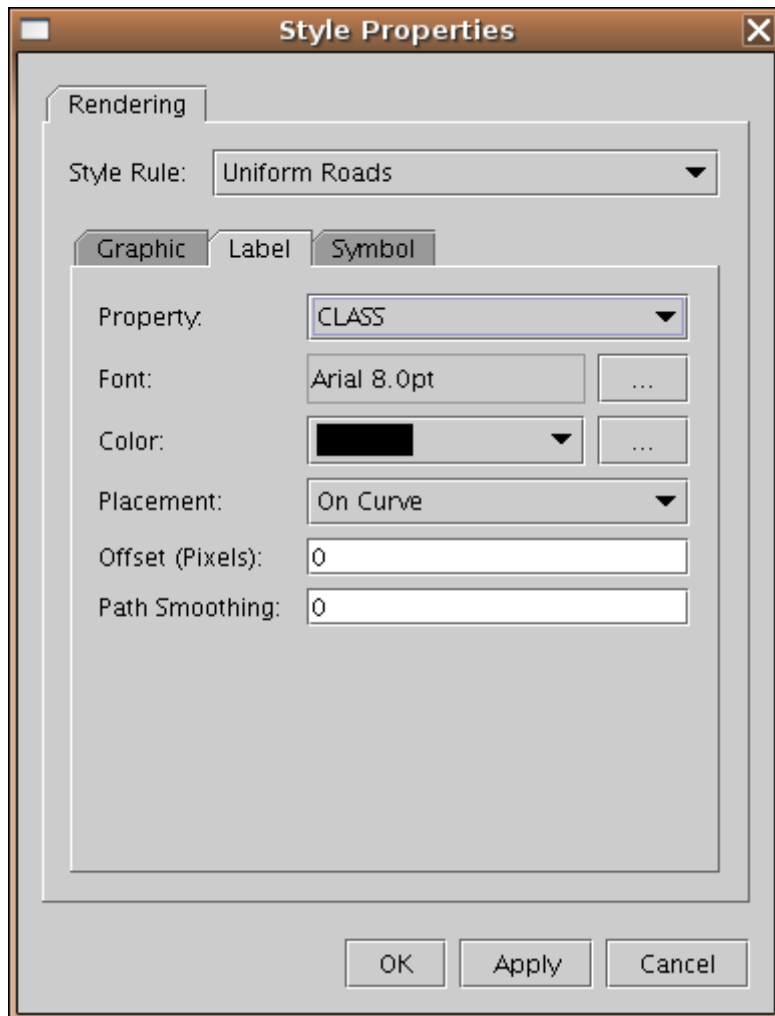


- **Antialiasing** - Activates the antialiasing option. Please refer to [The Graphic Panel](#) for more information.
- **Outline** - This group of options sets the road outline.
 - **Width** - Defines a width value for the outline. The value can be expressed in any of the units described in [Table 19:Stroke Width Units](#).
 - **Color** - Defines the outline color. Outlining can be disabled by unchecking the left control. A color can be picked from the drop-down.
- **Centerline**

- **Width** - Defines a width value for the centerline. The value can be expressed in any of the units described in [Table 19:Stroke Width Units](#).
- **Color** - Defines the outline color. The drawing of centerlines can be disabled by unchecking the left control. A color can be picked from the drop-down.

The Label Panel

Figure 102: Uniform Roads - Label Panel



- **Property** - This drop down lists all properties exposed by the data source (either from a WFS or local shapefile) and allows you pick the one you want to use as the labeling property.
- **Font** - Allows you to choose the font to be used for text labels through a standard font chooser.



The fonts available in the font chooser are the one installed on your local computer. Ensure the font you have chosen is available on your remote server. If the selected font is not available on the remote server, font substitution will take place and the produced map may be slightly different from the preview in the ERDAS APOLLO Style Editor.

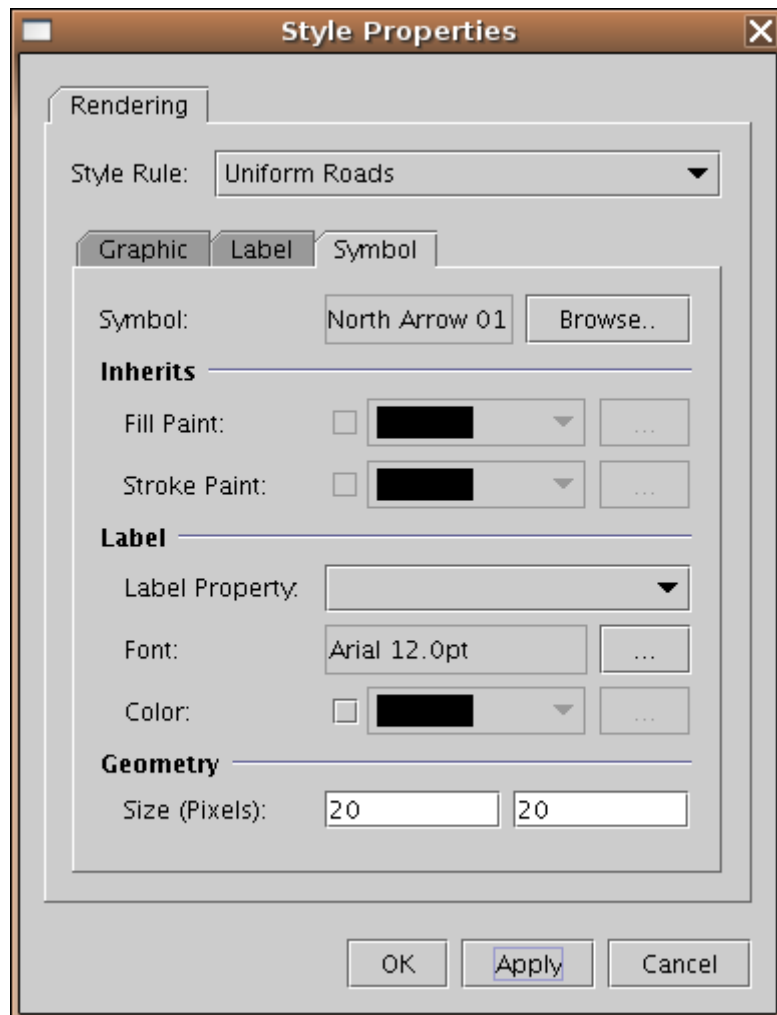
- **Color** - Defines the text color for the label. A color can be chosen from the drop down color list or by using the color palette by clicking the "... " button.
- **Placement** - You may choose to align your feature labels in the same location with respect to the feature being labeled. Choosing the alignment management option allows you to pick the location in which the label should be located in terms of the point feature.

Table 28: Placement Options

Horizontal	The labels are written horizontally at the centroid of the feature geometry.
On Curve (Stairs)	The individual characters from the text are drawn along the curve without being rotated, producing a staircase effect.
On Curve	The individual characters from the text are drawn and rotated to follow the curve.
Over Curve	The individual characters from the text are drawn and rotated to follow the curve, but are offset to "float" over the curve.
Under Curve	The individual characters from the text are drawn and rotated to follow the curve, but are offset to "hang" under the curve.

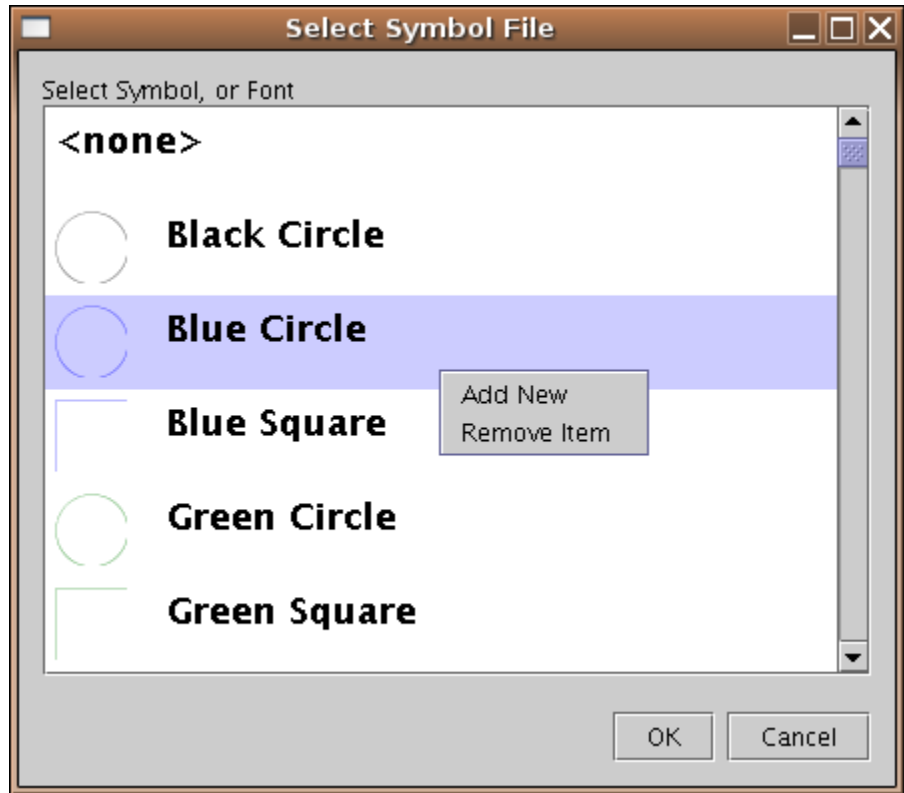
The Symbol Panel

Figure 103: Uniform Roads - Graphic Panel



- **Symbol** - Defines the symbol to be used. You can choose the symbol in the Select Symbol window accessible via the **Browse** button.

Figure 104: Uniform Roads - Select Symbol



The contextual menu obtained by right-clicking the Select Symbol window will allow you to add new items (SVG, PNG, GIF or TrueType font) and remove the existing ones.

If you choose to add a new item, a browse window will open directly in ERDAS APOLLO Style Editor's Symbols folder where you will find other useful images (e.g. in Road_Signs you will find a shield to properly label U.S. routes).

- **Inherits** - This group of options is specific to SVG symbols which allow color override.



If the selected symbol is not an SVG, or if it's an SVG which doesn't support this feature, this group will be not accessible and its options will be displayed in light grey.

- **Fill Paint** - Define the interior color of the selected symbol.
- **Stroke Paint** - Defines the outline color of the selected symbol.
- **Label**

- **Label Property** - This drop down lists all properties exposed by the data source (either from a WFS or local shapefile) and allows you pick the one you want to use as the labelling property.
 - **Font** - Allows you to choose the font to be used for text labels through a standard font chooser.
 - **Color** - Defines the text color for the label. A color can be chosen from the drop down color list or by using the color palette by clicking the "... " button.
- **Geometry**
 - **Size** - Allows you to define the size of your symbol.

Sample Styles

Figure 105: Uniform Roads - Sample Styles



Table 29: Uniform Roads - Detail on the Sample Styles

Highways	Displays a red centerline with a black outline.
Roads	Displays a gray centerline with a black outline.

Known Symbol" Rule

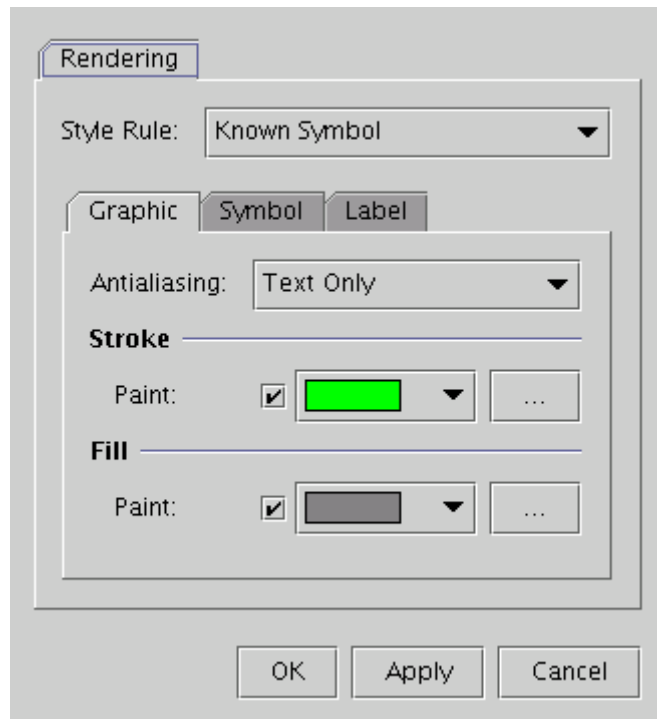
This rule is meant to display symbols quickly for selected features rather than portray their feature geometry. For performance reasons, it only allows the selection of the symbol from a fixed, predefined set of fast-to-render markers rather than allowing the selection of an arbitrary SVG, PNG or GIF file like the other rules do.

This rule can be used with any kind of geometry. The configuration is composed of three panels:

- The Graphic panel: Like the "Uniform" styling rule Graphics panel, you may configure the options for styling feature geometry in terms of color, size, and type.
- The Marker panel: You may configure the appearance of your symbol in this panel in terms of size and rotation angle.
- The Label panel: This panel allows to you control your label options. This panel is the same as for displaying point geometries. Please refer to [The Label Panel](#) for more information.

The Graphic Panel

Figure 106: Known Symbol - Graphic Panel

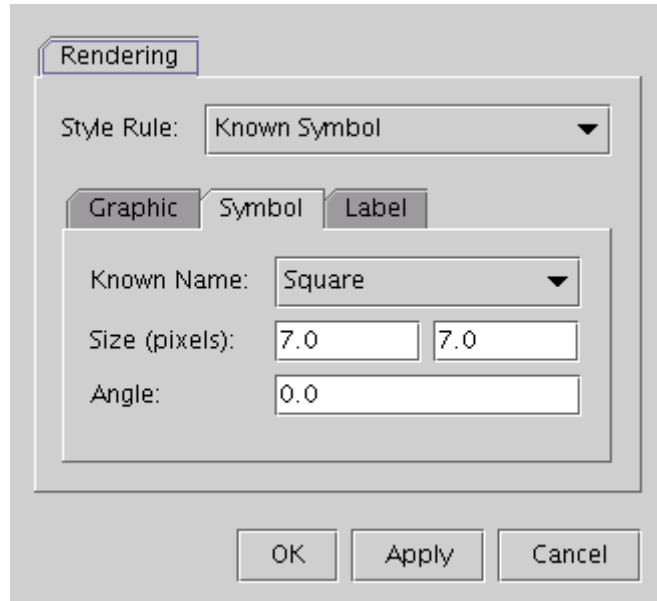


- **Antialiasing** - Activates the antialiasing option. Please refer to [The Graphic Panel](#) for more information.
- **Stroke - Paint** - This parameter defines the color to use when displaying the symbol. This parameter may be deactivated by unchecking the check box. Colors can be picked from the drop down color schema list.

- **Fill - Paint** - This parameter defines the stroke color use the render the symbol. Fill may be deactivated by un-ticking the check box. Color can be picked from the drop down color list.

The Symbol Panel

Figure 107: Known Symbol - Symbol Panel



- **Known Name** - This drop down list allows you choose one of the below-listed symbol shapes:

Table 30: Known Symbol Shapes

Square	Moon	Circle	Plus
Triangle	Pentagon	Cross	Hexagon

- **Size (pixels)** - Allows you to define the width and height of the symbols to be displayed.
- **Angle** - Allows you to set a rotation angle for the symbols to be displayed.

Sample Styles

Figure 108: Known Symbol - Sample Styles

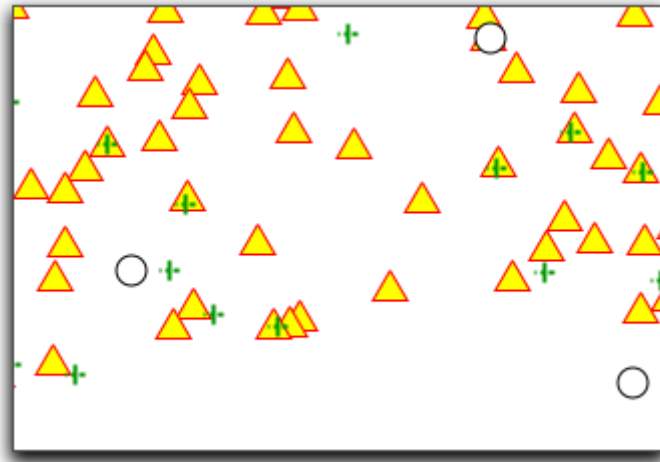


Table 31: Known Symbol - Sample Styles

Place names	White circles with a black stroke
Protected Areas	Green crosses
Land Parcels	Yellow filled triangle with a red stroke

Feature Numberer" Rule

Purpose

The "Feature Numberer" marks the features whose centroids are the nearest from the center of the map with sequential numbers. A symbol can be specified, in which case the label containing the feature number is overlaid on top of it. A maximum number of symbols/labels can be set; if reached, only the features that are the nearest from the map center get numbered and displayed.

Rule configurator

The GUI configurator of the "Feature Numberer" rule is composed of three panels:

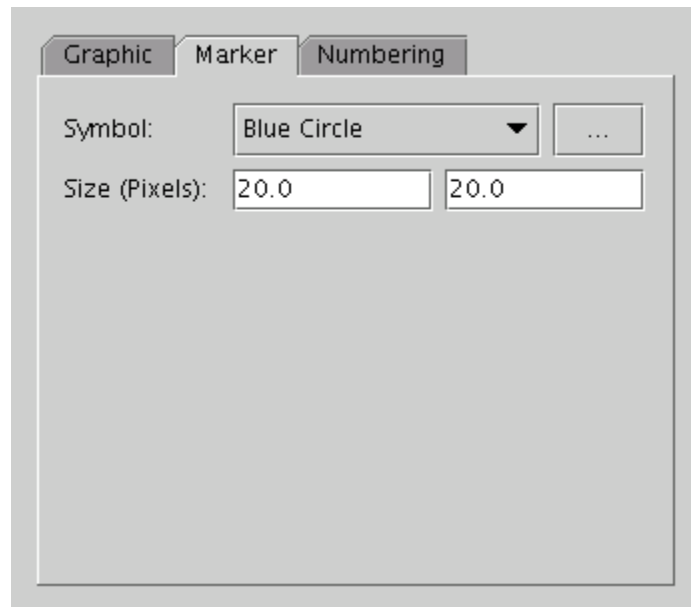
- The "Graphic" panel: Provides generic graphic options; please refer to **"Uniform" Rule** for more information.

- The "Marker" panel: Allows the selection of a symbol on which the numeric counter labels will be overlaid.
- The "Numbering" panel: Allows the configuration of the aspects of the rule that affect numbering.

The "Marker" panel

This panel allows the selection of a symbol on which the labels representing the numeric counter will be overlaid. It contains the following configurable properties:

Figure 109: Feature Numberer - Marker Panel

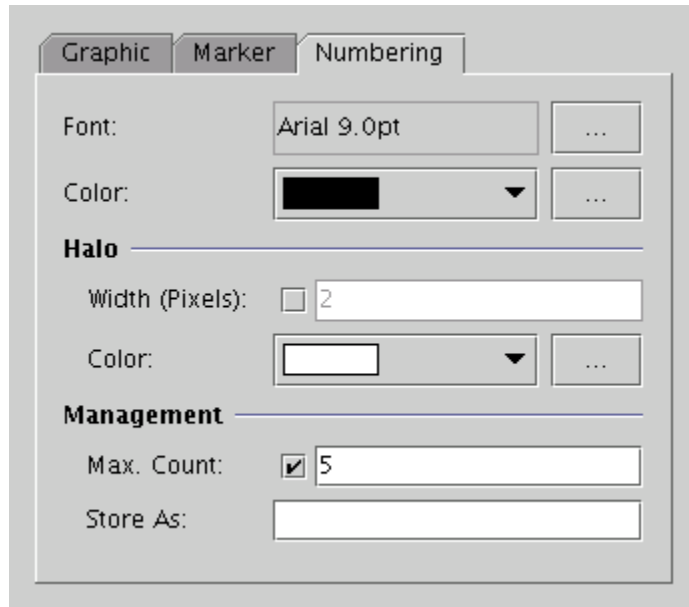


- **Symbol** - You can either select a symbol already present in the symbol library of the provider, or import a new one using the browse ("...") button.
- **Size** - The width and height (in pixels) to which the chosen symbol will be resized for display on the map. An appropriate marker size should be chosen according to the numbering font, so that the text appears "circled" by the selected symbol.

The "Numbering" panel

This panel contains the properties related to the numbering (and subsequent labeling) of features.

Figure 110: Feature Numberer - Numbering Panel



Please refer to **"Uniform" Rule** for a description of the "Font", "Color", "Halo/Width" and "Halo/Color" properties.

- **Management/Max Count** - Allows to constrain the number of numbered features to be displayed on the map; for example, only the five nearest interest points can be displayed this way. If the left-side control is left unchecked, no constraints are applied and all features are drawn.
- **Management/Store As** - This advanced property is useful only when the rule is used in conjunction with external clients; when sets, it instructs the rule to store a list of (feature, number) pairs in the rendering context for later processing.

HTML Report" Rule

This rule allows you to easily create a tabular HTML format for data styled in the ERDAS APOLLO Style Editor.

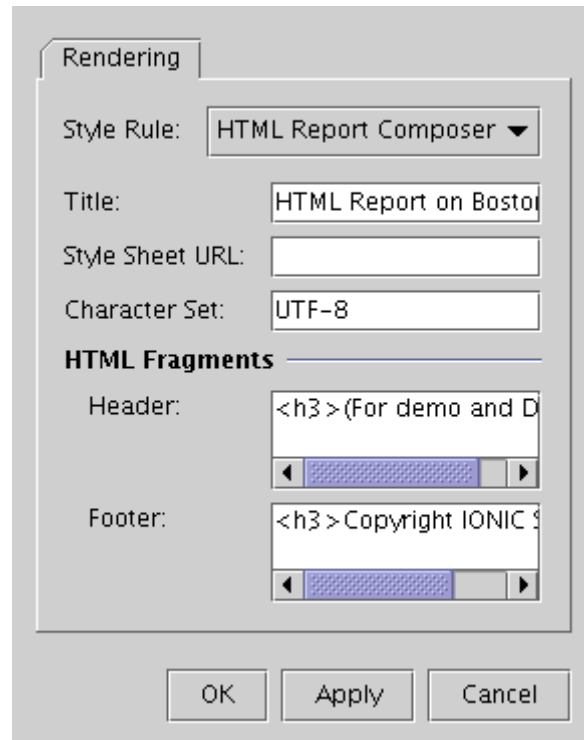
This rule can be used with any kind of geometry. There are two places in the tool where setting has to be done:

At the provider level, the "Edit Reporting Style..." menu item allows to define the report layout: title, header, footer, etc.

At the level of each feature type, creating a "HTML Report Fragment" style allows to set parameters specific to each feature type to output.

The Global Report Panel

Figure 111: HTML Report - Global Report Panel



- **Style Rule** - This drop down menu allows you to select a document template. At this release, there is only one template: the HTML Report Composer.
- **Title** - This option allows you to define a title for the generated HTML report. The text entered here will be included in the report between the <title></title> tags.
- **Style Sheet URL** - In this parameter, you can specify the URL for an external cascading style sheet (CSS). The produced HTML is composed of a known set of objects and classes and may be styled by an well formatted external CSS.

Here is a description of classes and objects used in the "HTML Report" template:

HTML CSS classes available in this document:

title: Global level 1 title.
header:Included HTML fragment.

fc-title: Feature Collection level 2 title.
fc-table: Feature Collection table.
fc-table-header: Feature Collection table header.

f-header: Feature header table row.
f-property-odd: Feature odd property table row.
f-property-even: Feature even property table row.
f-omitted-header: Omitted features header table row.

leader: Feature property row leader.
property-name: Feature property property name column.
property-value: Feature property property value column.

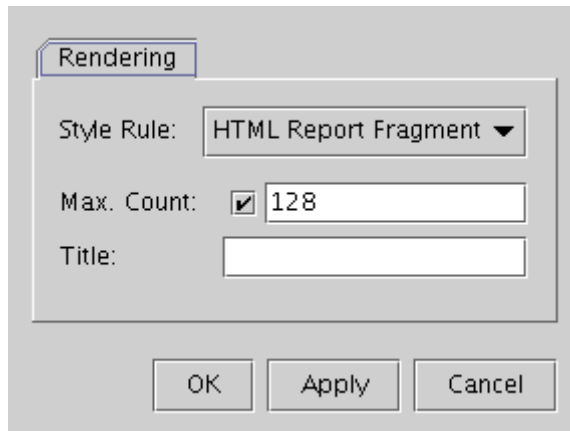
totals-title: Totals level 2 title.
totals: Computed totals text.

footer: Included HTML fragment.

- **Character Set** - Allows to define the set of characters to use. It mainly applies for non-US users or when special characters are used. Default is UTF-8.
- **HTML Fragments** - These two fields are meant to be filled with specific HTML code to be added at the top and the bottom of the generated document. This allows customization such as the ability to add your company's logo to the header or to place a disclaimer at the bottom of the page.
 - **Header** - The HTML code to be added at the top of the document.
 - **Footer** - The HTML code to be added at the bottom of the document.

Feature-specific HTML Panel

Figure 112: HTML Report - Feature Fragment Panel



- **Style Rule** - This drop down menu allows you to select a document template. At this release, there is only one template: the HTML Report Fragment.

- **Max. Count** - This option allows you to set a maximum number of features to be listed in the generated HTML report for this feature type. This option can be deactivated by un-checking the control.
- **Title** - It allows you to define a title for your feature type. If none given, the feature type name will be used.

Sample Styles

Here is a sample HTML page generated with this rule:

```
<html>
  <head>
    <title>HTML Report on Boston Shape</title>
    <style type="text/css">
      .f-header { background: #bbbbbb; font-weight: bold }
      .f-omitted-header { background: #bbbbbb }
      .f-property-even { background: white }
      .f-property-odd { background: #eeeeee }

      .leader { background: white }
      .totals { font-weight: bold }
    </style>
  </head>
  <body>
    <h1 class="title">HTML Report on Boston Shape</h1>
    <div class="header">
      <!-- start of header -->
      <h3>(For Demo and Doc purposes only)</h3>
      <!-- end of header -->
    </div>

    <h2 class="fc-title">Feature Collection (highways, 1
feature):</h2>
    <table class="fc-table" border="0" cellspacing="0"
cellpadding="3" width="100%">
      <tr class="fc-table-header">
        <td class="leader">&nbsp;&nbsp;&nbsp;</td>
        <td class="property-name">Property Name</td>
        <td class="property-value">Value</td>
      </tr>
      <tr class="f-header">
        <td colspan="3">Feature highways.6</td>
      </tr>
      <tr class="f-property-even" valign="top">
        <td class="leader"></td>
        <td class="property-name">ROUTE_</td>
        <td class="property-value">7</td>
      </tr>
      <tr class="f-property-odd" valign="top">
        <td class="leader"></td>
        <td class="property-name">ROUTE_ID</td>
        <td class="property-value">7</td>
      </tr>
      <tr class="f-property-even" valign="top">
        <td class="leader"></td>
```

```

        <td class="property-name">RT_NUMBER</td>
        <td class="property-value">28</td>
    </tr>
    <tr class="f-property-odd" valign="top">
        <td class="leader"></td>
        <td class="property-name">GEOMETRY</td>
        <td class="property-
value">MultiLineString(EP
SG:26986)
        LineString(EP
SG:26986)
        Point:(EP
SG:26986)2 ...</td>
    </tr>
</table>
<p class="totals">
    Total Feature Count: 1
</p>
<div class="footer">
    <!-- start of footer -->
    <h3>Copyright ERDAS Inc. 2009</h3>
    <!-- end of footer -->
</div>
</body>
</html>

```

Figure 113: HTML Report- Header Result

HTML Report on Boston Shape

(For Demo and Doc purposes only)

Feature Collection (highways, 1 feature):

Property Name	Value
Feature highways.6	
ROUTE_	7
ROUTE_ID	7
RT_NUMBER	28
GEOMETRY	MultiLineString(EP SG:26986) LineString(EP SG:26986) Point:(EP SG:2

Feature Collection (place_names, 1 feature):

Figure 114: HTML Report- Footer Result

TEXT_LEVEL	1
ANNO_NAME	PLACES
TEXT_ANGLE	0.0
TEXT_SIZE	70.0
TEXTSTRING	MATTAPAN
GEOMETRY	Point:(EPSG:26986)233961.25,891353.5

Total Feature Count: 2

Variable Markers" Rule

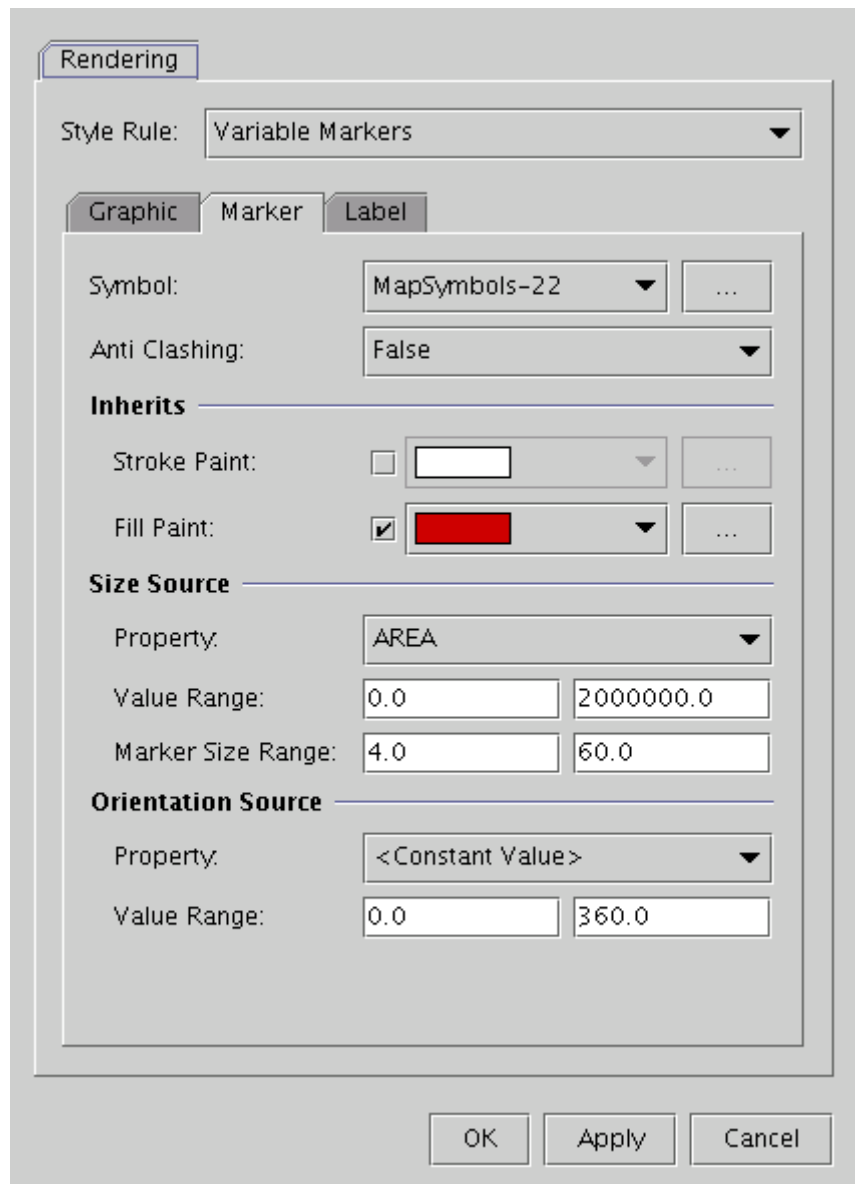
This rule is meant to pin a marker on geometries, optionally scaling and rotating it according to feature's property values. This rule is also known under the name Pinner.

The rule configuration is made of three panels:

- The Graphic panel: Where you define the type of antialiasing to use. Please refer to **"Uniform" Rule** for more information about the options available in the panel.
- The Marker panel: Where you define the marker to be used and how to configure rotation and scaling according to the feature's properties.
- The Label panel: Where you define the labeling. Please refer to **The Label Panel** for more information about the options available in the panel.

The Marker Panel

Figure 115: Variable Markers - Marker Panel



- **Symbols** - This option lets you choose a symbol either in the drop down list of standard symbols or by selecting an SVG, PNG or GIF file on your local computer or mounted network disks. The chosen symbol will be resized according to the property chosen in the "Size Source" section.

There are two kind of symbols. Some contains all the stroke and fill color within, the others can be customized using the "Inherits" section of this panel.

Table 32: Symbols

Blue Circle, Blue Square, Green Circle, Green Square, Red Circle, Red Square	Draws, according to the selected symbol, a green square or red circle, ...
Inherited Style Circle or Square	Renders a Circle (or Square) which colors are defined in the "Inherits" section of the Marker panel

- **Anti Clashing** - Enables you to activate the Clash Manager on the generated marker. The Clash Manager is in charge of making all markers visible and avoid them to overlap either by slightly moving them or group some of them in one single marker if they are to be drawn in the same area. Clash Management is made according to the map request's scale.
- **Inherits section** - This section is used to define the stroke and fill colors for an "Inherited Style" symbol.
 - **Stroke Paint** - Defines the stroke color for the selected "Inherited Style" symbol. Color can be chosen from the drop down color list or using the color palette by clicking the "...". The stroke color can be set to none by unchecking the check box left of the colors drop down list.
 - **Fill Paint** - Defines the fill color for the selected "Inherited Style" symbol. Color can be chosen from the drop down color list or using the color palette by clicking the "...". The fill color can be set to none by unchecking the check box left of the colors drop down list.
- **Size Source section** - This set of options determines how the size of the marker would be computed according to the selected feature's property, a property's value range and a marker size range. The final marker size is computed by mapping the value range to the marker size range linearly.
 - **Property** - The numeric property used when varying the size of the marker.
 - **Value Range** - The range (min., max.) of the values from the numeric property. Values outside of this range will be automatically clipped the nearest bound
 - **Marker Size Range** - The range (min., max.) of allowed sizes for the marker.

- **Orientation Source section** - This set of options determines how the orientation of the marker would be computed according to the selected feature's property and a rotation value range. The final rotation is computed by mapping the value range to the [0-360] interval linearly
 - **Property** - The numeric property used when varying the rotation of the marker.
 - **Value Range** - The range (min., max.) of the values from the numeric property. Values outside of this range will be automatically clipped the nearest bound.

Figure 116: Variable Markers - Sample Style



Patterner" Rule

This rule is meant to fill polygonal geometries with a symbol serving as a pattern. A template mechanism on the symbol file name allows you to change the symbol based on the values of feature types properties.

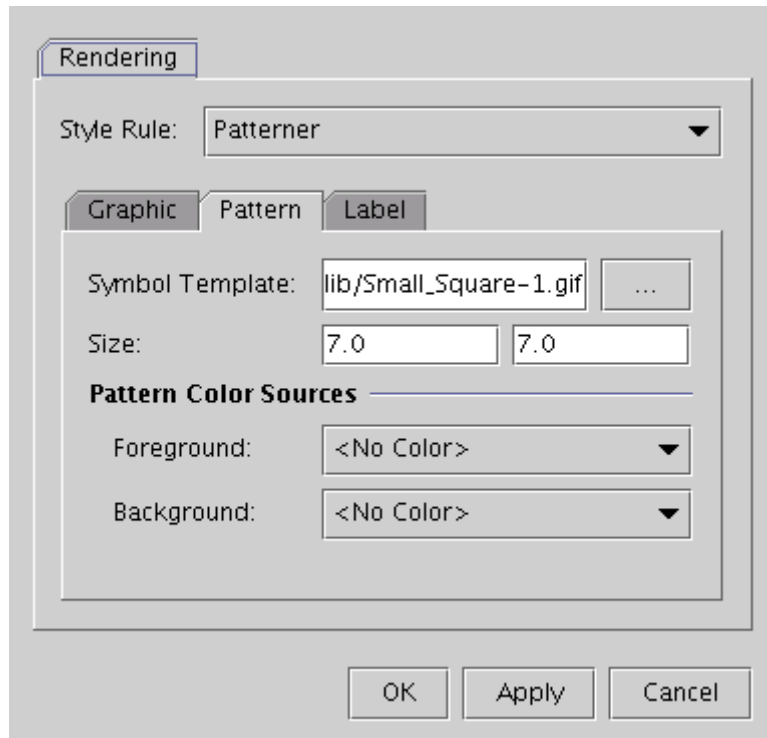
The rule configuration is made of three panels:

- The Graphic panel: Where you define the stroke and centerline color. Please refer to **"Uniform" Rule** for more information about the options available in the panel.
- The Pattern panel: Where you may define the pattern to be used.

- The Label panel: Where you define all about labeling. Please refer to [The Label Panel](#) for more information about the options available in the panel.

The Pattern Panel

Figure 117: Patterner - Pattern Panel



- **Symbol Template** - Here you may provide a link to a symbol either local or remote through an HTTP link. You may also use a relative reference to the rendering symbol directory of the portrayal engine.
- You may also enter a template evaluating to the name of a symbol in the renderer library, with portions between curlyes substituted with the corresponding feature property values, e.g.: 'lib/{TYPE}-{CODE}.png', at rendering time. Literal left curlyes can be included by doubling them.
- **Size** - The pixel-equivalent size of a pattern title
- Pattern Color Source section
 - **Foreground** - Here you may select an integer (RGB 32 bits) property of the feature whose value will be used as the pattern foreground color.

- **Background** - Here you may select an integer (RGB 32 bits) property of the feature whose value will be used as the pattern foreground color.

Symbol Roller" Rule

Purpose

The "Symbol Roller" styling rule stamps scaled/rotated symbols along curves. It obtains the symbols to use for stamping by cycling in a user-configured list.

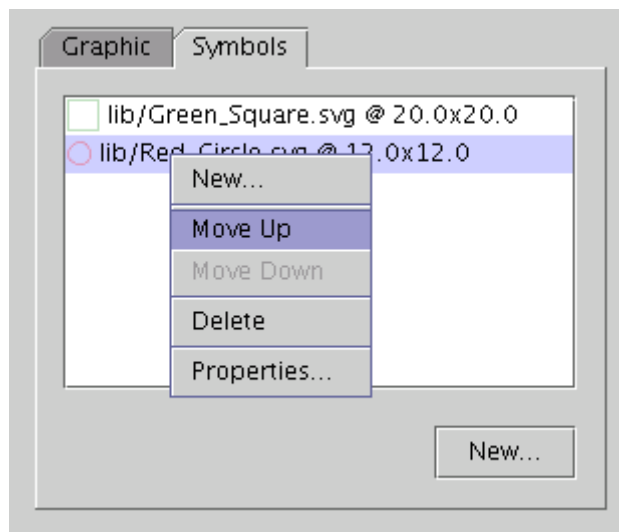
Rule configurator

The GUI configurator of the "Feature Numberer" rule is composed of two panels:

- The "Graphic" panel: Provides generic graphic options; please refer to "**Uniform" Rule** for more information.
- The "Symbols" panel: Provides the configuration mechanism for the list of symbols used for stamping.

The "Symbols" panel

Figure 118: Symbol Roller - Symbol List



This panel allows to insert, edit, delete and reorder the entries of the list of symbols used for stamping. Note that the rule cycles in that list to determine the next symbol to apply, so the order of the entries affects the rendering.

The list contained in the panel provides the following commands (which can be accessed by the contextual menu shown when the control is right-clicked):

- New... - Creates a new entry, and show the companion configuration dialog. The entry will be inserted in the list when the dialog is closed by selecting "OK".

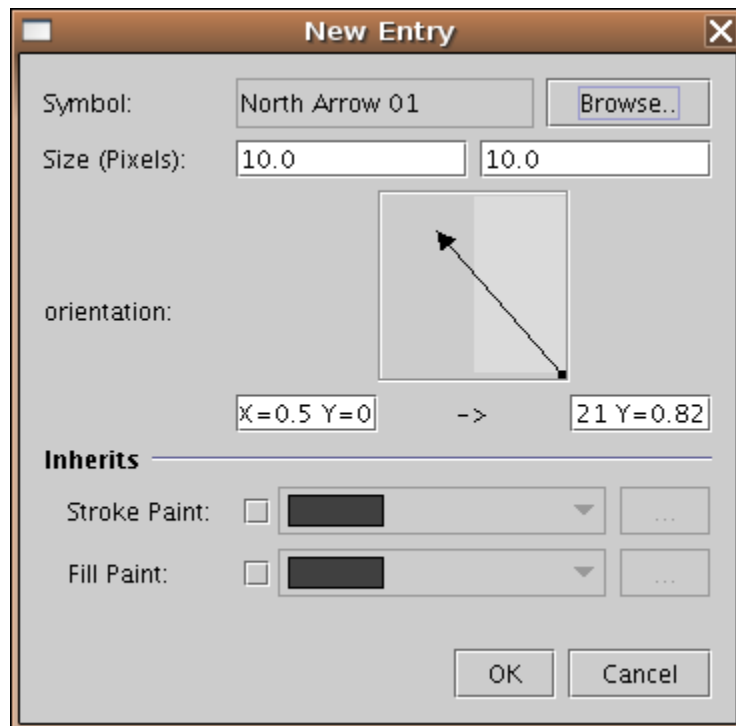
See [The entry configuration dialog](#) for more information.

- Move Up - Moves the current entry up by one position.
- Move Down - Moves the current entry down by one position.
- Delete - Removes the current entry for the list (destructive operation).
- Properties... - Shows the companion configuration dialog for the current entry. The "Next" command is a simple shortcut that allows to skip to the next entry once the current one is configured.

See [The entry configuration dialog](#) for more information.

The entry configuration dialog

Figure 119: Symbol Roller - Entry Editor



This dialog is shown when one of the "New..." or "Properties..." commands described above is invoked.

The properties presented in this dialog are mostly similar to those referred in the "Graphic" panel description in **"Uniform" Rule**. However, some attributes are rule specific, such as the symbol orientation option, in the center of the window. With this option you can change the original orientation of the symbol in order to adjust it to your specific needs, simply by changing the orientation of the arrow to the desired position.

Common Elements

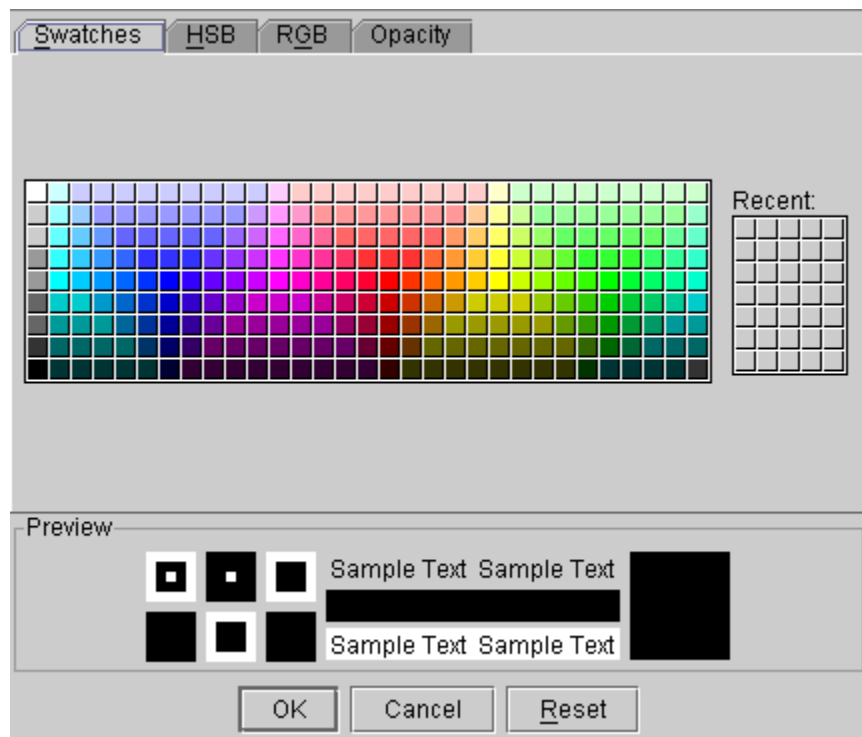
This section describe windows and palette commonly used in the style definition like color palette or font chooser.

The Color Chooser

The Color Chooser is a pop-up window accessible from anywhere in the application where a color is requested. It offers the possibility to pick one color from an RGB palette or to make a specific color with Hue-Saturation-Brightness values or Red-Green-Blue values. It also offers the ability to set an opacity level for a rendered color.

The Swatches Panel

Figure 120: The Color Chooser - Swatches Panel

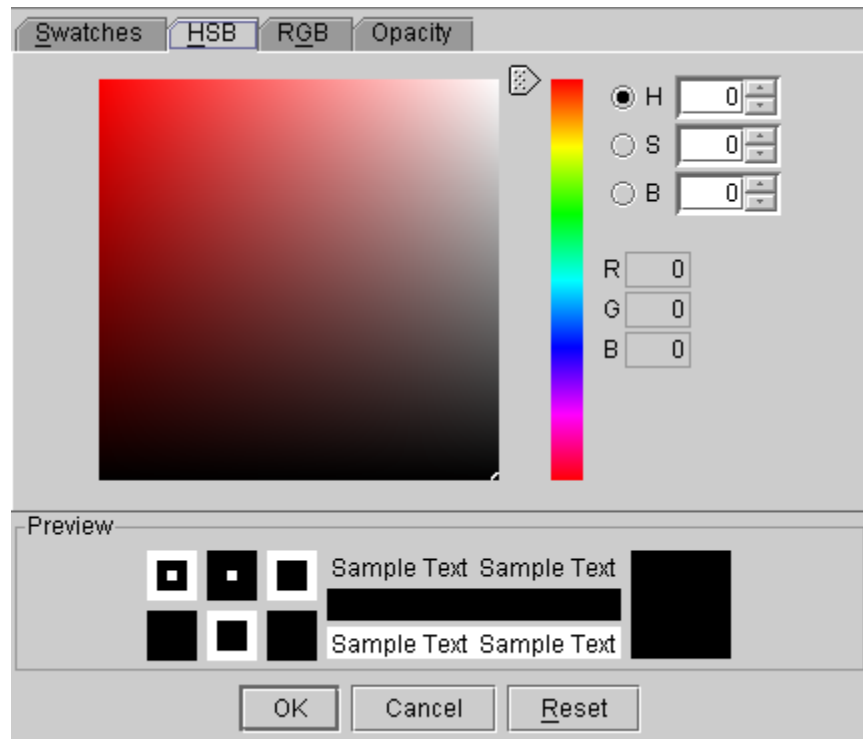


Select one of the swatches available from the complete palette or from the recently used colors (in the "recent" palette showing the recently chosen colors).

The Preview section displays a set of sample usage of the selected color for texts, areas, and reverse texts.

The HSB Panel

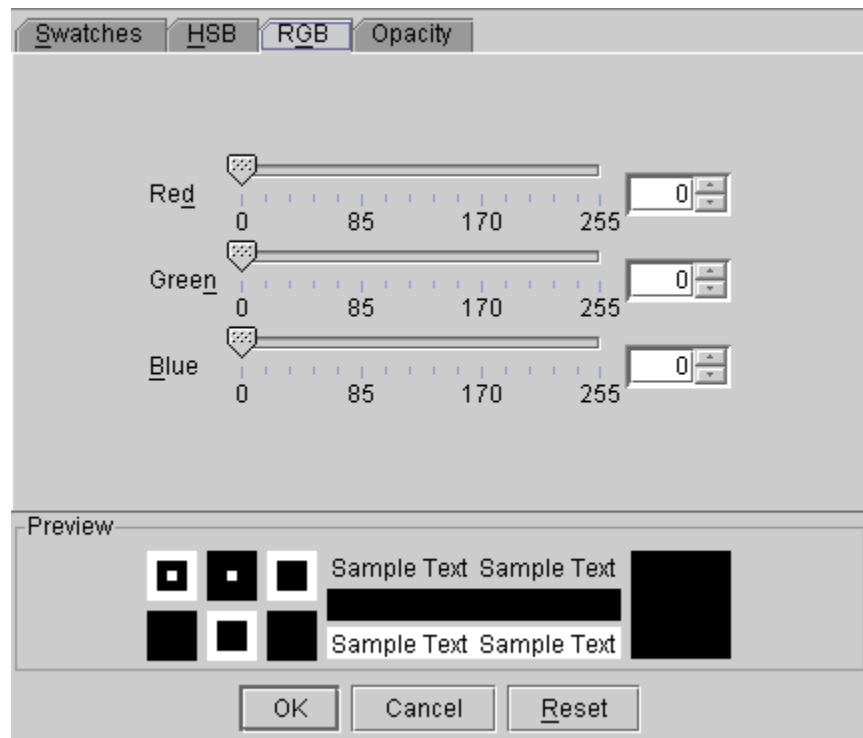
Figure 121: The Color Chooser - HSB Panel



This panel allows you to select a color by giving its hue-saturation-brightness values or selecting from within the color gradient. The panel provides, as an indication, the values of the selected color in RGB.

The RGB Panel

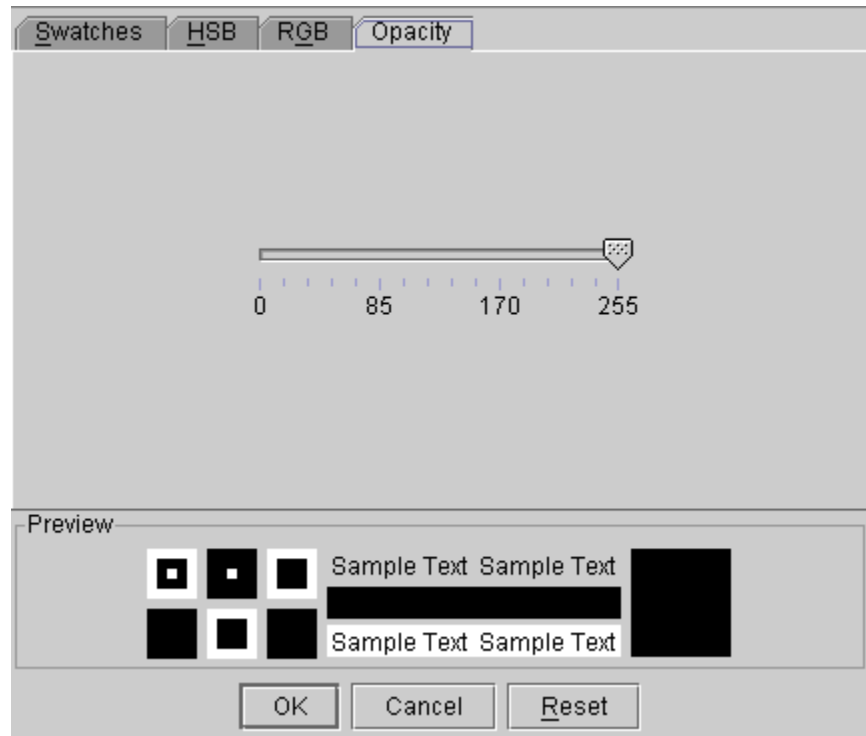
Figure 122: The Color Chooser - RGB Panel



The RGB (Red-Green-Blue) panel allows you to select a color by entering its RGB values (ranging from 0 to 255).

The Opacity Panel

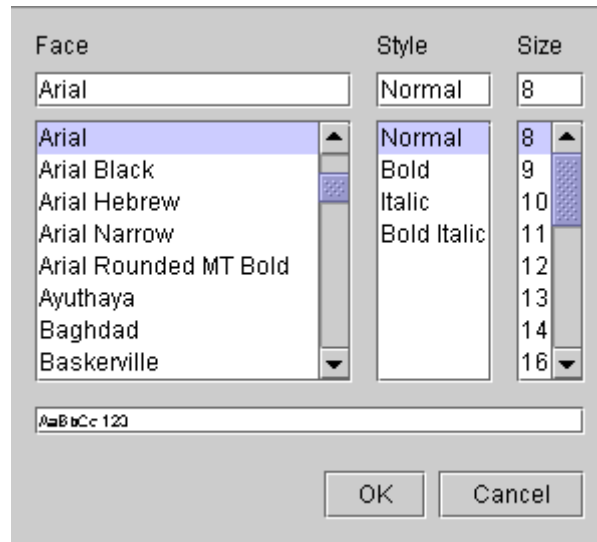
Figure 123: The Color Chooser - Opacity Panel



The opacity panel provides a way to set an opacity or transparency level to the selected color. Opacity values range from 0 to 255, 0 being totally transparent and 255 completely opaque.

The Font Selector

Figure 124: The Font Selector



The font selector is a pop-up window used each time a font has been requested as your input. It offers basic font selection behavior and allows you to select one of the fonts installed on your local machine.



Be sure to select a font which will also be available on the server where you want to deploy your styles. If the selected font is not available on the server, the server will substitute the most similar system font available.

FAQ/Troubleshooting

This chapter gives answers to various questions as well as troubleshooting techniques. It also addresses several common use cases.

FAQ

I want to send WMS requests to a Map Server but there seems to be several syntaxes.

ERDAS APOLLO Server currently supports syntaxes for OGC WMS 1.0.0, WMS 1.1.1 and WMS 1.3.0. Please refer to [The Web Map Service \(WMS\)](#) on page 573, for a description of the allowed syntaxes.

I want to send WMS requests to a Map Server, but I don't know what are the allowed parameters.

[The Web Map Service \(WMS\)](#) on page 573 describes the various requests that a WMS supports, and distinguishes between the several versions of the OGC Web Map Server specifications.

I want to send requests to a WFS but I don't know the proper syntax to use.

There are two types of HTTP requests to a WFS: HTTP-GET and HTTP-POST, and two versions of the WFS specification: WFS 1.0.0 and WFS 1.1.0. [The Web Feature Service \(WFS\)](#) on page 583 describes those alternatives.

I want to send WFS GetFeature requests to a Feature Server but I don't know the allowed parameters.

[The Web Map Service \(WMS\)](#) on page 573 describes the various requests that a WFS supports, and distinguishes between the several versions of the OGC Web Feature Server specifications.

Is SOAP supported by ERDAS web services?

Yes, ERDAS's vector (WFS), coverage (WCS) and raster (WMS) services support SOAP requests.

The SOAP requests are encapsulating the "OGC XML Stack" defining the XML syntax to be used in HTTP-POST requests. So, for example, if a WFS GetCapabilities request in HTTP-POST is sent, the request will look like:


```
<?xml version="1.0" encoding="UTF-8" ?>
<ogcwfs:GetCapabilities version="1.0.0" service="WFS"
  xmlns:ogcwfs="http://www.opengis.net/wfs" />
```

This can easily be converted into a SOAP request by adding the <Envelope> and <Body> tags around the XML statements. The SOAP request will look like:

```
<?xml version="1.0" encoding="UTF-8" ?>
<soap:Envelope
  xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/" >
  <soap:Body>
    <ogcwfs:GetCapabilities version="1.0.0" service="WFS"
      xmlns:ogcwfs="http://www.opengis.net/wfs"/>
  </soap:Body>
</soap:Envelope>
```

Is the POSTGIS package required for the POSTGRESQL connector?

No, it is supported but not mandatory. The POSTGIS package provides support for geometry collection (multi XXX) storage and for associated geometry operations (through the "GEOS" module), but ERDAS APOLLO also supports PostgreSQL without the POSTGIS extension.

I only want to give access to my vector data as maps, and not GML nor Shapefiles. Is it possible?

Yes, it is. In the Data Manager when editing the provider, **Miscellaneous** tab > **Enabled Interface** > **WMS** or **WCS/WFS**. Then all the OGC WFS request types (WFS GetCapabilities, DescribeFeatureType, GetFeature and Transaction) will be disabled and will only allow WMS GetCapabilities, GetMap, GetFeatureInfo and DescribeLayer requests. Note that the opposite can be done as well. Disable the WMS requests and allow the WFS requests.

I sent a WMS GetFeatureInfo request and I got a certain set of entities back. How can I tune this set?

In the OGC-WMS 1.1.1 specification, the GetFeatureInfo request is intended to search on the clicked point (given in pixel space). In ERDAS's implementation, the search is made in a polygon of 10x10 pixels around the clicked point. The size of this box can be changed by adding the "CAPTURE" parameter in the request. The default value is 10 pixels. It can be set to any positive integer value.

I want the ERDAS APOLLO Style Editor and the ERDAS APOLLO Administration Console to access my services through a proxy. Is it possible?

For the ERDAS APOLLO Style Editor, a desktop application, edit the `styleeditor.sh` file with a text editor and add -
Dhttp.proxyHost=MYPROXYHOST -
Dhttp.proxyPort=MYPROXYPORT just after the `java` command where `MYPROXYHOST` is the IP of the proxy and `MYPROXYPORT` is the port of the proxy.

I want my WFS-T over Oracle to run long transactions. Is there a way to keep my requests coherent while the transaction is processed?

One possible solution is for an ERDAS WFS to be configured with Oracle Workspace Manager through an SQL statement being executed at instantiation time. So, you could have two WFSes addressing the same data source, on two different workspaces. One WFS would be used for the long transaction, by duly authorized people, the second WFS being publicly accessed. That second WFS will not see any change until a merge of the workspaces is executed. The syntax to link a WFS provider with an Oracle Workspace is:

```
<LSTRING NAME="initscript" SUBST="false">  
{call dbms_wm.gotoWorkspace('MYWORKSPACE')};  
</LSTRING>
```

The OGC WFS 1.0 and 1.1 specifications have no provision for long transactions management, meaning that after that initialization phase, no other explicit action can be done on the workspace by the WFS (like activate versioning, merging or removing a workspace, ...). Such an action has to be taken independently from the WFS. OGC is currently working on WFS 1.2 and FE 1.2, and "long transactions" management is being addressed.

ScaleHint and ScaleDenominator, what does that correspond to?

`ScaleHint` suggests minimum and maximum scales for which it is appropriate to display the layer in the WMS 1.1.1 capabilities document. The property is optional and just a hint.

Definition given by OGC (OGC WMS 1.1.1 Specification, section 7.1.4.5.8): Consider a hypothetical map with a given Bounding Box, width and height. The central pixel of that map (or the pixel just to the northwest of center) will have some size, which can be expressed as the ground distance in meters of the southwest to northeast diagonal of that pixel. The two values in ScaleHint are the minimum and maximum recommended values of that diagonal.

The <MinScaleDenominator> and <MaxScaleDenominator> elements define the range of scales for which it is appropriate to generate a map of a Layer. The common pixel size is defined to be 0.28mm x 0.28mm (millimeters). This definition is given by OGC (OGC WMS 1.3.0 Specification, section 7.2.4.6.9).

How to transition from a ScaleHint value to a ScaleDenominator value? We convert from ScaleHint to ScaleDenominator by dividing the Min and Max of the ScaleHint by 0.0003959797974 (length of the diagonal of a standard pixel, i.e. 0.00028 metre) * sqrt(2).

Troubleshooting

There is some strange behavior in my GUI on Windows: For example, there is an incorrect or lack of the screen refresh.

Deactivate the hardware acceleration.

The style I have just written does not seem to produce any results. What is the problem?

First, ensure that the style is stored in the right place in one of the directory hierarchies included in the search path. Check that the feature type and style names are lowercase and that the style filename and extension have the right case. Enable the ERDAS APOLLO Portrayal Engine logs (inside the provider.fac file) and check the search sequence to ensure it matches one of the deployed styles.

If the ERDAS APOLLO Portrayal Engine successfully loads the style, check the style properties or SLD syntax as well as the exception logs.

How can I prevent Oracle 10gAS from producing an OutOfMemory error when deploying ERDAS APOLLO?

As in the other application servers, it is recommended that enough memory is provided to the JVM when running ERDAS servlets. A minimum of 256 MB is mandatory - more is better. With Oracle Application Server, an alternative is to deploy the webapps in more than one OC4J container. Allocate one container for erdas-apollo, and one for apollo-client.

Why is my output from an Indexed WCS partly empty? It looks as if all the images are not mosaicked?

By default, the WCS IndexProvider will only output, or render 500 granules to prevent long waiting time for small scale requests.

I want to use the Tomcat connection pool for my WFS to manage connections to Oracle, but I get an "oracle.sql.Number" error. Is it supported?

Yes, you can use a data pool different from the internal one to connect to Oracle, by setting the "JNDI Data Source" field of the "Data Source" tab page for your provider to "java:comp/env/jdbc/OracleDB" in the ERDAS APOLLO Data Manager. But you will also need to make sure that the proper JDBC library is used. Indeed, ERDAS provides, in the WEB-INF/lib directory of the webapp, the Oracle JDBC Thin driver. It must be removed from your classpath if you want the driver from Tomcat to be used. Simply removing ojdbc****.jar from that directory should solve the problem.

I want to define properties which are measurements, and associate them units. I configured my mapping file, but the units are ignored.

In the mapping file the association has to be declared AFTER the unit and the property are known. We recommend the <UnitAssociation> element to be set at the end of the mapping file.

When displaying arcs, they look like line strings

When the antialiasing is off, the JDK renderer tries to flatten the curve. On our side we assure that at least 200 pts are drawn, which is enough on most cases, except for very large arcs or circles.

I enable all operations on my WFS, but the "Insert" operation does not appear in the capabilities

If the "Primary" tag is not set in the mapping file for your feature type, or the <NoPrimary> tag is set, the Insert operation is disabled.

After I made changes to my configuration, clicking "Restart Service" or "Flush service cache" in the ERDAS APOLLO Data Manager does not suffice for my changes to be taken into account. Why?

Caching is the cause of the changes to be ignored: most of the time, clicking "Restart Service" link and "Flush Service Cache" in the ERDAS APOLLO Data Manager forces the reloading of the configuration files (mapping, security, ...). But the application server itself is sometimes doing some caching, so that the files are not actually reread. Restarting the application server, along with clicking "Restart Service" and "Flush Service Cache" in the ERDAS APOLLO Data Manager, will always solve the problem.

Rebuilding the Webapps

The Ant script has been built during the installation and is located at `<APOLLO_HOME>/build.xml`, in order to rebuild the webapps with their default configuration at any time.



Ant is installed along with the product, in the tools/ant folder. For automatic call of this tool, add `$APOLLO_SERVER_INSTALL/tools/ant/bin` to your PATH.

To do this, enter **ant** (if it is in your PATH) or **<APOLLO_HOME>/tools/ant/bin/ant** instead, in the prompt of `$APOLLO_SERVER_INSTALL` directory as shown below:

Example: Rebuilding Webapps using Ant

```
ant tomcat5

Buildfile: build.xml

init:

tomcat5:

configure-eas-tomcat5:

init:

configure-tomcat5:

clean:
    [echo] erdas-apollo::clean - remove temporary/intermediate
files

setup-default:
    [echo] erdas-apollo::setup-default - copy and configure
default webapp to b
uild dir
    [mkdir] Created dir: $APOLLO_SERVER_INSTALL\webapps\erdas-
apollo\build
    [copy] Warning: $APOLLO_SERVER_INSTALL\webapps\erdas-
apollo\default not found.

build-erdas-apollo:

configure-eas-full:
    [copy] Copying 825 files to
$APOLLO_SERVER_INSTALL\webapps\erdas-apollo\build
    [copy] Copied 183 empty directories to 1 empty directory
under $APOLLO_SERVER_INSTALL\webapps\erdas-apollo\build

configure-eas-light:
```

```

configure-eas:

copy-custom-providers-resources:

replace-tokens:

build-eas-tomcat5:

package:
  [delete] Deleting: $APOLLO_SERVER_INSTALL\dist\tomcat5\erdas-
  apollo.war
  [zip] Building zip:
  $APOLLO_SERVER_INSTALL\dist\tomcat5\erdas-apollo.war
  [delete] Deleting directory
  $APOLLO_SERVER_INSTALL\webapps\erdas-apollo\build

clean:
  [echo] apollo-client::clean - remove temporary/intermediate
  files

setup-default:
  [echo] apollo-client::setup-default - copy and configure
  default webapp to
  build dir
  [mkdir] Created dir: $APOLLO_SERVER_INSTALL\webapps\apollo-
  client\build
  [copy] Copying 1668 files to
  $APOLLO_SERVER_INSTALL\webapps\apollo-client\build

build-erdas-apollo:

replace-tokens:

build-eas-tomcat5:

package:
  [delete] Deleting:
  $APOLLO_SERVER_INSTALL\dist\tomcat5\apollo-client.war
  [zip] Building zip:
  $APOLLO_SERVER_INSTALL\dist\tomcat5\apollo-client.war
  [delete] Deleting directory
  $APOLLO_SERVER_INSTALL\webapps\apollo-client\build

BUILD SUCCESSFUL
Total time: 2 minutes 34 seconds

```



If you want to build the webapps for other application server, you just have to use a goal other than the default one:

- generic: generate war files for generic application servers
- jboss: generate ear files for the JBoss 4.2 application server
- tomcat5: generate war files for the Tomcat 5.5 application server

- tomcat6: generate war files for the Tomcat 6 application server
- weblogic: generate war files for the WebLogic 10.1 application server

Deploying WAR Files on Supported Servlet Engines

Web Applications are commonly packaged as a single file with the ".war" extension. During installation of your ERDAS APOLLO Server, the webapps are automatically generated and stored in the installation directory. The way to deploy one or more of those web applications in a servlet engine is specific to each of those products. This chapter gives guidelines on how to deploy your applications in some of the supported servlet engines.

As soon as something has to be modified in your web app (new license, patched library, ...), you first need to update the expanded directories in the installation directory. Then, you have to rebuild your war files and redeploy them. To rebuild the war files, see [Rebuilding the Webapps](#).

JBoss

See the deployment on JBoss in the *ERDAS APOLLO QuickStart Guide*.

Jakarta Tomcat

See the deployment on Tomcat in the *ERDAS APOLLO QuickStart Guide*.

Detailed Parameters of a Provider

Lists of Possible Parameters

The following tables list the entire set of possible <PARAM> elements that can be given to a provider. Those parameters only accept two attributes: "NAME", which is the key of the parameter, and "VALUE", which is its string value. A provider can also contain parameters with complex properties, i.e. not just a string value, but one or more sub-parameters. These complex parameters are stored in elements named <PARAMBLOCK>. They have a "NAME" attribute, and their value is a set of sub-elements which are themselves <PARAM> or <PARAMBLOCK> elements. In the tables below, those parameter blocks appear with the form "NAME (PARAMBLOCK)" and their values are a table of sub-parameters. A third type of parameter, named LSTRING, is used to give the parameter a free long string value. It has a NAME attribute, a SUBST attribute and a free-text as PCDATA. It looks like: <LSTRING NAME="initscript">a text string</LSTRING>. Its common use is to provide a string in another language (SQL, Java, ...) for execution by another engine. The SUBST attribute, if set to false, will prevent the "{...}" pattern from being substituted like a variable.

Some parameters apply to most provider types. Others are more specific to certain data sets. Others apply in the scope of only one framework (map, vector, coverage, ...). The tables below list those parameters exhaustively.

The tables are organized in such a way that you will rapidly find which parameters apply to your provider: each sub-section below applies to a given providers.fac, and in each table, the providers to which the parameter applies are checked.

The first table lists the parameters that apply to all frameworks (map, feature, coverage, terrain).



The parameter names are case-insensitive. The values are generally case-sensitive.

Parameters Common to All Types of Providers

Those parameters are understood by each provider type in each framework, and are used to enrich service metadata (capabilities) documents. The way they appear in the document vary depending on the pertaining OGC specification.

Table 33: Parameters Applying to All Providers

NAME	Detailed Meaning
ABSTRACT	Defines the service abstract (or description) in the generated service metadata (capabilities) document. Optional.
CONTACT (PARAMBLOCK)	<p>Holds a set of <PARAM> elements, which describe the "contact" information that will appear in the <Service> tag of the capabilities in the ContactInformation or ResponsibleParty sub-section.</p> <ul style="list-style-type: none"> • Organization - company name • AddressType - Postal, ... • AddressBody - street, box nr, ... • City - city name • PostCode - postal code • State - state name when applicable • Country - country name • Person - full name • Position - job title • Voice - phone nr • Fax - fax nr • Email - email address • OnlineResource - URL for more information
COPYRIGHT	Defines a text to appear in the upper left corner of each image generated by a GetMap request, and in the GML document produced by a GetFeature request. Optional.
HTTPCACHE	This parameter controls the data returned in the last-modified http header. If set to "now" (the default), the current date is returned if set to a date (ISO 8601 format), this date is returned in the last modified header. A specific date allows the Internet chain to cache the result. (use it with caution!)
KEYWORDS	A comma-separated list of keywords that will appear in the service metadata (capabilities) document. Optional.
LEGENDURL	The URL template to legend icons that will appear in LegendURL tags in the WMS service metadata (capabilities) document. Optional. A blank value uses the value of the <LEGEND> tag in the <CONFIGURATION> section.
METAURL	The URL template to metadata files that will appear in MetadataURL tags in the capabilities document. If not set, no MetadataURL will appear, but if it is set to an empty string (VALUE=""), it takes the value given in the <METADATA> tag of the <CONFIGURATION> section. Optional.

Table 33: Parameters Applying to All Providers (Continued)

OWSEXTENDEDURL	An URL (relative or not) to a file containing the ows extended capabilities of the operations metadata. The XML root should be ExtendedCapabilities elements and be a correct XML file.
OWSINFO (PARAMBLOCK)	Starting with OGC Web Services specifications versions WMS 1.3, WFS 1.1, WCS 1.1 and WTS 1.0 based on the "OWS Common" one, this parameter holds a set of <PARAM> elements, which correspond to the content of the ServiceIdentification and ServiceProvider sections of the service metadata (capabilities). Similarly to the "CONTACT" parameter, this paramblock-type property holds the following sub-elements (also described in the OGC OWS Common specification): Abstract, AdministrativeArea, City, Code, CodeSpace, Constraints, ContactHours, ContactInstructions, ContactURL, Country, DeliveryPoints, Emails, Faxes, Keyword1, Keyword2, Keyword3, KeywordCode1, KeywordCode2, KeywordCode3, KeywordCodeSpace1, KeywordCodeSpace2, KeywordCodeSpace3, Organization, PostCode, ResponsibleName, ResponsiblePosition, ResponsibleRole, ResponsibleRoleSpace.
OWSINFOURL	Starting with OGC Web Services specifications versions WMS 1.3, WFS 1.1, WCS 1.1 and WTS 1.0 based on the "OWS Common" one, we use a single template for the ServiceIdentification and ServiceProvider sections of the service metadata (capabilities). This template is a flat file referenced by this parameter. Mandatory.
REMOVE_INFO_FORMAT	The comma-separated list of internal formats to remove from the GetFeatureInfo request type capabilities and to disable when requested. The possible values are: GML, HTML, TXT, XML. Optional.
REMOVE_MAP_FORMAT	The comma-separated list of internal formats to remove from the GetMap request type capabilities and to disable when requested. The possible values are: GIF, JPEG, PNG, SVG, TIFF, WBMP, XBMP. Optional.
SECURITY	This optional parameter allows limiting access to the service to authenticated users. At run time, the user will be asked to identify before being allowed to retrieve data. Its possible values are: <ul style="list-style-type: none"> • a string, composed of a couple of blank-separated names: "name password" • "provider", security is then redirected to the underlying provider, if it supports it • a string, composed of an encrypted couple of blank-separated names, for Digest-mode authentication. This encrypted string is obtained by the tool com.ionicsoft.security.web.AuthenticatorFactory, packaged in tools/ows.
SECURITYFILTER	URL of a document. When fine-grain security is activated, this parameter allows to relocate the filter.xml file. By default, this file is in WEB-INF/classes/com/ionicsoft/security/auth, thus applying the filter to ALL providers. Optional.

Table 33: Parameters Applying to All Providers (Continued)

SECURITYRESOLVER	URL of a document or "container". When fine-grain security is activated, this parameter allows to relocate the resolver.xml file or delegate the Subject building to the underlying provider. By default, this file is in WEB-INF/classes/com/ionicsoft/security/auth, thus applying the filter to ALL providers. Optional.
SERVICE	The name of the service (as returned in the capabilities). This parameter allows mentioning the proper Service Name in the <Service> section of the capabilities document. Optional.
SRSSTRICTBEHAVIOR	This parameter allows only requests made in a SRS published in the capabilities even if it is able to do more. The default value is "false".
TITLE	This parameter allows mentioning a Service Title (or label) in the <Service> section of the capabilities document. Optional.
USESTANDARDERSRS	This parameter allows to include standard SRSes (currently EPGS:4326) in the capabilities document. The default value is "true".

Parameters for the Map Framework

The "map" servlet allows to expose as OGC-compliant WMS sets of data or non-compliant map servers. The following table lists the parameters that apply to one or more of the providers supported by this servlet. Please verify that your provider type uses a given parameter before using it.

The third column in the table below mentions the types of providers to which the parameter applies. Some of them are grouped into sets which are:

- Image: the 3 image providers (Simple, MultiSimple and Layer) publishing sets of raster data files
- Database: the 2 providers accessing databases like ArcSDE-Raster and Oracle 10g GeoRaster
- All: all providers in the map framework

Table 34: Parameters Applying to Map Providers

NAME	Detailed Meaning	Applicable Providers
AGGREGATES (PARAMBLOCK)	This block contains a set of sub-blocks, one per virtual layer to expose. See the definition of the Context provider for details.	Context provider
AJUST_BOX	If set to TRUE, forces the ImageServer to keep the image aspect ratio. Set to FALSE by default.	Image

Table 34: Parameters Applying to Map Providers (Continued)

ALLOWSEARCH	Allows a search for the first file able to be decoded if the path is a directory.	Single Image
CASCADE	Shows the list of layers to expose. It can contain a comma-separated list of layer names, the "*" sign for all layers, or an empty string "" for none.	Context provider
CASCADE_LOGGING	If set, logs the messages produced by the services invoked by the context (default is true).	Context provider
CONNECT	The connect string describing the general URL to connect to the data source. It must have the structure of the generalized data source URL: protocol://host:port/p1+v1/p2+v2, e.g., oracle://erdas/sid+erdas/user+dummy/password+dummy . However, for some types of providers, the string can differ slightly. Please refer to Provider Types for a detailed description of the connection strings. See also "defaultRowPrefetch" description below (1).	Database
DISABLED_CAPABILITIES	The comma-separated list of information to NOT include in the capabilities document. One or more of ScaleHint, Resolution.	All
FILE	Allows to explicitly mention the index file name. Defaults to PRIME_IDX . Only applies to the Image Collection Provider.	Image Collection provider only
FORWARDAUTHENTICATION	If set, the current authentication is propagated to the the services invoked by the context.	Context provider
ISAUTHORIZATIONEXCEPTION CRITICAL	Set it to true if an authorization exception (such as a request for an image at a forbidden scale) is considered critical. If set to false, the image will simply not appear; if set to true, an exception will be thrown.	Image
ISCONFORMED	If set to true, tells the framework that it can rely on the output from the provider (background color, transparency) and does not need to apply color changes which is always time consuming.	Image
LAYERABSTRACT	Descriptive text given to the layer for the WMS simple or layered image provider.	Simple Image and Image Collection providers
LAYERS	URL to the configuration file.	Map Dressing provider
LAYERTITLE	Human title given to the layer for the WMS simple or layered image provider.	Simple Image and Image Collection providers

Table 34: Parameters Applying to Map Providers (Continued)

LIMITEDCOLOR	This parameter, which value is a color (rgb, hexadecimal or named), will mention what is the single background color that the connected server can provide. It will allow the servlet to support other requested background colors.	All
LIMITEDSIZE	Some map servers can only output map with a given pixel size. If that size is given in that parameter, the servlet is able to manage requests with other sizes.	Proxy WMS provider
LIMITEDTRANSPARENCY	Some map servers sometimes do not support transparency for formats that do support it, usually GIF and PNG. This parameter, which value is the format name, allows the servlet to add transparency to non-transparent images.	Proxy WMS provider
NAME	Layer name.	Simple Image and Image Collection providers
PASSWORD	The user password used if the proxied WMS server is protected by a simple authentication schema such as basic or digest.	Proxy WMS provider
PATH	For data that are flat files, their location is given as a path.	Image

Table 34: Parameters Applying to Map Providers (Continued)

<p>PORTRAYCONFIG (PARAMMAP)</p>	<p>Defines the portrayal configuration.</p> <pre><PARAMMAP NAME="portrayconfig" DIR="/home/java/javatest/rendering2" LOADER="java,property,sld" DIRV1="/home/java/javatest/renderingv1" VERSION="2" MANAGEMENT="always" /></pre> <ul style="list-style-type: none"> • DIR - the directory where portrayal styles are searched • LOADER- -comma-separated list of portrayal language, telling the order of search of those languages. The languages are: java, property, sld. • DIRV1 - directory to search for version 1 property styles. • VERSION - styles version used in DIR. Default is 2. • MANAGEMENT - engine caching behavior for previously loaded styles. "Always" implies no caching, "none" means full caching, and "checked" means that the style timestamp is checked before re-loading. • Portray provider 	
<p>PREFERREDFORMAT</p>	<p>The preferred output format of the provider. It appears in the capabilities document.</p>	<p>All</p>
<p>PUBLISHEDSRS</p>	<p>The list of srs to appear in the service metadata (capabilities) in addition to the srs parameter value.</p>	<p>All</p>
<p>QUALITY (PARAMBLOCK)</p>	<p>This parameter allows to fix the quality of the image output by a GetMap request.</p> <pre><PARAMBLOCK NAME="quality" > <PARAM NAME="PNG" VALUE="30" /> <PARAM NAME="JPEG" VALUE="10" /> </PARAMBLOCK></pre> <ul style="list-style-type: none"> • PNG - this parameter determines whether output is an 8-bits or 24-bits image. The value ranges from 0 to 100. Less than 50 means 8-bits. The default is 70. • JPEG - this parameter determines the level of quality for JPEG output. The value ranges from 0 to 100, 0 being the maximum compression. Default is 75. • All 	

Table 34: Parameters Applying to Map Providers (Continued)

REMOVE_SRS	The list of srs to hide from the capabilities document. Used to prevent the underlying WMS from using this SRS even if requested by the client.	Proxy WMS provider
RULEDIR	Root directory of the portrayal styles.	Map Dressing provider
SORT_LAYERS	Requires the sorting of the layers by their name in the capabilities output. The default value is true, except for the PROXY provider.	All except Pyramid provider
SRS	The SRS ID of the images. When also set in the file header or database, the SRS will be overridden by the one given in the providers.fac .	Image, Database
URL	The URL of the map source to connect to.	Proxy WMS provider
USER	The user name used if the proxied WMS server is protected by a simple authentication schema such as basic or digest.	Proxy WMS provider
anonymous (PARAMBLOCK)	<p>Defines the pyramid. One PARAMBLOCK of this type per pyramid level</p> <p>1..n</p> <ul style="list-style-type: none"> • MINSCALE - minimum scale at which that provider will be called. The default value is 0. • MAXSCALE - maximum scale at which that provider will be called. The default value is Double.MAX_VALUE. • ID - name of the provider to proxy. • MASTER - Must be set to either "true" or "false". "True" signifies that the provider will export the capabilities information (service name, title and abstract, supported requests, layer list, supported SRSEs) to the pyramid. The default value is "false". It is highly recommended to have only one proxied provider labelled "master". If more than one proxied provider is set as the "master", the one that covers the lowest scale will be used. • Pyramid provider 	
DISABLED_CAPABILITIES	A list of some information disabled in the capabilities section. It is a comma separated list of values. Allowed values are "scalehint", "resolution".	All (but has only effect on the WMS interface of our WCS)

Table 34: Parameters Applying to Map Providers (Continued)

DOCLIPPING	Requires the clipping of the request box by the server box. The possible values are "always", "never" or "auto". In "auto" mode, the clipping is only done for non ERDAS WMS.	Proxy WMS provider
------------	---	--------------------

(1) defaultRowPrefetch: This additional parameter of the "connect" string parameter allows optimizing performance by allowing the framework to extract data from the data server by blocks of records. The value is the number of records to fetch together and decrease the traffic. The best gain in performance identified was when the database was not on the same machine as the Web Service. In this case, the network is a bottleneck and the defaultRowPrefetch provides a huge gain.

```
<PARAM NAME="connect"
```

```
VALUE="oracle://myhost:1521/user+boston/password+boston/SID+BOSTON/defaultRowPrefetch+1000/"
/>
```

Parameters for Vector Providers (WFS Servlet)

The third column in the table below mentions the types of providers to which the parameter applies. Some of them are grouped into sets which are:

- Database: the providers accessing databases like ArcSDE, PostgreSQL/PostGIS, Oracle Thin and Oracle OCI.
- All: all providers in the wfs framework

Table 35: Parameters Applying to Vector Data Providers

NAME	Detailed Meaning	Applicable Providers
CONNECT	The connect string describing the general URL to connect to the data source. It must have the structure of the generalized data source URL: protocol://host:port/p1+v1/p2+v2, e.g., oracle://erdas/sid+erdas/user+dummy/password+dummy . However, for some types of providers, the string can differ slightly. Please refer to Provider Types for a detailed description of the connection strings. See also "defaultRowPrefetch" description below (1). Note that the "connect" parameter is ignored if "jndidatasource" is set.	Database, WFS Proxy provider, GML
CREDENTIALNAME	The name of the credential to use to retrieve the login credential associated to the current user. This name is any URL. The login credential will be retrieved on the current user(subject) using this name and the 'database' service (see the Advanced Security chapter).	Database, WFS Proxy provider

Table 35: Parameters Applying to Vector Data Providers (Continued)

DISABLEDINTERFACE	Allow to disable an interface. The value is a comma separated list of interface names. Current interfaces are wfs and wms. This is intended to make a provider a strict wfs or a strict wms.	All
FETCHSIZE	Optional parameter that specifies the number of rows to fetch from the database. If the parameter is not used at all, 1000 rows are fetched. If it is used and its value is a positive number, the number of rows specified by the value will be fetched. If it is used and its value is set to any negative number, all of the rows will be fetched.	PostgreSQL and MS SQL Server providers
GEOMTEXTSIZE	Forces use of JDBC preparedStatements in GetMap requests to an Oracle provider. The value is the number of ordinates beyond which the SQL query is formatted with bind variables. Default value is 500. A value of 0 disables the reformatting. A value of 1 will always apply it. Only applies to the Oracle and Oracle OCI Providers.	Oracle Thin and Oracle OCI providers
INITSCRIPT (LSTRING)	<p>This property allows initialization actions on the underlying data source system. One use is an SQL statement for creation of a functional index, i.e., an index on a function, on a RDBMS like Oracle.</p> <pre><LSTRING NAME="initscript" > ALTER SESSION SET QUERY_REWRITE_ENABLED= TRUE; ALTER SESSION SET QUERY_REWRITE_INTEGRITY = TRUSTED; </LSTRING></pre> <p>The user must have the "QUERY_REWRITE" privilege on the appropriate tables. Another use is the call to the data source initialization methods for the Simple Framework provider.</p> <pre><LSTRING NAME="initscript" > call MAPINFO.LOAD_FILE('MID','C:\Erdas\data\mapinfo\boston\h ighways.MID', 'C:\Erdas\data\mapinfo\boston\highways.MIF', 'highways','EPSG:26986'); call csv.load_file('DM', 'C:\Erdas\data\csv\country.csv', 'NAME, POPULATION,GEOM','STRING,INTEGER,GEOMETRY', 'EPSG:4326',-180,-90,180,90); </LSTRING></pre> <p>A third use is to call call a procedure of the datastore, like setting a workspace on Oracle. Such a call implies to surround the statement by curly braces, and so to add the SUBST="false" to prevent those braces from being interpreted as enclosing variables.</p> <pre><LSTRING NAME="initscript" SUBST="false"> {call dbms_wm.gotoWorkspace('MYWORKSPACE')}; </LSTRING></pre>	Database, Simple Framework provider

Table 35: Parameters Applying to Vector Data Providers (Continued)

JNDIDATASOURCE	The framework can make use of a connection pool provided by the container (instead of Apollo's internal one). The parameter must contain a connection pool data source name, in the form of a string of the form: "java:comp/env/my-res-ref-name" where "my-res-ref-name" is the name defined in the JNDI InitialContext object provided by the container. For example, in BEA WebLogic 6.1, the information is found in the <res-ref-name> section in the ejb-jar.xml file. Note that this parameter has priority over the "connect" parameter.	Database
MAPPING	The URL of the mapping resource document (the description of how defined types are mapped onto the underlying data store). The value is the URL of an XML document describing the mapping between the feature type name and the entry in a SQL table. That URL can have several forms: If it is a simple file name, it is searched in the same directory as the providers.fac file. If it is a full path, the root directory is where the service is started. If the URL begins with the protocol obj:, its root directory is the package of the calling service. If the URL starts with the protocol res:, the root directory is the CLASSPATH. A "file:" protocol can also be used. (Optional).	All except the Proxy WFS provider
MAXFEATURECOUNT	This is the maximum number of features that can be returned by the WFS on one call. If maxfeaturecount=20, no more than 20 features will be returned at once. This is a way to protect the user from getting one million unwanted results on a wrong query or to protect the content of the databases by allowing a maximum 20 result in output.	All
MAXOPEN	The maximum number of connections that the pool will accept to open to the database at any one time. When this maximum is reached, the incoming connections are queued. Warning: if this maximum is not indicated, there is no maximum value. This means that the database system can be overloaded by too many connections. If the system is not really big, set the maxopen.<PARAM NAME="maxopen" VALUE="60" />	Database
MODE	For Shapefiles, this parameter defines the geometry type computing mode. Values can be "safe" (allow mixing geometries), "normal" (based on file header) or "computed" (loop on whole file).	Shapefile provider
MULTIPATH (PARAMBLOCK)	For Shapefiles, this block holds a set of 1 or more PATH parameters, each giving the location of a set of files.	Shapefile provider
NOPOSTGIS	If set to true and the PostgreSQL source is PostGIS-enabled, it uses the classical PostgreSQL data types (point, path, polygon) instead of the OGC-compliant ones (GEOMETRY, MULTIPOINT, ...).	PostgreSQL/PostGIS provider

Table 35: Parameters Applying to Vector Data Providers (Continued)

<p>OVERRIDECONNECT</p>	<p>This parameter allows you to set a connection string using a system-dependent syntax, compared to the "CONNECT" parameter using a conventional and convenient URL. When using that parameter, the usual CONNECT property only needs to be set for the USER and PASSWORD attributes. The others can be ignored or set to a fake value.</p> <p>Among others, the OVERRIDECONNECT parameter allows you to set a connection string to a load balancing system. On Oracle, it can be:</p> <pre>jdbc:oracle:thin:@(DESCRIPTION= (LOAD_BALANCE=on) (ADDRESS=(PROTOCOL=TCP)(HOST=clusnode- 1vip)(PORT=1521)) (ADDRESS=(PROTOCOL=TCP)(HOST=clusnode- 2vip)(PORT=1521)) (CONNECT_DATA=(SERVICE_NAME=myservice)))</pre>	<p>Database</p>
<p>PATH</p>	<p>For Shapefiles, their location is given as a path.</p>	<p>Shapefile provider</p>
<p>POOLSIZE</p>	<p>The number of connections to pool per user. The WFS framework contains a JDBC connection pool, so that several connections can live simultaneously. By default, the size of that pool is 10. A negative value means no pooling. Keeping some connections pooled allows to optimize the request time by limiting the number of connection open/close actions.</p>	<p>Database</p>
<p>TYPES</p>	<p>The URL of the types resource document (the list of all used types). Same kind of URL as for the "MAPPING" tag. The document will contain the definition of the feature types to publish. Note: If the "mapping" tag is absent, the "types" document is expected to contain the mapping information.</p>	<p>All except the Proxy WFS provider</p>
<p>USEPOSTGIS</p>	<p>ERDAS WFS tries to detect automatically if Postgis is installed by performing the following query: select postgis_version() from geometry_columns and checking that the result contains at least 1 row. If the result contains no row, the wfs uses postgres queries. However, it is possible to force the usage of PostGIS queries by setting this parameter to true: <PARAM NAME="usepostgis" VALUE="TRUE" /></p>	<p>PostgreSQL/PostGIS provider</p>

Parameters for the Coverage Framework

The third column in the table below mentions the types of providers to which the parameter applies. Some of them are grouped into sets which are:

- Multiple: the providers managing a set of granules, either seamless or explicit: it includes MultiSimple, Indexed and Hierarchical providers.
- All: all providers in the wcs framework

Table 36: Parameters Applying to Coverage Providers

NAME	Detailed Meaning	Applicable Providers
ALLOWSEARCH	Allows a search for the first file able to be decoded if the path is a directory.	Single Coverage
BACKGROUNDVALUE	It contains the value by band that represents the absence of data. By analogy, for images it is the background value that will be set as transparent. The value can be either a comma-separated list of values, one per band, or a single value which will apply to all bands.	All except Pyramid provider
CONNECT	The connect string describing the general URL to connect to a raster DBMS source. It must have the structure of the generalized data source URL: protocol://host:port/p1+v1/p2+v2, e.g., oracle://erdas/sid+erdas/user+dummy/password+dummy . However, for some types of providers, the string can differ slightly. Please refer to Provider Types for a detailed description of the connection strings. See also "defaultRowPrefetch" description below (1).	All except Pyramid provider
COVERAGEOFFERING (PARAMBLOCK)	Encapsulates a set of PARAMBLOCK, one per Coverage Offering for which additional information is defined 0..1 DecoderName (PARAMBLOCK) 1..n For each decoder block, a set of PARAM elements: one or more of: NAME, TITLE, ABSTRACT, KEYWORDS. All except Pyramid provider	

Table 36: Parameters Applying to Coverage Providers (Continued)

DATASTATISTICS	Specify some of the coverage metadata to improve performance of portrayal when the Normalize filter is used. A comma-separated list of four values: nb bands, data type (1 for unsigned short), min value, max value. Sample entry: <PARAM NAME="DataStatistics" VALUE="7,1,-100,16000"/>	All
DISABLED_CAPABILITIES	The comma-separated list of information to NOT include in the capabilities document. One or more of ScaleHint, Resolution.	All
EXPOSURE	SHOW_ALL: show all coverage offerings in the Capabilities document SHOW_AGGREGATES: show only aggregates nodes (non-terminal coverage offerings), this has a meaning only with Hierarchical providers (use only with Image Archive) SHOW_NOTHING: Do not show any coverage offerings in the Capabilities document Default value is SHOW_ALL	Multiple, Image Archive
GDALPATH	The value is the path to the directory where the "gdal_tool" executable lies. In ERDAS APOLLO distribution, the tool is located in <APOLLO_HOME>/tools/native/gdal.	All except Pyramid provider
HEGPATH	The path to the HEG tool installation directory. See Provider Types for more detail.	All except Pyramid provider
INDEXINGSERVER	The name of the provider, inside the providers.fac referenced in the INDEXINGPROVIDER parameter, that constitutes the indexing system. If INDEXINGTYPE is "GML", this properly holds the filtering ID given to the tiles in the indexing system.	Multiple, Image Archive
INDEXINGPROVIDER	The URL to the providers.fac file holding the definition of the indexing system, or to the GML file (if INDEXINGTYPE is set to "GML").	Multiple, Image Archive
INDEXINGTYPE	The indexing system can be either a OGC-WFS compliant Feature Server, a OGC CS-W compliant catalog or a GML file. The value "CATALOG" is needed to index in a Catalog. "WFS" is the default.	Multiple
ISAUTHORIZATIONEXCEPTION CRITICAL	Set it to true if an authorization exception (such as a request for an image at a forbidden scale) is considered critical. If set to false, the image will simply not appear; if set to true, an exception will be thrown.	Simple Coverage, Multiple, Image Archive
MAXCACHE	The maximum number of external process (GDAL or HEG) results that can be cached to answer subsequent GetCoverage or GetMap requests. If not specified, the caching is disabled.	All except Pyramid provider

Table 36: Parameters Applying to Coverage Providers (Continued)

MAXSTITCH	Defines the maximum number of Coverage granules to concatenate in the requested output. It is intended to prevent long output time for small scale requests. Default value:25. -1 means no limit. Only applies to WCS Index Providers.	All except Pyramid provider
MODE	NORMAL: value used with small test data pools, the resulting behaviour is that a WCS and WMS GetCapabilities on the service will show the CoverageOfferings defined at the service startup. The service has to be restarted to show newly indexed CoverageOfferings. DYNAMIC: value used with dynamic data pools, the resulting behaviour is that the WCS and WMS GetCapabilities on the service will show the CoverageOfferings even if they have been indexed after the service start. Default value is NORMAL	Multiple, Image Archive
NODEEXCLUSION	Defines the way a disabled aggregate behaves. The allowed values are "recursive" (the default) or "single". The "recursive" value means: if an aggregate is WMS/WCS disabled, it and its subtree will not appear in the WMS/WCS capabilities and will not be available for GetMap/GetCoverage/DescribeCoverage requests. The "single" value means: if an aggregate is WMS/WCS disabled, it will not appear in the WMS/WCS capabilities and will not be available for GetMap/GetCoverage/DescribeCoverage requests WHILE its subtree remains visible and available. It means that the subtree will be attached/linked to the parent of the disabled aggregate. The disabled aggregate will not appear in capabilities document, and so its subtree will be attached to the parent of the disabled aggregate.	Image Archive only
PATH	For data that are flat files, their location is given as a path.	All except Pyramid provider
PORTRAYALMODE	Rendering policy applied when requesting maps on an aggregate. Possible values are: <ul style="list-style-type: none"> • standard - to get the existing behaviour (default) • style - to render an aggregate as a whole only if a style or a pyramid is defined. It is not rendered as a whole if the children are homogeneous. • deep - the aggregate is never rendered as a whole. each child is rendered separately. 	Image Archive

Table 36: Parameters Applying to Coverage Providers (Continued)

<p>QUALITY (PARAMBLOCK)</p>	<p>This parameter allows to fix the quality of the image output by a GetMap request.</p> <pre><PARAMBLOCK NAME="quality" > <PARAM NAME="PNG" VALUE="30" /> <PARAM NAME="JPEG" VALUE="10" /> </PARAMBLOCK></pre> <ul style="list-style-type: none"> • PNG - this parameter determines whether output is an 8-bits or 24-bits image. The value ranges from 0 to 100. Less than 50 means 8-bits. The default is 70. • JPEG - this parameter determines the level of quality for JPEG output. The value ranges from 0 to 100, 0 being the maximum compression. Default is 75. • All 	
<p>QUERYABLES</p>	<p>path to the queryables.xml file. This file defines the additional custom queryables (search criteria) defined for this Image Archive provider.</p>	<p>Image Archive only</p>
<p>SORT_LAYERS</p>	<p>Requires the sorting of the layers by their name in the capabilities output. The default value is true, except for the PROXY provider.</p>	<p>All except Pyramid provider</p>
<p>SRS</p>	<p>The SRS ID of the coverages. When also set in the file header or database, the SRS will be overridden by the one given in the providers.fac .</p>	<p>All except Pyramid provider</p>
<p>TMPPATH</p>	<p>As soon as external binaries are used, the temporary directory given by this parameter is needed to store its output. Optional. If absent GDALPATH or HEGPATH is used.</p>	<p>All except Pyramid provider</p>
<p>VERYLARGETRIGGER</p>	<p>Allow to enforce specific management for large output sizes, as soon as the output size requested (pixel width * pixel height) is larger than VERYLARGETRIGGER*VERYLARGETRIGGER. Default value: 2500.</p>	<p>All</p>
<p>WMS_REPROJECTION_QUALITY</p>	<p>Defines the precision to use when data reprojection is required when performing WMS requests. The value is a number between 0 and 100 (default is 100). 100 means the best quality and implies the use of the standard reprojection engine which reprojects every pixel. Other values implies the use of a faster but less precise engine, the quality factor is passed to that engine.</p>	<p>All</p>

Table 36: Parameters Applying to Coverage Providers (Continued)

WCS_REPROJECTION_QUALITY	Defines the precision to use when data reprojection is required when performing WCS requests. The value is a number between 0 and 100 (default is 100). 100 means the best quality and implies the use of the standard reprojection engine which reprojects the whole coverage. Other values implies the use of a faster but less precise engine, the quality factor is passed to that engine.	All
anonymous (PARAMBLOCK)	<p>Defines the pyramid. One PARAMBLOCK of this type per pyramid level.</p> <p>1..n</p> <ul style="list-style-type: none"> • MINSCALE - minimum scale at which that provider will be called. The default value is 0. • MAXSCALE - maximum scale at which that provider will be called. The default value is Double.MAX_VALUE. • ID - name of the provider to proxy. • MASTER - Must be set to either "true" or "false". "True" signifies that the provider will export the capabilities information (service name, title and abstract, supported requests, layer list, supported SRSEs) to the pyramid. The default value is "false". It is highly recommended to have only one proxied provider labelled "master". If more than one proxied provider is set as the "master", the one that covers the lowest scale will be used. • Pyramid provider 	

(1) defaultRowPrefetch: This additional parameter of the "connect" string parameter allows optimizing performance by allowing the framework to extract data from the data server by blocks of records. The value is the number of records to fetch together and decrease the traffic. The best gain in performance identified was when the database was not on the same machine as the Web Service. In this case, the network is a bottleneck and the defaultRowPrefetch provides a huge gain.

```
<PARAM NAME="connect"
```

```
VALUE="oracle://myhost:1521/user+boston/password+boston/SID+BOSTON/defaultRowPrefetch+1000/"
/>
```

Provider Types

This appendix presents the detail of the configuration of each type of connector:

- Connector Name to use for specific types of data sources
- Specific configuration for each type of connector
- What is supported

Connectors

In this section, the types of connectors that are contained or are options of the product are described. There are three types of connectors. The first, behind the "vector" servlet, is managing vector data sources. The second, behind the "map" servlet, is handling raster data. The third is behind the "coverage" servlet and manages coverages. The table below shows each type of connector, the name of the connector and the servlet to which it belongs.



The descriptions in this chapter are for information purposes only and makes no assumption on user rights. Additional licenses may be required to use the optional connectors.

Table 37: Types of Connectors

Connectors	Name of the Connector	Servlet name
Oracle 9i,10g,11g with JDBC thin driver	com.ionicsoft.wfs.provider.oracle.OracleProvider	wfs
Oracle 9i,10g,11g with JDBC OCI driver	com.ionicsoft.wfs.provider.oracle.OracleOCIProvider	wfs
ESRI Shapefile and DBF files	com.ionicsoft.wfs.provider.shapev2.ShapeProvider	wfs
PostgreSQL Server (v 7.2.x to 8.3.x) + PostGIS (v 0.8.2 to 1.2.0)	com.ionicsoft.wfs.provider.postgresql.PostgreSQLProvider	wfs
DBF files	com.ionicsoft.wfs.provider.access.DBFProvider	wfs
Microsoft Access data	com.ionicsoft.wfs.provider.access.AccessProvider	wfs
Any other ODBC data source	com.ionicsoft.wfs.provider.access.OdbcProvider	wfs
GML 2.1.2, GML 3.1	com.ionicsoft.wfs.provider.gml.GMLProvider and GMLTransProvider	wfs

Table 37: Types of Connectors (Continued)

ArcSDE 8.x,9.x	com.ionicsoft.wfs.provider.arcsde82.ArcsdeProvider	wfs
Proxy WFS	com.ionicsoft.wfs.provider.proxy.ProxyProvider	wfs
Simple Framework	com.ionicsoft.wfs.provider.simple.SimpleProvider	wfs
DGN V7, V8 through FME	com.ionicsoft.wfs.provider.fme.FmeProvider	wfs
MS SQL Server 2008	com.ionicsoft.wfs.provider.sqlserver.SqlServerProvider	wfs
ArcSDE-RASTER 8.x, 9.x with JNI (deprecated)	com.ionicsoft.wmtmap.provider.sderaster.SdeProvider	map
ArcSDE-RASTER 9.x with Java	com.ionicsoft.wmtmap.provider.sderaster.SdeProviderJ	map
Simple Image	com.ionicsoft.wmtmap.provider.image.SimpleProvider	map
Image Collection	com.ionicsoft.wmtmap.provider.image.IndexProvider (deprecated)	map
Multiple Images	com.ionicsoft.wmtmap.provider.image.MultiSimpleProvider (deprecated)	map
Proxy WMS	com.ionicsoft.wmtmap.provider.proxy.ProxyProvider	map
Map Dressing	com.ionicsoft.wmtmap.provider.map.MapPresentationProvider	map
Pyramid WMS	com.ionicsoft.wmtmap.provider.proxy.ScaleProvider	map
Portray	com.ionicsoft.wmtmap.provider.sld.PortrayProvider	map
Context	com.ionicsoft.wmtmap.provider.cascading.ContextProvider	map
Oracle 10g Georaster	com.ionicsoft.wmtmap.provider.oracle.RasterProvider	map
Simple Coverage	com.ionicsoft.wmtmap.provider.coverage.SimpleProvider	coverage
Indexed Coverages	com.ionicsoft.wmtmap.provider.coverage.IndexProvider	coverage
Multiple Coverages	com.ionicsoft.wmtmap.provider.coverage.MultiSimpleProvider	coverage
Oracle 10g Georaster WCS	com.ionicsoft.wmtmap.provider.coverage.GeoRasterProvider	coverage
HDF-EOS Simple Coverages	com.ionicsoft.wmtmap.provider.coverage.SimpleHEGProvider (deprecated)	coverage
HDF-EOS Indexed Coverages	com.ionicsoft.wmtmap.provider.coverage.IndexHEGProvider (deprecated)	coverage
Multiple HDF-EOS Coverages	com.ionicsoft.wmtmap.provider.coverage.MultiSimpleHEGProvider (deprecated)	coverage
Pyramid WMS/WCS	com.ionicsoft.wmtmap.provider.coverage.ScaleProvider	coverage

WFS - or Vector - Connectors

Before describing the parameters for each of the vector connectors, it is useful to provide a comparative summary of what each supports. The table below summarizes the specific connector support.



Some connectors that are not described in this guide may be relative to special needs. If any particular connector is required, please contact **ERDAS Support** for the extended list of existing or desired connectors.

Table 38: WFS Providers Implementation Level

WFS Connectors	Oracle Thin	Oracle OCI	Postgre SQL/ Post-GIS	DBF and ODBC	Shapefile	MS-Access	Simple Framework	ArcSDE	GML	DGN	SQL Server 2008
WFS types:											
- Transaction	X	X	X					X	X		X
- WFS Basic	X	X	X	X	X	X	X	X	X	X	X
Mapping types:											
- Explicit	X	X	X	X	X	X	X	X	X	X	X
- SQL	X	X	X	X	X	X	X	X		X	X
- Automatic	X	X	X								
- AutoGen	X	X	X								
- Relational	X	X									
Info in DB:											
- SRS								X			X
- Bounding Box	X	X			X			X		X	X
Operators supported:											
- Geometries ¹	Basic + Multi	Basic + Multi	Basic + Multi	Point	Basic + Multi	Point	Basic + Multi	Basic + Multi	Basic + Multi	Basic + Multi	Basic + Multi

Table 38: WFS Providers Implementation Level (Continued)

- Spatial Queries ²	Rect + 2nd + 3rd (+Neighbor)	Rect + 2nd + 3rd (+Neighbor)	Rect + 2nd + 3rd	Rect	Rect	Rect	Rect	Rect + 2nd	Rect	Rect	Rect + 2nd + 3rd
--------------------------------	------------------------------	------------------------------	------------------	------	------	------	------	------------	------	------	------------------

1. Basic: Point, LineString, Polygon. Multi: MultiPoint, MultiLineString, MultiPolygon, Polygons with holes.

2. Rect: The BBOX rectangular operators. 2nd: Binary geographic operators = Intersects, Equals, Disjoint, Within, Overlaps, Crosses, Contains, Touches. 3rd: Tertiary geographic operators = Beyond, DWithin.

Oracle Connector

In order to access Oracle databases, the ERDAS WFS goes through JDBC drivers. ERDAS currently supports the Thin and OCI drivers. Note that Oracle Spatial is mandatory only when advanced GIS operations are used in the database. The JDBC **must** be the Oracle JDBC driver.

Oracle JDBC Thin Driver

As explained in the "Provider Configuration" chapter, add a <CREATE> element in which the JCLASS attribute must be: com.ionicsoft.wfs.provider.oracle.OracleProvider in the providers.fac file.

The "connect" parameter must contain a connection string of the form:

```
oracle://<host>[:<port>]/user+<username>/password+<password>/SID+<oracle_sid>
```

Where <host>, <port>, <username>, <password> and <oracle_sid> are information to locate the Oracle server and the schema to connect to it. Other parameters, such as the mapping and schema files, are common to all types of providers.



If accessing an Oracle database, use the ojdbc14.jar archive provided with that version of Oracle, or a later one (like ojdbc5.jar). The ojdbc14.jar provided with an older Oracle version is likely to fail.

Oracle JDBC OCI Driver

In the providers.fac, add a <CREATE> element in which the JCLASS attribute must be:

```
com.ionicsoft.wfs.provider.oracle.OracleOCIProvider
```

The "connect" parameter must contain a connection string of the form:

```
oracle://<host>[:<port>]/user+<username>/password+<password>/SID+<oracle_sid>
```

Where <host>, <port>, <username>, <password> and <oracle_sid> are information that locates the Oracle server as well as the schema to connect to it. Other parameters, such as the mapping and schema files, are common to all types of providers.



The Oracle OCI driver is not only composed of a jar or zip file. Several libraries also need to be installed and the related environment variables set (PATH on Windows, LD_LIBRARY_PATH or SHLIB_PATH on Unix).



In the case of an Oracle Spatial database, the value of the first SRS given in the mapping for a given feature type will be matched with the SRID given in the USER_SDO_GEOM_METADATA table.



In the case of an Oracle Spatial view, the USER_SDO_GEOM_METADATA table must be filled for the view.



The whole connection string can be encrypted in order to prevent password disclosure.

Differences between Oracle OCI and Thin Driver

The thin driver is a full Java class4 driver that can run on any platform. The OCI driver is a native JDBC driver that is specific for each platform.

The thin driver is in Java and in some case, may perform a little slower than the OCI driver. The OCI Driver, when the installed version matches the database version, is more scalable, has listeners that use less memory and may even provide caching mechanism to enhance performance.

Oracle RAC

If you need to be able to support a large number of concurrent users, you can set up the system so that the load is balanced between several Oracle servers. You can achieve this by setting up an Oracle Real Application Cluster (RAC). Oracle maintains the most thorough directions for setting up and managing an Oracle RAC on their web site: http://download.oracle.com/docs/cd/B19306_01/rac.102/b14197/to c.htm

On the provider side, in order to benefit from the RAC, the usual database connection string (host, port, sid, user, password) has to be replaced with a "JDBC Connection String" (the actual parameter is named "overrideconnect") containing all the connection options needed. In case of an Oracle cluster of four machines accessed using the Oracle thin JDBC driver, the string will look like:

```
jdbc:oracle:thin:@(DESCRIPTION= (LOAD_BALANCE=on)
  (ADDRESS= (PROTOCOL=TCP) (HOST=clusnode-1vip) (PORT=1521))
  (ADDRESS= (PROTOCOL=TCP) (HOST=clusnode-2vip) (PORT=1521))
  (ADDRESS= (PROTOCOL=TCP) (HOST=clusnode-3vip) (PORT=1521))
  (ADDRESS= (PROTOCOL=TCP) (HOST=clusnode-4vip) (PORT=1521))
  (CONNECT_DATA= (SERVICE_NAME=myhost)))
```

Note that a classical Oracle server can also be accessed using a similar configuration instead of the usual one. For example, for a connection string set with host=myoraclehost, port=myoracleport, sid=myoraclesid, user=myuser, password=myspassword, the following JDBC Connection String is equivalent:

```
jdbc:oracle:thin:@(DESCRIPTION= (ADDRESS_LIST= (ADDRESS
  = (PROTOCOL=TCP) (HOST=myoraclehost) (PORT=myoracleport))) (CONNECT_DATA= (SID=myoracle
  sid)))
```

THE USER AND PASSWORD have to be set in the usual Connection String property but all the other parameters in that property can be left blank.

PostgreSQL/PostGIS

This provider applies to PostgreSQL data sources (versions 7.2 to 8.3 have been tested) as well as to PostGIS (version 0.8.2 to 1.2.0 have been tested) along with GEOSS 2.2.3. Depending on the data source and the geometry types of the spatial data, it must be initialized differently.

Globally, to set up a PostgreSQL or a PostGIS connector, in the providers.fac file, add a <CREATE> element in which the JCLASS attribute must be:

```
com.ionicsoft.wfs.provider.postgresql.PostgreSQLProvider.
```

The "connect" parameter must contain a connection string of the form:

```
postgres://<host>[:<port>]/user+<username>[/password+<password>
]/database+<dbname>
```

Where <host>, <port>, <username>, <password> and <dbname> are information to locate the PostgreSQL server and the database to connect to it. Other parameters, like the mapping and schema files, are common to all types of providers.



For early PostgreSQL 7.x versions, after installing the server, make sure that the start script (/etc/init.d/postgresql) mentions the proper option to accept incoming sockets. If not, edit this file and add this option to the line "su -l postgres ...": -o "-i". If not, connection to the server remotely will not be possible.



For the servlet to successfully connect to the PostgreSQL server, allow non-local connections. Therefore, "host" entries in the Postgres pg_hba.conf file, and set tcpip_socket=true in the postgresql.conf file.



Note that frequent updates, i.e., insert, delete, in PostgreSQL database end up causing PostgreSQL to return strange and inconsistent results. It is recommended that the "vacuum" command be run on the database to clean up the deleted tuples in the tables and fix inconsistent results. This should be done following a successful backup.



The whole connection string can be encrypted in order to prevent password disclosure.



The key generation option fid="auto" (attribute of the "Primary" property in the mapping file) is not supported for PostgreSQL/PostGIS as this DBMS does not support the unique identifier auto-generation option.

PostgreSQL

If the data source is not PostGIS-enabled, some restrictions apply:

- Queries mentioning the geometric properties are limited mainly to searches inside a Box: Disjoint, Touches, Crosses, Intersects and BBOX operators have the "Box" behavior. Overlaps, Contains and Distance operators are unsupported, as they produce unexpected results.
- The types of geometries are limited to point, line, polygon. (Doughnut polygons and multigeometries are not supported)

- Index creation: The PostgreSQL database supports several index types except Quad Tree indexes. To create an index on Point geometries, convert the Point geometry onto a box in the index expression: CREATE INDEX ON mytable USING RTREE (box(column)).

If the PostgreSQL database is PostGIS-enabled, inhibit it and use the classical PostgreSQL geometries (point, path, polygon), by setting the "NOPOSTGIS" parameter to "TRUE" in the providers.fac file.

PostGIS

After installing PostGIS, enable the database. PostGIS-enabling the PostgreSQL database is achieved by running a couple of Postgres commands explained in the PostgreSQL documentation:

- createlang plpgsql <my_database>
- psql -d <my_database> -f postgis.sql

Check that the database is PostGIS-enabled by looking for a table named "geometry_columns".



ERDAS PostgreSQL provider only needs the "POSTGIS" and the "GEOS" libraries to be installed. The "PROJ" library, used for coordinate transforms, is not used.

Shapefiles

As explained in the "Provider Configuration" chapter, in the providers.fac file, add a <CREATE> element in which the JCLASS attribute must be:

```
com.ionicsoft.wfs.provider.shapev2.ShapeProvider
```

The "path" and "multipath" parameters give the access path(s) to the Shape and DBF files. Note that on Unix platforms, all file extensions must be lowercase.

Other parameters like the mapping and schema files, are common to all types of providers.

Restrictions and limitations: At this time, the operations allowed on this type of data source are limited to non-transactional ones and queries on geometric properties can only search them in a rectangular box.

ArcSDE

To set up an ArcSDE connector, use the Administration Console to add a new service of type "ArcSDE Provider".

The "Connection String" parameter must be set to locate the ArcSDE server: <host>, <port>, <username>, <password> and <instancename> are information to locate the ArcSDE server and the database to connect to it. Other parameters, like the mapping and schema files, are common to all types of providers.

Restrictions:

- Queries mentioning the geometric properties are limited to one geographic filter per query. The supported operators are: BBOX, Intersect, Equals, Disjoint, Within, Overlaps, Crosses and Contains. Beyond, Neighbor and DWithin are not supported because they are not implemented in the underlying ArcSDE API.
- The types of geometries are limited to point, line, polygon.



Before configuring the service, it is necessary to add the ArcSDE SDK library to the web app. This library is composed of a small set of jar files available in the ArcSDE Installation Directory, under the lib folder. They can be named jsdeXX_sdk.jar, jpeXX_sdk.jar, and possibly icu4j.jar where XX is the version of ArcSDE.

Copy those jar files into <APOLLO_HOME>/webapps/erdas-apollo/profiles/eas/WEB-INF/lib (for APOLLO Essentials) or into <APOLLO_HOME>/webapps/erdas-apollo/profiles/eaim/lib (for APOLLO Advantage/Professional). Rebuild the erdas-apollo webapp by running ant from <APOLLO_HOME> as described in [Rebuilding the Webapps](#) and redeploy them into your servlet engine as described in [Deploying WAR Files on Supported Servlet Engines](#).

To use some of the command-line tools with ArcSDE, copy those jar files in the <APOLLO_HOME>/tools/ directory and update the runfromsqlsde. scripts accordingly.*



The whole connection string can be encrypted in order to prevent password disclosure.

Tips on how to create ArcSDE geographic tables

Reminder: ArcSDE uses the concept of TABLE similar to Oracle as a storage of classical (i.e. nongeographic) information. As soon as a geographic dimension is given, by means of one or more geometric columns it is designated a FEATURE CLASS.

How to Create ArcSDE Geographic Tables Over ArcSDE/Oracle Spatial

Depending on the nature of the data, several methods can be used.

- Shapefile Data: Use the "shp2sde" command available in the ArcSDE distribution to populate the data. If the geometry is to be stored as an Oracle Spatial Geometry Type (SDO_GEOMETRY), make sure that the DBTUNE table contains an entry with parameter_name="GEOMETRY_STORAGE" and config_string="SDO_GEOMETRY". If not, the geometry will be created either as an ArcSDE Binary or as an ArcSDE BLOB. Refer to ArcSDE Configuration and Tuning Guide for Oracle for more information.

Example of a shp2sde command

The following command takes the BUSINESS.shp,shx,dbf files provided in the distribution (under data/erdas-apollo/db/arcSDE) and populates them as an ArcSDE feature class for the user "SDE" using the "ERDAS" keyword as entry in the DBTUNE table :

```
shp2sde -o create -l business,geometry -f BUSINESS -a all -G  
26986 -e p -u SDE -k ERDAS
```

- SQL Table Creation Script: Run it first with an Oracle front end like SQL*Plus. Then "register" the feature class in ArcSDE using the sdelay -o register ... command, as in the following example.

Example of an Oracle spatial table registration in ArcSDE

The following commands create the Oracle BUSINESS table and registers it as a feature class of type "point" in ArcSDE, under the "SDE" user, using the "BUS_ID" column as primary key managed by the user. Both scripts are available in the distribution, under data/erdas-apollo/db/arcSDE and in data/erdas-apollo/db/arcSDE/bus_create_sde.txt.

```
sqlplus sde/sde @bus_create.sql sdelay -o register -l  
business,geometry -u SDE -c bus_id -u SDE -e p -C USER
```

- ArcSDE `shtable -o create ...` command: This is, available in the ArcSDE distribution followed by a `sdlayer -o add ...` command. Make sure to have the proper value for the `GEOMETRY_STORAGE` parameter in the `DBTUNE` table to ensure the geometry is created as an Oracle Spatial `SDO_GEOMETRY`. The following example illustrates this method.

Example of creating a feature class directly in ArcSDE

The following commands are taken from the `bus_create_sde.txt` file provided in the distribution under `data/erdas-apollo/db/arcSDE`. They create a table named "BUSINESS", in the "SDE" user and then converts it into a feature class with "geometry" as geographic column of type Point.

```
shtable -o create -t business -p SDE -u SDE -d "BUS_ID integer,
BUS_NAME string(25),BUS_TYPE string(10),
STREET_NAM string(17), BUILDING_N string(4), POSTCODE
string(5), CITY string(8), TELEPHONE string(12),
TOTAL_EMPL double(12,1),LOCK_ID string(255)"
sdlayer -o add -l business,geometry -e p -u SDE -G 26986 -g 500
-p SDE -k ERDAS
```

- Other tools, like FME and ArcGIS, allow data to be imported into the ArcSDE database.

How to Create ArcSDE Geographic Tables Over ArcSDE/MS-SQL

Depending on the initial data, several methods can be used to import them in ArcSDE/MS-SQL.

- Importing Shapefile data into ArcSDE can be easily done using a `shp2sde` tool like for ArcSDE/Oracle. The `GEOMETRY_STORAGE` default setting (to `SDEBINARY`) is generally best.
- Use one of the MS-SQL editors to run a SQL script. The distribution contains the `bus_create_mssql.sql` and `lock_mssql.sql` scripts that match the MS-SQL syntax.
- Create the feature class combining the ArcSDE `shtable` and `sdlayer` commands like for ArcSDE/Oracle.

ERDAS APOLLO WFS over ArcSDE behaves sometimes differently when on top of MS-SQL, compared to the Oracle version. Here are some tips that will assist in troubleshooting and tuning your configuration.

- If the mapping between a feature type attribute and the ArcSDE column mentions a non-existent column name, the WFS GetCapabilities and DescribeFeatureType requests will behave normally. The GetFeature command, however, will return an empty feature collection. No error message will appear, as ArcSDE Java API does not report any.
- In the WFS mapping file, if the schema="..." attribute of the <SQL> element mentions a non-existent SDE user, the result of the GetFeature request will be an empty collection with no error message (ArcSDE Java API does not report any). Moreover, make sure that the user name is in UPPERCASE.
- For the WFS Transactional interfaces to run properly, make sure that the LOCKTIMEOUT table is created. See the files data/erdas-apollo/db/oracle/lock.sql, data/erdas-apollo/db/arcscde/bus_create_sde.txt and data/erdas-apollo/db/arcscde/lock_mssql.sql for ways of creating this table.

Useful ArcSDE Commands

Here is a set of commands commonly used with ArcSDE to set up a WFS.

- To remove a feature class, first remove the layer, then remove the table:

```
sdelayer -o delete -l protectedareas,shape -u sde
```

```
sdetable -o delete -t protectedareas -u sde
```

- Before importing a Shapefile into ArcSDE, look at its structure, using:

```
shpinfo -o describe -f texas_counties -d both
```

- To accurately control the visibility of the tables and columns in the WFS, use the FromSQLGenerator tool for ArcSDE, as provided as part of the distribution.

DBF Files

To set up a DBF connector, add a <CREATE> element into the providers.fac file in which the JCLASS attribute must be:

```
com.ionicsoft.wfs.provider.access.DBFProvider
```

The "connect" parameter must contain a connection string of the form:

```
DBF:///dir+<path>
```

Where <path> gives the access path to the DBF file. Be aware that this path must, on Unix, have the form: //<host>/<dir> where <host> can be empty, and <dir> is a full path, starting and composed of "/" as separator. On Windows platforms, the path is of the form: //<host>/<Unit>:<dir> where <host> can be empty, <Unit> is a unit letter and <dir> is a directory, starting and composed of "\" as separator.

Example on Unix:

```
DBF:///dir+///export/home/Erdas/ArcIMS/data/london
```

Example on Windows:

```
DBF:///dir+C:\ArcIMS\data\london
```

Other parameters, like the mapping and schema files, are common to all types of providers.

Restrictions:

- The geometries supported on this data source can only be of "Point" type, and mapped to a couple of columns, one for the X and one for the Y. The mapping has to be achieved with a <Geometry> tag, the "nameSQL" attribute mentioning the column names. Example:
<Geometry name="Geometry"
nameSQL="COORD_X,COORD_Y" />
- The operations allowed are limited to non-transactional ones
- Queries mentioning the geometric properties can only search them in a rectangular box

Microsoft

As soon as the product is installed on a Windows server or workstation, it is possible to access data sources that are ODBC compliant.

ODBC data source

If the database is defined by an ODBC source on the server, it is possible to manage it by using the JCLASS value of:

```
com.ionicsoft.wfs.provider.access.OdbcProvider .
```

The "connect" parameter must contain a connection string of the form:

```
odbc:///source+<source_name>
```

Where <source_name> gives the name of the ODBC source defined.

Other parameters, like the mapping and schema files, are common to all types of providers.

Restrictions:

- The geometries supported on this data source can only be of "Point" type, and mapped to a couple of columns, one for the X and one for the Y. The mapping has to be achieved with a <Geometry> tag, the "nameSQL" attribute mentioning the column names. Example:
<Geometry name="Geometry"
nameSQL="COORD_X,COORD_Y" /> . See more complete example in next section.
- The operations allowed are limited to non-transactional ones
- Queries mentioning the geometric properties can only search them in a rectangular box

MS-Access

If the database is a .mdb file, it is possible to use either the previous ODBC provider or one specific to MS-Access files. This last one will provide additional capabilities as it will allow requests containing "LIKE" clauses which templates are compatible with a Microsoft implementation.

The JCLASS value of

com.ionicsoft.wfs.provider.access.AccessProvider .

The "connect" parameter must contain a connection string of the form:

```
odbc:///source+<source_name>
```

Where <source_name> gives the name of the ODBC source defined.

Other parameters, like the mapping and schema files, are common to all types of providers.

Restrictions:

- The geometries supported on this data source can only be of "Point" type, and mapped to a couple of columns, one for the X and one for the Y. The mapping has to be achieved with a <Geometry> tag, the "nameSQL" attribute mentioning the column names. Example:
<Geometry name="Geometry"
nameSQL="COORD_X,COORD_Y" />
- The operations allowed are limited to non-transactional ones
- Queries mentioning the geometric properties can only search them in a rectangular box

Example of a Provider Entry on top of a MS-Access Database

```
<CREATE ID="ESA_ACCESS"  
JCLASS="com.ionicsoft.wfs.provider.access.AccessProvider">
```



```

<PARAM NAME="name" VALUE="ESA_ACCESS"/>
<PARAM NAME="title" VALUE="ERDAS WFS server over ESA ATSR
Fires"/>
<PARAM NAME="connect" VALUE="odbc:///source+ESA" />
<PARAM NAME="types" VALUE="obj:///ESA_ACCESS_types.xml" />
</CREATE>

```

In the above provider entry, the mapping and schema file are merged in one file, `ESA_ACCESS_types.xml`. For the mapping file, as MS-Access has no geometry data types, a geometry tag as sub-element of the SQL tag has to be used (see the description in [Table 45: Sub-Elements of the SQL Tag](#)). Here is an example of mapping for the above example, assuming the columns `ESAF_LONG` and `ESAF_LAT` in your table hold the X and Y coords of the geometry.

Example: The mapping and schema file for a MS-Access provider

```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE xsd:schema PUBLIC "-//W3C//DTD XMLSCHEMA 19991216//EN"
"" >
<xsd:schema xmlns:xsd="http://www.w3.org/1999/XMLSchema"
  xmlns:gml="http://www.opengis.net/gml"
  xmlns:wfs="http://www.erdas.com/wfs"
  targetNamespace="http://www.erdas.com/wfs"
  elementFormDefault="qualified" version="0.1">

  <xsd:import namespace="http://www.opengis.net/gml"

schemaLocation="http://www.opengis.net/namespaces/gml/core/feature.xsd" />

  <FeatureType name="FIRE" content="elementOnly"
base="gml:AbstractFeatureType">
  <xsd:element name="DATE" type="xsd:date" minOccurs="0"
maxOccurs="1"/>
  <xsd:element name="HOUR" type="xsd:string" minOccurs="0"
maxOccurs="1"/>
  <xsd:element name="NDVI" type="xsd:string" minOccurs="0"
maxOccurs="1"/>
  <xsd:element name="STATION" type="xsd:string" minOccurs="0"
maxOccurs="1"/>
  <xsd:element name="LAT" type="xsd:float" minOccurs="0"
maxOccurs="1"/>
  <xsd:element name="LONG" type="xsd:float" minOccurs="0"
maxOccurs="1"/>
  <xsd:element name="Geometry" type="gml:PointPropertyType"
minOccurs="0" maxOccurs="1"/>
</FeatureType>

  <Mapping>
  <SQL name="wfs:FIRE">
  <Table nameSQL="BS_V_ESA_FIRE" />
  <Primary name="ESAF_ID" type="xsd:integer" />
  <Element name="DATE" nameSQL="ESAF_DATE" />
  <Element name="HOUR" nameSQL="ESAF_HOUR" />
  <Element name="NDVI" nameSQL="ESAF_NDVI" />

```

```
<Element name="STATION" nameSQL="ESAF_STATION" />
<Element name="LAT" nameSQL="ESAF_LAT" />
<Element name="LONG" nameSQL="ESAF_LONG" />
<Geometry name="Geometry" nameSQL="ESAF_LONG,ESAF_LAT" />
</SQL>

<Info name="wfs:FIRE">
  <SRS>EPSG:4326</SRS>
  <BoundingBox SRS="EPSG:4326" minx="-180." miny="-" maxx="180."
maxy="90." />
</Info>
</Mapping>

</xsd:schema>
```

SQLServer 2008

This connector allows you to create a vector service provider whose data source is a Microsoft SQL Server 2008 database.

The easiest way to create this type of vector service provider is to use the service provider creation wizard inside the ERDAS APOLLO Data Manager, but you can also create it manually by editing the `providers.fac` file for vectors. This file is located inside the directory `<APOLLO_HOME>\config\erdas-apollo\providers\vector`.

To create an entry for this service provider, you need to add a `<CREATE>` element with the value of the `JCLASS` attribute set to `com.ionicsoft.wfs.provider.sqlserver.SqlServerProvider`, and the `CONNECT` parameter for the entry must contain a connection string in the following format:

```
sqlserver://<host>[:<host>[:<port>]/database+<dbname>/user+<username>/password+<password>
```

Where the `<host>`, `<port>`, `<dbname>`, `<username>` and `<password>` are the name of the computer where SQL Server is installed, the port number to communicate with the SQL Server instance, the name of the database containing the vector data, a user account with permission to access the database, and the password for that account.

Adding Shapefiles to a Microsoft SQL Server 2008 Database

If you need to add shapefiles to a SQL Server database, you can use one of the following tools:

- **Shape2SQL**
Available for free from the SharpGIS website at <http://www.sharpgis.net/page/Shape2SQL.aspx>
- **FME Workbench**
Included with the ERDAS APOLLO Feature Interoperability add-on.

GML and GML-T

This connector permits the use of the WFS interfaces on top of a GML document. This document, as it is mentioned by a URL, can be either a file or the result of a HTTP call.

In the providers.fac <CREATE> element, the JCLASS attribute must be:

com.ionicsoft.wfs.provider.gml.GMLProvider (or GMLTransProvider to set up a WFS-T)

The "connect" parameter must contain an URL to the GML document to parse. In particular, if the document is an XML file located in the same directory as the providers.fac, this URL will be:

```
obj:///mygmlfile.xml.
```

The schema file, given in the "types" parameter, must be the one on which the GML document validates.

The mapping file must contain at least the following entries:

- A <Mapping> tag with a <SQL> section for each feature type. In that section, there must be a minimum of a <NoPrimary/> entry. There can be no <Element> tag. But if there is one or more <Element> tag defined, it must be valid in the sense that the given feature property name must be found in the schema file.
- A <Info> tag per feature type, or one for all. If none, the capabilities document will not declare this feature type. This <Info> tag will contain at least a <SRS> tag and a <BoundingBox> tag. If not, the default values will be taken.

Limitations:

- Queries mentioning the geometric properties are limited, mainly to Bounding Box searches.
- The size of the connected document must be small as each request implies parsing the whole document.
- The XML Schema file mentioned in the "types" parameter must be the one that validates the GML document.
- No feature or property name can be changed. The features are output as they are in the GML document.
- Only the GMLTransProvider supports the Transaction operation (INSERT, UPDATE, DELETE). Integrity is not guaranteed if concurrent transactions are sent.

This connector allows you to create a vector service provider whose data source is a DGN V7 or DGN V8 file.

You can create this type of vector service provider using the service provider creation wizard in the ERDAS APOLLO Data Manager, or you can create it manually by editing the `providers.fac` file for vectors. Either way, you will first need to install the ERDAS APOLLO Feature Interoperability add-on so that the ERDAS APOLLO system can work with DGN V7 and DGN V8 files. The instructions for installing the ERDAS APOLLO Feature Interoperability add-on are available in Appendix B of the *ERDAS APOLLO QuickStart Guide*.

To manually create an entry for this type of service provider, you need to open the `providers.fac` file located inside the directory `<APOLLO_HOME>\config\erdas-apollo\providers\vector`. Add a `<CREATE>` element with the value of the JCLASS attribute set to `com.ionicsoft.wfs.provider.fme.FmeProvider`.

Inside the `<CREATE>` element, you need to have the following parameters:

```
<PARAM NAME="FORMAT" VALUE="IGDS"/>
<PARAM NAME="TYPES" VALUE="defaultsql3:4326"/>
<PARAM NAME="METAURL" VALUE="inherit"/>
<PARAM NAME="OWSINFOURL" VALUE="./wfs_md.xml"/>
<PARAM NAME="CONNECT" VALUE="C:\\Erdas\\Data\\fme_dgn\\43j1.dgn"/>
```

The `TYPES` and `MAPPING` parameters can be set to GML schema and mapping files for more flexible settings. They can be automatically produced with the "Generate Types and Mapping" query in the Data Manager. They can also be produced manually from the "runfromsqlfme.bat" script available in `tools/ows`.

The `FORMAT` parameter has to be set to "IGDS" for V7 and V8 DGN files.

The Directives parameters defined by FME can be overridden. Note that the `OEM_LICENSE_ID` and `OEM_KEY` directives are preset to the same values as ERDAS IMAGINE, according to the license terms. They can be changed by explicitly setting the following parameters:

```
<PARAM NAME="OEM_LICENSE_ID" VALUE="my_license_id" />
<PARAM NAME="OEM_KEY" VALUE="my_oem_key" />
```

Limitations

- The ERDAS APOLLO Feature Interoperability add-on is currently only available for Windows machines.
- The FME/DGN connector only supports DGN V7 and V8.

- The ERDAS APOLLO Feature Interoperability add on is completely separate and different from ERDAS IMAGINE Feature Interoperability.

Technical Details

- The portrayal of DGN data is predefined in a jar file, for the "fme-type" base data type. It can be overridden just like the other data types.
- The "Generate Types and Mapping" task of the Data Manager can also be run from the command-line by calling the appropriate FromSqlGenerator tool (like the existing ones defined in <APOLLO_HOME>/tools/ows). The appropriate script is named runfromsqlfme.bat . To run it successfully, you need to use at least the -connection and -table parameters.

For example:

```
runfromsqlfme -connection C:/Erdas/Data/MyDgnFile.dgn -table %
```

- Portrayal of symbology: the default FME Directive on styling symbology results in the symbols to be expanded as geometries (polygons,...) associated to the features. Changing the directive value in the providers.fac file will lead to the symbol being converted as a Point geometry associated to the feature.
- The feature types are built based on the DGN class numbers with the semantic: FT<classnr>. It can be changed in the mapping file.
- Some feature types correspond to decoration in the map: frame, title,... They can be ignored.
- The texts and lines default styling is using the portrayal directives found in the DGN file. For an alternate styling, additional styles can be created the same way as the other service types.

Proxy WFS

This connector applies to feature servers that are already OGC-WFS compliant.

Its goal is to act as a Proxy to prevent the user from directly accessing the given feature server. This may be done either because it is in an Intranet or because the client, an applet for example, is not allowed to connect to any server other than instance of ERDAS APOLLO one. This proxy role can also be extended to allow security negotiation with the underlying service when secured. It is achieved using the "USER" and "PASSWORD" parameters. ERDAS APOLLO currently supports both BASIC and DIGEST authentication mechanisms.

Example of a Proxy WFS Entry

```
<CREATE ID="PROXY"
JCLASS="com.ionicsoft.wfs.provider.proxy.ProxyProvider" >
<PARAM NAME="title" VALUE="ProxyProvider on WFS"/>
<PARAM NAME="abstract" VALUE="ProxyProvider"/>
<PARAM NAME="keywords" VALUE="Proxy" />
<PARAM NAME="connect"
VALUE="http://webservices.ionicsoft.com/worldData/wfs/WORLDWIDE
" />
<PARAMBLOCK NAME="contact">
<PARAM NAME="Organization" VALUE="ERDAS, Inc."/>
<PARAM NAME="AddressType" VALUE="Postal"/>
<PARAM NAME="AddressBody" VALUE="5051 Peachtree Corners
Circle"/>
<PARAM NAME="City" VALUE="Norcross"/>
<PARAM NAME="PostCode" VALUE="30092-2500"/>
<PARAM NAME="State" VALUE="GA"/>
<PARAM NAME="Country" VALUE="USA"/>
<PARAM NAME="Person" VALUE="Luc Donea"/>
<PARAM NAME="Position" VALUE="Product Manager"/>
<PARAM NAME="Voice" VALUE="+1 770 776 3400"/>
<PARAM NAME="Fax" VALUE="+1 770 776 3500"/>
<PARAM NAME="Email" VALUE="info@erdas.com"/>
<PARAM NAME="OnlineResource" VALUE="http://www.erdas.com"/>
</PARAMBLOCK>
</CREATE>
```

Simple Framework

The Simple Framework is a toolkit for accessing flat files (or other data sources) through a JDBC driver. It provides a framework for connecting custom data sources. Its configuration is described in this section. See the Developer Guide for documentation on how to develop on top of this API.

Introduction

The Simple Framework is a toolkit for accessing flat files through a JDBC driver. It provides a framework for connecting your own vector data sources. This section targets the users of the JDBC interface or the WFS framework. For connecting new kind of vector data sources or file formats please refer to the Developer Guide.

SQL Support

Case sensitivity:

The Oracle convention has been adopted for schema objects (table, column, schema, function and procedure names). In a SQL statement, you represent the name of an object with a quoted identifier or a nonquoted identifier.

- A quoted identifier begins and ends with double quotation marks ("). If you name a schema object using a quoted identifier, then you must use the double quotation marks whenever you refer to that object.
- A nonquoted identifier is not surrounded by any punctuation.

Nonquoted identifiers are not case sensitive. They are interpreted as uppercase. Quoted identifiers are case sensitive. By enclosing names in double quotation marks, you can give the following names to different objects in the same namespace:

```
employees  
"employees"  
"Employees"  
"EMPLOYEES"
```

Note the following names are interpreted the same, so they cannot be used for different objects in the same namespace:

```
employees  
EMPLOYEES  
"EMPLOYEES"
```

The SYS Module

This modules contains the TAB and DUAL tables, and the LOAD_MODULE and a sample UPPER functions.

The TAB Table

This table contains all the tables of the current schema.

Example: Select all the tables of the current schema

```
select * from TAB
```

The DUAL Table

The DUAL table is the dummy table, for generic requests.

Example: Select an expression

```
select UPPER('lower') || ' is upper case' from DUAL
```

The LOAD_MODULE Function

The LOAD_MODULE function allows to load a new module.

Table 39: The LOAD_MODULE Function

Parameters of the "call" command	Type	Description
LOAD_MODULE	void	
SCHEMA	java.lang.String	The schema in which the module must be loaded
CLASS_NAME	java.lang.String	The qualified class name of the module to be loaded

The UPPER Function

The UPPER function returns the uppercase version of a given String

Table 40: The UPPER Function

Parameters of the function call	Type	Description
UPPER	java.lang.String	
STRING	java.lang.String	The String to be uppered

The SPATIAL Module

This module provides a BBOX operator, for spatial extents management.

The BBOX operator allows to make a spatial selection based on a bounding box.

Table 41: The LOAD_MODULE Function

Parameters of the function call	Type	Description
BBOX	java.lang.Boolean	
GEOMETRY	java.lang.Double	A geometry column
XMIN	java.lang.Double	Lowest X coordinate
YMIN	java.lang.Double	Lowest Y coordinate
XMAX	java.lang.Double	Highest X coordinate
YMAX	java.lang.Double	Highest Y coordinate
SRSNAME	java.lang.String	SRS Name

Example: Filter on a bounding box

```
select * from CITIES where SPATIAL.BBOX(GEOMETRY, 5, 50, 6, 51, 'EPSG:4326')
```

The CSV Module

The LOAD_FILE function allows to load a CSV file.

Table 42: The LOAD_FILE Function

Parameters of the "call" command	Type	Description
LOAD_FILE	void	
SCHEMA	java.lang.String	The schema the CSV file must be loaded into
FILE_NAME	java.lang.String	The CSV file path
COLUMN_NAMES	java.lang.String	Comma-separated list of column names (surrounded by quotes)

Table 42: The LOAD_FILE Function (Continued)

COLUMN_TYPES	java.lang.String	Comma-separated list of column types (surrounded by quotes)
SRSNAME	java.lang.String	The SRS Name for the extent coordinates
XMIN	java.lang.Double	Abscisse of the lower left corner of the extent
YMIN	java.lang.Double	Ordinate of the lower left corner of the extent
XMAX	java.lang.Double	Abscisse of the upper right corner of the extent
YMAX	java.lang.Double	Ordinate of the upper right corner of the extent

We could imagine to have a LOAD_DIR function that loads the CSV files found in a directory.

Using the Simple Framework

In the WFS providers.fac, a small set of parameters is mandatory for a Simple Framework provider to start: connect, types and initscript.

The `connect` parameter:

This parameter contains the JDBC connection string. The protocol must be `param` and the host must be `localhost`. A "driverclass" attribute must be set to `com.ionicsoft.wfs.jdbc.geojdbc.jdbc.GeoDriver`. A "database" attribute will set the initial schema of the session.

The `types` parameter:

This parameter references a file that will hold the XML schema and the mapping. But you can also set the value "defaultsql", to allow the framework to build all the feature types information on the basis of the JDBC metadata.

The `initscript` parameter:

It references the method to invoke for loading a given data source. Its content is mainly a set of calls to the "call" function, the arguments varying depending on the driver to use. Note that you can configure this parameter with more than one "call", in order to present data from heterogeneous datasources into a single WFS.

In the ERDAS APOLLO distribution, the following modules and the corresponding callable invokers are available:

- CSV (Comma Separated Value File Format): LOAD_FILE (as described above and source provided in the docs directory of the distribution)

- MAPINFO (MapInfo MID/MIF vector exchange format):
LOAD_FILE(String schemaName, String midFileName, String mifFileName, String tableName, String srs), LOAD_DIR(String schemaName, String dir, String srs) (sample provider provided in the WFS providers.fac)
- SHAPE (ESRI Shapefile): LOAD_SHAPE(String schemaName, String dir, String name, String srs), LOAD_DIR(String schemaName, String dir, String srs)
- TEXT (simple tab-separated text files): LOAD_FILE(String schemaName, String tableName, String file, String columnList)

For the sample CSV connector, the object is named "csv", and the loading method is "load_file", to load an individual file. The parameters to the method are: the schema, the path to the file, the comma-separated list of columns names and types, and the extent of the data. Example: load_file('WORLD', 'c:\data\country.csv', 'name, population,geom', 'STRING, LONG, GEOMETRY', 'EPSG:4326', -102.0, 10.0, 30.0, 50.60)

Example of a Simple Framework WFS Entry

```
<CREATE ID="SAMPLE_CSV"
JCLASS="com.ionicsoft.wfs.provider.simple.SimpleProvider" >
  <PARAM NAME="title" VALUE="CSV Countries"/>
  <PARAM NAME="connect"
VALUE="param://localhost/driverclass+com.ionicsoft.wfs.jdbc.geo
jdbc.jdbc.GeoDriver/database+WORLD" />
  <PARAM NAME="types" VALUE="defaultsql" />
  <LSTRING NAME="initscript">
    call
    CSV.LOAD_FILE('WORLD', 'C:\Erdas\ApolloServer\data\csv\country.c
sv', 'name,population,geom',
    'STRING, LONG, GEOMETRY', 'EPSG:4326', -102.0, 10.0, 30.0,
50.60);
  </LSTRING>
</CREATE>
```

WMS - or Raster - Connectors

The first set of raster connectors are applied to the ERDAS Image Server. Another set of connectors mainly act as "Proxies" to various map sources Those sources are either remote (Proxy WMS, Portray Provider) or local (Map Dressing, Pyramid Provider).

Simple Image

Appendix "Image Server" section 2 describes in detail all parameters of this provider.

Image Collection

Appendix "Image Server" section 3 describes in detail all parameters of this provider.

Multiple Images

Configuring a set of images consists of configuring each of the individual images in a single directory and, then, mentioning the directory in a provider. By doing so, each image will become a Layer in the server. The layer name is similar to the image name. The difference is that the "." separating the image name and its extension is replaced with an underscore (_).

Example of a Multiple Image Provider Entry

```
<CREATE ID="ATLANTA_IMGLIST"
  JCLASS="com.ionicsoft.wmtmap.provider.image.MultiSimpleProvider"
  >
  <PARAM NAME="SRS" VALUE="EPSG:4269"/>
  <PARAM NAME="PATH"
  VALUE="C:/Apollo/ErdasApolloServer/data/erdas-
  apollo/images/landsat_imgtiles"/>
  <PARAM NAME="NAME" VALUE="ATLANTA_IMGLIST"/>
  <PARAM NAME="TITLE" VALUE="Landsat List of Images"/>
</CREATE>
```

Proxy WMS

This connector applies to map servers that are already OGC-WMS compliant. It achieves two goals.

First, it acts as a Proxy to prevent the user from directly accessing the given map server. This may be done either because it is in an Intranet or because the client, an applet for example, is not allowed to connect to any server other than instance of ERDAS APOLLO one. This proxy role can also be extended to allow security negotiation with the underlying service when secured. It is achieved using the "USER" and "PASSWORD" parameters. ERDAS APOLLO currently supports both BASIC and DIGEST authentication mechanisms.

Secondly, it allows the enhancement of the connected server that could have some limitations. The enhancements are either automatic (output format, coordinate transformation, etc.) or explicit (through parameters like LIMITEDCOLOR, LIMITEDSIZE and LIMITEDTRANSPARENCY). Note that enhancements can also be restrictions, through the REMOVE_MAP_FORMAT and REMOVE_INFO_FORMAT, to restrict the set of output formats permitted.

Example of a Proxy WMS Entry

```
<CREATE ID="PROXYDEMIS"
  JCLASS="com.ionicsoft.wmtmap.provider.proxy.ProxyProvider">
  <PARAM NAME="TITLE" VALUE="Proxy on DEMIS" />
  <PARAM NAME="ABSTRACT" VALUE="Proxy on DEMIS" />
  <PARAM NAME="URL"
  VALUE="http://www2.demis.nl/wms/wms.asp?wms=WorldMap" />
</CREATE>
```

Map Dressing

The MapDressing service allows the placement of common map production elements (north arrow, scale bar, grid, etc). As MapDressing is a provider in the WMS servlet, configuration information is located in this appendix.

To configure, enter the connector type, the name of the configuration file, and provide the path in the root rendering directory that contains the needed styles.

Example of a Map Dressing Entry

```
<CREATE ID="MAPDRESSING"

JCLASS="com.ionicsoft.wmtmap.provider.map.MapPresentationProvider">
  <PARAM NAME="NAME" VALUE="MAPDRESSING"/>
  <PARAM NAME="TITLE" VALUE="ERDAS APOLLO WMS server for maps
  dressing"/>
  <PARAM NAME="ABSTRACT" VALUE="OGC-compliant Map Server
  maintained by ERDAS (http://www.erdas.com). It allows you to add
  some Map Layout to the produced maps."/>
  <PARAM NAME="LAYERS" VALUE="obj:///mappresentation_layers.xml"
  />
  <PARAM NAME="RULEDIR"
  VALUE="D:/Erdas/ApolloServer/config/erdas-apollo/rendering/" />
</CREATE>
```

The "layers" parameters mention the file containing the configuration: layer names and titles, style names and titles, dimension names and defaults.

It is recommended that no change is made to the content of this file except for cosmetic needs, i.e., title, default value. If more changes are made to this file other than the recommendations provided above, ensure that the portrayal styles are still coherent with the changes, and that the requests still succeed.

Use of this service is described in the "Portrayal Capabilities" chapter.

Pyramid Provider

For various reasons, map providers may want to display different data depending on the scale of the map extent. One reason is for the accuracy of the data: a satellite image taken at a scale of 1:1000 may display poorly at a scale of 1:100.000. Conversely, a digital photo taken at a scale of 1:100.000 will appear as big pixels at a scale of 1:1000 and could be replaced with another photo taken at a smaller scale. Map providers may also want to have the option to switch from raster imagery displayed at large scale to vector data at smaller scales.

The pyramid provider fulfills all of these use cases, but the most common scenario is to display and configure the same imagery data at several scales depending on the viewable scale range.



A Standard Scale, as defined in the OGC SLD 1.0.0 Implementation Specification, is actually the scale denominator relative to a "standardized rendering pixel size". The "standardized rendering pixel size" is defined to be 0.28mm × 0.28mm (millimeters).

What is the Pyramid Provider?

The pyramid provider connector acts as a proxy provider on top of one or more providers and chooses between the providers for each request depending on the requested scale. Therefore, it is necessary to configure one provider for each different data sources and display scale, plus the pyramid provider itself.

Pre-Requisites - Set up Providers to be Proxied

Since the pyramid provider proxies providers, each of the providers need to be already defined in the providers.fac file.

The pyramid provider will inherit the global capabilities information from one of the proxied providers, named the "master". So, configure at least one of the proxied providers with meaningful information for the service name, title and abstract, the supported requests, the layer list, the supported SRSEs.

However, the other proxied provider will also supply information to the pyramid to let it figure out how to proxy the requests and maybe convert the outputs, e.g., supported output formats, size and transparency limitations.

A single major constraint has to be fulfilled by each of the proxied providers; their layer names must be the same as the master provider's one. Because each GetMap request to the pyramid mentions one or more layer names, those names are forwarded to the proxied providers.

Providers Configuration Example

```
<CREATE ID="BOSTON_LI"
JCLASS="com.ionicsoft.wmtmap.provider.imageProvider.LayerProvid
er" >
  <PARAM VALUE="EPSG:26986" NAME="srs" />
  <PARAM VALUE="C:/Erdas/ApolloServer/data/erdas-apollo/images"
NAME="path" />
  <PARAM VALUE="BOSTON_LI" NAME="name"/>
  <PARAM VALUE="Boston Imagery" NAME="title"/>
</CREATE>
```

```

<CREATE ID="BOSTON_JPG"
JCLASS="com.ionicsoft.wmtmap.provider.imageProvider.SimpleProvider" >
  <PARAM VALUE="EPSG:26986" NAME="srs" />
  <PARAM VALUE="C:/Erdas/ApolloServer/data/erdas-apollo/images/boston_1/237890.jpg" NAME="path" />
  <PARAM VALUE="BOSTON_LI" NAME="name"/>
  <!-- the name is set to the same as the BOSTON_LI provider, to allow pyramid -->
  <PARAM VALUE="Boston Image in JPEG" NAME="title"/>
</CREATE>

```

Pyramid Provider Configuration

Follow the steps outlined below to configure a pyramid provider. As with all provider configurations, direct editing of the providers.fac file will be required.

1. Set a unique ID for the pyramid provider in the current providers.fac file. Like any other WMS provider, the pyramid provider must have an ID that will be used in subsequent WMS requests.
2. Set the JCLASS attribute of the ID to "com.ionicsoft.wmtmap.proxy.ScaleProvider".
3. Then, create one <PARAMBLOCK>...</PARAMBLOCK> block for each provider to proxy.
4. Define the following parameters using the <PARAM> elements for each <PARAMBLOCK>.
 - The "minScale" parameter must be set to a positive number and correspond to a minimum scale at which that provider will be called. The default value is 0.
 - The "maxScale" parameter must be a positive number and correspond to the maximum scale at which that provider will be called. The default value is Double.MAX_VALUE.
 - The mandatory "id" parameter must contain the name of the provider to proxy.
 - The "master" parameter must be set to either "true" or "false". "True" signifies that the provider will export the capabilities information (service name, title and abstract, supported requests, layer list and supported SRSes) to the pyramid. The default value is "false".

It is highly recommended to have only one proxied provider labeled "master". If more than one proxied provider is set as the "master", the one that covers the lowest scale will be used.

5. The list of <PARAMBLOCK> blocks will be automatically sorted by ascending scale range.

Example:Pyramid Provider Configuration

```
<CREATE ID="PYRAMID"
JCLASS="com.ionicsoft.wmtmap.provider.proxy.ScaleProvider" >
  <PARAMBLOCK >
    <PARAM NAME="minscale" VALUE="0" />
    <PARAM NAME="maxscale" VALUE="100000" />
    <PARAM NAME="id" VALUE="BOSTON_LI" />
    <PARAM NAME="master" VALUE="true" />
  </PARAMBLOCK>
  <PARAMBLOCK >
    <PARAM NAME="minscale" VALUE="100000" />
    <PARAM NAME="id" VALUE="BOSTON_JPG" />
  </PARAMBLOCK>
</CREATE>
```



For performance purposes, a configuration of 20 or less proxied providers per pyramid is recommended.

Usage Tips

- Ensure that each of the proxied providers publishes similar layer names.
- Ensure the pyramid provider does not contain scale range overlaps or gaps. To avoid gaps, put similar extreme values. If two providers have overlapping scales, the one with the smallest "minScale" value is taken.



If the request corresponds to a scale gap, the master provider will be called.

- Ensure that only one PARAMBLOCK item has the "master" parameter set to true. If no master is set, the provider corresponding to the lowest scale will be used for to provide capabilities information.

Portray Provider

What is the Portray Provider?

The portrayal provider is a connector that allows a user to obtain maps from WFSes using SLD as information encoding.

This provider is an OGC-compliant WMS because it provides valid responses to GetCapabilities and GetMap requests.

However, the portray provider does not disclose layer or style information. Layers and styles are defined in the SLD document that MUST be passed along to any GetMap request. This also means that any attempt to send a GetMap request without the associated SLD document which contains UserLayers and UserStyles elements will produce an error.

Configuration

Similar to other provider types, an entry in the providers.fac file of the "map" servlet is needed. Classically, this entry will contain the "ID" of the provider: a JCLASS specifying the provider type. The entry must be written as follows:

```
'com.ionicsoft.wmtmap.provider.sld.PortrayProvider'.
```

The following parameters need to be supplied:

- The "name" parameter will appear as Service Name in the capabilities document.
- The "title" parameter will appear as Service Title in the capabilities document.
- A "PARAMMAP" element indicates where the portrayal styles and symbols needed by the provider are located and how they are managed. This element also has a set of attributes:
 - The NAME must be "portrayconfig"
 - DIR notes where pre-defined styles and symbols will be found. It is relevant when "NamedStyles" are queried or when symbol names are mentioned in "Mark" elements.
 - LOADER defines the order of precedence for the various types of styles. It only applies when Named Styles are used.
 - VERSION identifies the portrayal styles syntax version. The default value is 2.
 - MANAGEMENT indicates the engine caching behavior for previously loaded styles. "Always" implies no caching, "none" means full caching, and "checked" means that the style timestamp is checked before re-loading.

The example below demonstrates how a Portray provider is configured.

Example of a Portray WMS Entry

```

<CREATE ID="PORTRAY"
JCLASS="com.ionicsoft.wmtmap.provider.sld.PortrayProvider" >
<PARAM NAME="name" VALUE="SLD_Portray"/>
<PARAM NAME="title" VALUE="SLD Portray WMS"/>
<PARAMMAP NAME="portrayconfig"
  DIR="C:\Erdas\ApolloServer\config\erdas-apollo\rendering"
  LOADER="java,property,sld"
  VERSION="2"
  MANAGEMENT="always" />
</CREATE>

```

How to Set up Requests to the Portray Provider

Firstly, ensure that all necessary information about the features that will be portrayed is available. This includes the URL of the WFS, the name of the feature type(s) and the relevant properties of these feature types. If only a subset of these features is to be portrayed it will be necessary to build an OGC-Filter expression that expresses this subset.

Secondly, indicate how each feature type is to be portrayed. If classifying features with one or more of properties, build as many portrayal rules as there are classes. If displaying all the features the same way, possibly with portrayal parameters varying the functions of one or more feature properties, one rule is sufficient.

Next determine the symbology to be used for each rule. SLD proposes a set of basic symbolizers such Point, Line, Text, or Polygon. For each, a different set of parameters must be set including stroke color, fill pattern, and label size.

Finally, decide the various parameters of the GetMap request. These parameters include: Box, SRS, width and height, background color, and transparency.

After accomplishing those steps, build the SLD document. Use the [OGC StyledLayerDescriptor 1.0 Implementation Specification](#) (available on the OGC website) for additional help. Check which SLD elements and tags are supported in the provider by referring to the SLD tags table in the "Portrayal Capabilities" chapter. This chapter also contains a sample SLD document.

ArcSDE-Raster Provider

An increasing number of people are storing raster images and coverages into ESRI ArcSDE using the "Raster data storage" option. Starting with ERDAS APOLLO 9.3, there is a way to expose those data as OGC WMS-compliant maps and very soon, as OGC WCS-compliant coverages.

This section provides some configuration information to be able to wrap one or more ArcSDE-Raster layers and expose them in the Interoperability bus.

The connector has been successfully tested with an ArcSDE 9.0 Java library connecting to an ArcSDE 9.0 server. Note that the ArcSDE C/C++ API is no more supported. It means the set up has to use the "ArcSDE Raster Provider (Java)" and no more the "ArcSDE Raster Provider (Binary)".

The parameters supported by this provider type are:

- **Connection String:** the connection string to the ArcSDE server. It is composed of the ArcSDE server host name or IP address, the service name must be "esri_sde", the user name and password in the ArcSDE database. (Mandatory)
- **layers:** this parameter is a "block" parameter holding the definition of one or more layers:

Parameters inside the "layers" parameter block:

- **Title:** the title given to the service. It will appear in the capabilities document. (Optional)
- **Abstract:** an abstract describing the service. It will appear in the capabilities document. (Optional)
- **table:** the ArcSDE table name, or a pattern using "%" and "?" as wildcards. See below for details on multi-layer definitions. (Mandatory)
- **pattern:** as soon as the "table" parameter contains a wildcard, this parameter must be made known with the value set to "true". (Optional)
- **column:** the name of the column that holds the raster data. It is needed when more than one column contain raster data. (Optional)
- **srs:** The coordinate system of the data framing is normally stored in the ArcSDE table. The esri.txt file included in cots-srs.jar makes the matching between that ESRI SRS name and the corresponding EPSG id used by the servlet. But if the ESRI SRS name is not set, the "srs" parameter can be set in the provider entry to override the one in the database. (Optional). However, it still implies adding the corresponding ESRI SRS value in the usersref.xml file (see **Other IDs** in **SRS Configuration Parameters**).
- **rgb:** defines the bands to associate to the red, green, blue and alpha channels of the produced image. If not set, it will take bands 1, 2 and 3 for R, G and B. The image will be made opaque. Other possible values are defined below. Note that for images of less than 3 bands, the parameter is mandatory as reference to a non-existent band will produce an error. (Optional)

Multi-layer definitions: If there are several ArcSDE raster tables and several layers are to appear in the service, there are two ways to do it. The first way is to use a pattern as table value (e.g. "uk_%"), that will produce one layer per raster table; the layer name will have the table name. The second way is to explicitly define several layers blocks.

RGB color setting: The "rgb" parameter can take a sequence of 3 or 4 comma-separated numbers, one for each color plus one for the alpha channel. It can also be set to one of the following pre-defined constants:

- "rgb": it associates the bands 1,2,3 to R,G,B. The image is opaque.
- "rgba": it associates the bands 1,2,3,4 to R,G,B,Alpha.
- "rgbz": it associates the bands 1,2,3 to R,G,B. The alpha channel is computed using the "Z" convention (alpha = 0 if all bands = 0).
- "bgr": it associates the bands 3,2,1 to R,G,B. The image is opaque.
- "bgrz": it associates the bands 3,2,1 to R,G,B. The alpha channel is computed.

Context Provider

This connector allows an OGC WMS Context to be viewed as a WMS service accepting GetCapabilities, GetMap and GetFeatureInfo requests. The context contained in the context file following the OGC specification can be provided by a variety of sources.

The versions of context files supported are 0.1.3, 0.1.4, 1.0.0, and 1.1.0.

Configuration

In order for this provider to work, insert the proper connector class in the JCLASS attribute.

The second task is to add a "context" parameter giving the name of the context file. The location of that file is a URL and the file name is inserted, its location will be found in the directory of the providers.fac.

```
<CREATE ID="BOSCON"
JCLASS="com.ionicsoft.wmtmap.provider.cascading.ContextProvider
" >
  <PARAM VALUE="CONTEXT ON BOSTON" NAME="name" />
  <PARAM VALUE="Boston Context Provider over services on ERDAS
public demo site" NAME="title"/>
  <PARAM VALUE="boston_ionic_context.xml" NAME="context" />
</CREATE>
```

As seen in the basic example above, it is also possible to add "name" and "title" parameters for custom values to appear as Service metadata in the capabilities document.



The context plugged in the provider cannot contain references to services located in the same web application. If this is done a deadlock will be created and the service will hang.

If some of the layers defined in the context file are to be hidden, or new layers that are aggregates of layers defined in the context file are to be defined, input additional optional parameters in the providers configuration:

- A "cascade" parameter will show the list of layers to expose. It can contain a comma-separated list of layer names, the "*" sign for all layers, or an empty string "" for none.
- The layer aggregates are configured in a block parameter named "aggregates". This block contains a set of sub-blocks, one per virtual layer to expose. Each of those sub-blocks will itself contain a set of parameters:
 - The "name" attribute of the sub-block is used as the key to the aggregate layer. The lowercase "name" is used as the layer name if no "name" parameter is defined in the sub-block.
 - An optional "name" parameter can be used as layer name if the lowercase "name" attribute does not match the deployment needs.
 - The optional "title" parameter will describe the layer in the capabilities document.
 - The optional "abstract" parameter will further describe the layer in the capabilities document.
 - The "layers" parameter will list the set of layers from the context to address when the current virtual layer is invoked. The rule is the same as for the "cascade" parameter. If the value is "*", the layers are ordered as in the context.
 - The optional "featureInfoLayers" parameter contains the layer name on which the GetFeatureInfo request will be forwarded. Note that the system currently only allows one layer name in that tag.
 - The "legendUrl" tag will allow the addition of a <LegendURL> tag in the capabilities document for this layer. The syntax is an URL or file pattern as in the classical LegendURL configurations. In particular, if the value is an empty string, the pattern defined in the "TEMPLATE" attribute of the <LEGEND> tag in the CONFIGURATION section will be used.

The example below illustrates this more advanced configuration

```
<CREATE ID="BOSCON2"
JCLASS="com.ionicsoft.wmtmap.provider.cascading.ContextProvider
" >
  <PARAM VALUE="SECOND CONTEXT ON BOSTON" NAME="name" />
  <PARAM VALUE="Boston Second Context Provider over services on
ERDAS public demo site" NAME="title"/>
  <PARAM VALUE="boston_ionic_context.xml" NAME="context" />
  <PARAM NAME="cascade" VALUE="PLACE_NAMES"/>
  <PARAMBLOCK NAME="aggregates">
    <PARAMBLOCK NAME="BASE_AND_HIGHWAYS">
      <PARAM NAME="name" VALUE="Base_And_HighWays"/>
      <PARAM NAME="title" VALUE="Base + Highways"/>
      <PARAM NAME="layers" VALUE="MASS,HIGHWAYS"/>
      <PARAM NAME="featureInfoLayers" VALUE="HIGHWAYS"/>
      <PARAM NAME="legendUrl" VALUE="{absolute}/{id}-{name}-
default.png"/>
    </PARAMBLOCK>
  <PARAMBLOCK NAME="ALL">
    <PARAM NAME="name" VALUE="ALL"/>
    <PARAM NAME="title" VALUE="Everything"/>
    <PARAM NAME="layers" VALUE="*/>
  </PARAMBLOCK>
</PARAMBLOCK>
</CREATE>
```

Main features

- The context's initial box and scale are used to set the capabilities BoundingBox.
- The context's SRS is published in the capabilities. If the "MaxScale" attribute is used for layers in the context document, the GetMap request will change the visible layers and styles depending on the scale.
- Each Layer defined in the context will produce a layer in the capabilities, except if the "cascade" parameter is used to hide some of them.
- Only the WMS layers and styles mentioned in the contexts file will appear and be callable in the service. All the others are hidden.
- If the "aggregates" block parameter is used, virtual layers can be defined.
- Some of the metadata information of the underlying WMS services can be overridden by information in the context file: BoundingBox, SRS, service Title and Abstract.

- SLD styling found in the context is supported by the service.
- If some setting is done in the providers.fac, such as the title and abstract, it overrides the information found in the Context.

Access to secure service

If several underlying services are secured, the provider should be configured to correctly transmit the login credentials to these services.

The simple way is to put the username and password in the service url (like `http://username:password@host/myservice`), but that's rarely acceptable

Otherwise the application server or this provider should be configured to have access to the credentials definition.

It is also possible to have the authentication context associated to a request made on the ContextProvider to be propagated to each servers defined in the context. It is useful to have a kind of single sign on. To do this, define the parameter `forwardAuthentication` in the configuration. The value of that parameter is not used.

```
<PARAM NAME="forwardAuthentication" VALUE="true" />
```

If the application server has not secured this service, you must secure it to force the application server to transmit the connected user.

```
<PARAM NAME="security" VALUE="" /> <!-- to accept any user using a BASIC scheme-->
```

If no credentials definition is associated to this provider, you must at least ensure the propagation of the application server ones.

```
<PARAM NAME="securityresolver" VALUE="container" />
```

or

```
<PARAM NAME="securityresolver" VALUE="acceptlogin" />
```

The last one will automatically reuse the authentication (basic scheme only) found in the http header as the default login credential.

Oracle 10g GeoRaster Provider

An increasing set of people are storing raster images and coverages into databases, and the Oracle 10g release includes a "georaster" module to store imagery and coverages, build pyramids. It comes with a Java API to load and retrieve those data. In ERDAS APOLLO 9.3, a connector is provided to expose those data as OGC WMS-compliant maps and very soon, as OGC WCS-compliant coverages.

This section explains the necessary configuration information to wrap one or more Oracle GeoRaster layers and expose them in the Interoperability bus.

It is assumed that the Oracle 10g configuration and data sets are valid. The connector has been successfully tested with an Oracle 10.1.0.3 server, using the corresponding Georaster Java API. Proper behavior is not guaranteed with other versions of Oracle and the API. The connector is currently available in Beta state.

Environment Configuration

In order to connect to an Oracle 10g server, ensure the proper JDBC jar file is available in the web application. This file is commonly named ojdbc14.jar and is provided with ERDAS APOLLO.

The Oracle Georaster module does not need any additional library to be installed.

Provider Setup

The setup consists of adding a new entry in the providers.fac file for the "map" servlet, defining for this entry a connection string and setting the Oracle tables to access. A sample provider, named BOSTON_OGEOR, is contained in the distribution. Uncomment it by removing the leading "<!--" line and the trailing "-->" line, and change its values to match the server and data. The sample provider is shown below.

Example of a Oracle GeoRaster Entry

```
<CREATE ID="BOSTON_OGEOR"
JCLASS="com.ionicsoft.wmtmap.provider.oracle.RasterProvider">
  <PARAM NAME="title" VALUE="ERDAS WMS server over Boston"/>
  <PARAM NAME="abstract" VALUE="WMS over BOSTON imagery in an
Oracle GeoRaster server."/>
  <PARAM NAME="connect"
VALUE="oracle://geo.raster.com/user+myuser/password+mypwd/sid+M
YSID" />
  <PARAM NAME="table" VALUE="BOSTON_GEOASTER" />
  <PARAM NAME="column" VALUE="georaster" />
  <PARAM NAME="id" VALUE="2" />
  <PARAM NAME="rasterdatatable" VALUE="rdt_boston" />
  <PARAM NAME="srs" VALUE="EPSG:26986" />
</CREATE>
```

The parameters supported by this provider type are:

- **title:** the title given to the service. It will appear in the capabilities document. (Optional)
- **abstract:** an abstract describing the service. It will appear in the capabilities document. (Optional)
- **connect:** the connection string to the Oracle server. It is composed of a protocol that must be "oracle", the Oracle server host name or IP address, the user, password and sid. (Mandatory)
- **table:** the Oracle table name, or a pattern using "%" and "?" as wildcards. See below for details on multi-layer definitions. (Mandatory)
- **pattern:** as soon as the "table" parameter contains a wildcard, this parameter has to be given, with the value set to "true". (Optional)
- **column:** the name of the column that holds the raster data. It is needed when more than one column contain raster data. (Optional)
- **id:** the id of the image, the default value is 1. (Optional)
- **rasterdatatable:** name of the raster data table associated with the current table. It is needed when more than one raster table is linked to the current table. (Optional)
- **srs:** The coordinate system of the data.
- **rgb:** defines the bands to associate to the red, green, blue and alpha channels of the produced image. If not set, it will take bands 1, 2 and 3 for R, G and B. The image will be made opaque. Other possible values are defined below. Note: The parameter is mandatory for images of less than 3 bands as a reference to a non-existent band will produce an error. (Optional)
- **layers:** this parameter is a "block" parameter holding the definition of one or more layers. See below for details on multi-layer definitions. (Optional)

Multi-layer definitions: There are two ways to implement the situation where there are several Oracle Georaster tables and several layers are to appear in the service. The first way is to use a pattern as table value (e.g. "uk_") that produces one layer per raster table with the layer name having the table name. The second way is to explicitly define several layers defining a set of <PARAMBLOCK> parameters as in the example below.

Example of a multi-layer Oracle GeoRaster Entry

```
<CREATE ID="MULTI_OGEOR"
JCLASS="com.ionicsoft.wmtmap.provider.oracle.RasterProvider" >
  <PARAM NAME="title" VALUE="ERDAS WMS server over several
tables"/>
  <PARAM NAME="connect"
VALUE="oracle://geo.raster.com/iuser+myuser/password+mypwd/sid+
MYSID" />
  <PARAMBLOCK NAME="layers">
    <PARAMBLOCK NAME="layer1">
      <PARAM NAME="table" VALUE="BOSTON_TABLE_1" />
      <PARAM NAME="name" VALUE="MyLayer1" />
      <PARAM NAME="title" VALUE="Title of my layer 1"/>
    </PARAMBLOCK>
    <PARAMBLOCK NAME="layerSet2">
      <PARAM NAME="table" VALUE="MASS_%" />
      <PARAM NAME="pattern" VALUE="true" />
      <PARAM NAME="title" VALUE="Massachussets layer"/>
    </PARAMBLOCK>
  </PARAMBLOCK>
</CREATE>
```

RGB color setting: The "rgb" parameter can take a sequence of 3 or 4 comma-separated numbers, one for each color plus one for the alpha channel. It can also be set to one of the following pre-defined constants:

- "rgb": it associates the bands 1,2,3 to R,G,B. The image is opaque.
- "rgba": it associates the bands 1,2,3,4 to R,G,B,Alpha.
- "rgbz": it associates the bands 1,2,3 to R,G,B. The alpha channel is computed using the "Z" convention (alpha = 0 if all bands = 0).
- "bgr": it associates the bands 3,2,1 to R,G,B. The image is opaque.
- "bgrz"it associates the bands 3,2,1 to R,G,B. The alpha channel is computed.

Limitations

The current implementation of this provider has limitations that will disappear when the implementation becomes final:

- The provider only supports BIP (band interleave by pixel), not BIL (band interleave by line) nor BSQ (Band Sequential) images.
- It is recommended to build square tiles (through the blocking size parameter), so that large scale requests will retrieve a limited set of tiles.

- The ULT (upper left pixel coordinate) must correspond to pixel 0,0.

WCS - or Coverage - Connectors

If there is a coverage or a set of coverages to publish through a WCS, they need to be configured in the framework as one of the four types of WCS connectors: Simple, MultiSimple, Index or Hierarchical. The following sub-sections explain in detail the configuration tasks needed for each of those connector types.



The Hierarchical provider is not part of the ERDAS APOLLO Essentials distribution. It is available in the ERDAS APOLLO Advantage/Professional product. Please contact [ERDAS Support](#) for more information on this provider.

Basically, these connectors support a set of standard formats (as BIL, GeoTIFF,...) using our own imagery decoder (ERDAS GIO decoders). As soon as the "hegpath" parameter is set, the HDF-EOS format is also supported. If the "gdalpath" parameter is set, the formats referenced in the GDAL library and mentioned in the "Using GDAL library" section below are supported as well. For each format, various color depths and types of bands are supported.

Every WCS provider is able to serve an ISO19115 metadata file per coverage offering. These metadata files are automatically generated by the WCS instance if they don't exist yet. The ISO19115 metadata files will be available through each Coverage Offering's metadataLink element in the capabilities document. This process is automatically enabled if the METADATA parameter block is present in the configuration part of the providers.fac file, and if a "metauri" tag is set for your provider.

```
<METADATA
  TEMPLATE="{absolute}/{id}/{metaname}.xml"
  DIR="/home/Erdas/ApolloServer/config/erdas-
apollo/metadata/coverage"
/>
```

Simple Coverage

When the coverage data are held in a single file, possibly along with descriptive files, this connector type is applicable.

In the WCS providers.fac file, the path to the data file and the coordinate system need to be set. Other parameters such as service name, title and description, contact information and a set of keywords provide more information to identify the service.

The following example shows the configuration of a coverage file in the WCS providers.fac. Note that the connector type is given by the "JCLASS" attribute which value must be "com.ionicsoft.wmtmap.provider.coverage.SimpleProvider".

Example of a Simple Coverage Entry

```
<CREATE ID="ATLANTA_SINGLE"
JCLASS="com.ionicsoft.wmtmap.provider.coverage.SimpleProvider"
>
  <PARAM NAME="name" VALUE="ATLANTA_SINGLE"/>
  <PARAM NAME="title" VALUE="Atlanta Coverage Server"/>
  <PARAM NAME="abstract" VALUE="City of Atlanta ECW Imagery"/>
  <PARAM NAME="keywords" VALUE="Atlanta,Coverage,ECW"/>
  <PARAMBLOCK NAME="contact">
    <PARAM NAME="Organization" VALUE="ERDAS, Inc."/>
    <PARAM NAME="AddressType" VALUE="Postal"/>
    <PARAM NAME="AddressBody" VALUE="5051 Peachtree Corners
Circle"/>
    <PARAM NAME="City" VALUE="Norcross"/>
    <PARAM NAME="PostCode" VALUE="30092-2500"/>
    <PARAM NAME="State" VALUE="GA"/>
    <PARAM NAME="Country" VALUE="USA"/>
    <PARAM NAME="Person" VALUE="Luc Donea"/>
    <PARAM NAME="Position" VALUE="Product Manager"/>
    <PARAM NAME="Voice" VALUE="+1 770 776 3400"/>
    <PARAM NAME="Fax" VALUE="+1 770 776 3500"/>
    <PARAM NAME="Email" VALUE="info@erdas.com"/>
    <PARAM NAME="OnlineResource" VALUE="http://www.erdas.com"/>
  </PARAMBLOCK>
  <PARAMBLOCK NAME="CoverageOffering">
    <PARAMBLOCK NAME="ECW">
      <PARAM NAME="name" VALUE="Atlanta"/>
      <PARAM NAME="title" VALUE="ECW Coverage Offering"/>
      <PARAM NAME="abstract" VALUE="ECW Coverage Offering, Imagery
data"/>
      <PARAM NAME="keywords" VALUE="Atlanta,Imagery,remote-
sensing,Coverage,ECW"/>
    </PARAMBLOCK>
  </PARAMBLOCK>
  <PARAM NAME="path" VALUE="C:/Erdas/ApolloServer/data/erdas-
apollo/coverages/mosaic/atl_tiles_1_1.ecw" />
  <PARAM NAME="srs" VALUE="EPSG:2240" />
</CREATE>
```

Note that the "CoverageOffering" parameter block allows you to rename your coverage offerings or just to enrich them with a title, abstract and keywords. One sub-block can be defined per coverage offering, each sub-block is named with the original name of each coverage offering. This functionality works with each type of WCS provider.

Indexed Coverages

In many cases, the data manager has a set of coverage files in a hierarchy of directories.

Moreover, this collection of coverages can be composed of a large volume of files that justifies the setup of an indexing method. In the current release, the indexer is Feature Server-based, and the WCS provider references this Feature source through its providers.fac, and the WFS provider name. The indexing system WFS can be set up either on top of a database (oracle or postgres) or on top of a GML file.

in order to set up a set of indexed coverages, follow these steps:

1. Setup a WFS provider that will reference the coverage indexes. Therefore:

Go to <APOLLO_HOME>/config/erdas-apollo/providers/coverage.

In that directory, edit the indexer.fac file. It contains the definition of several WFS providers, each relating to a database type (Oracle, Postgres or GML file) and instance. Duplicate one and change its name (the ID attribute), its connection string (the "connect" parameter) and its name (give it the same value as the ID).

If the database does not exist, run the create<DB>.sql script to have the tables and indexes built in the chosen database. <DB> stands for "Oracle" or "Postgres".

If using a GML file, start from an existing file. Rename the "empty_gml_index.gml" and use it to build the indexes. Note that the provider type must be a Transaction GML provider (com.ionicsoft.wfs.provider.gml.GMLTransProvider) in order to enable indexing.

2. Setup a WCS over the coverage data. Therefore:

Edit the providers.fac file. In that file, create a new provider to link to the data as in the example below. Most of the information is similar to what must be defined in the case of a simple coverage. The connector type is given in the "JCLASS" attribute and its value must be "com.ionicsoft.wmtmap.provider.coverage.IndexProvider".

The "indexingServer" parameter has to reference the WFS provider name chosen in step 2, and the "indexingProvider" parameter must reference the indexer.fac file name. The "indexingType" parameter must be set at the value "WFS"



To prevent large output times, no more than 25 granules will be extracted for a GetCoverage or GetMap request output. The value can be changed by setting the "MaxStitch" parameter.

3. Populate the index information from the coverages directories into the WFS. To do so, you can either use the desktop "Indexing System Viewer" tool or the command-line "runwfsindexer" tool, both part of the ERDAS APOLLO distribution. See the "tools and viewers" chapter for instructions on how to index coverages with the Indexing System Viewer. For easy indexing using the command-line tool, here is the sequence:

In a console window, go to the <APOLLO_HOME>/tools/ows directory.

In that directory, execute the runwfsindexer tool as explained in the "Data Indexer" section of the "Tools and Viewers" chapter in order to load the reference to the coverage files into the WFS. The command could be:

Example: Sample coverage indexer command

```
# cd /home/Erdas/ApolloServer/tools/ows
# runwfsindexer -factory /home/Erdas/ApolloServer/config/erdas-
apollo/providers/coverage/providers.fac
-name MYWCSPROV -command addDir -datapath
/home/ErdasApollo/data/erdas-apollo/coverages/mosaic
```

The optional 'command' parameter accepts the following values (default is "addDir"):

- addDir: recursively add a hierarchy of directories. The root directory is given by the 'data' parameter.
- addFile: add a single file, whose path is defined by the 'data' parameter.
- deleteFile: delete entries from a single file whose path is given by the 'data' parameter.
- deleteIndexingSystem: remove all the entries in this indexing system.
- updateIndexingSystem: update indexing system metadata (should be used after multiple deleteFile calls).
- deleteCoverage: delete a single coverage offering by name; the name being given in the 'data' parameter.

Note that in the procedure described here above, and if the indexing is done on top of a GML file, it's not necessary to setup manually a WFS and reference it in the WCS providers.fac. The only step needed is to reference directly the GML file in the "indexingProvider" parameter instead of the corresponding WFS providers.fac file. The "indexingType" parameter must be set at the value "GML".

We recommend you to use the ERDAS APOLLO Administration Console to configure all your providers.

Example of an Indexed Coverage Set Entry

```
<CREATE ID="ATLANTA_MOSAIC"
JCLASS="com.ionicsoft.wmtmap.provider.coverage.IndexProvider" >
  <PARAM NAME="name" VALUE="ATLANTA_MOSAIC"/>
  <PARAM NAME="title" VALUE="Aerial imagery over Atlanta"/>
  <PARAM NAME="abstract" VALUE="Aerial imagery over City of
Atlanta"/>
  <PARAM NAME="keywords" VALUE="atlanta,imagery,mosaic,aerial"/>
  <PARAMBLOCK NAME="contact">
    <PARAM NAME="Organization" VALUE="ERDAS, Inc."/>
    <PARAM NAME="AddressType" VALUE="Postal"/>
    <PARAM NAME="AddressBody" VALUE="5051 Peachtree Corners
Circle"/>
    <PARAM NAME="City" VALUE="Norcross"/>
    <PARAM NAME="PostCode" VALUE="30092-2500"/>
    <PARAM NAME="State" VALUE="GA"/>
    <PARAM NAME="Country" VALUE="USA"/>
    <PARAM NAME="Person" VALUE="Luc Donea"/>
    <PARAM NAME="Position" VALUE="Product Manager"/>
    <PARAM NAME="Voice" VALUE="+1 770 776 3400"/>
    <PARAM NAME="Fax" VALUE="+1 770 776 3500"/>
    <PARAM NAME="Email" VALUE="info@erdas.com"/>
    <PARAM NAME="OnlineResource" VALUE="http://www.erdas.com"/>
  </PARAMBLOCK>
  <PARAMBLOCK NAME="CoverageOffering">
    <PARAMBLOCK NAME="ECW">
      <PARAM NAME="name" VALUE="Atlanta layer"/>
      <PARAM NAME="title" VALUE="Atlanta Coverage Offering layer"/>
      <PARAM NAME="abstract" VALUE="Atlanta Coverage Offering layer,
Imagery data"/>
      <PARAM NAME="keywords" VALUE="Atlanta, Imagery, remote-
sensing, coverage, ECW"/>
    </PARAMBLOCK>
  </PARAMBLOCK>
  <PARAM NAME="indexingProvider"
VALUE="file:///C:/Erdas/ApolloServer/config/erdas-
apollo/providers/coverage/ATLANTA.gml" />
  <PARAM NAME="indexingServer" VALUE="ATLANTA" />
  <PARAM NAME="srs" VALUE="EPSG:2240" />
</CREATE>
```

Multi Simple Coverages

A set of coverage files, in a hierarchy of directories can be presented as a homogenous layer of data using the IndexProvider. However, a data manager might need to present these files as separate data sources.

For example, four files can be indexed and served using an IndexProvider. The user will then see one coverage offering but will not be able to distinguish from which file comes a data subset.

If these files are served using a MultiSimpleProvider, the user will then see four coverage offerings each one named with the name of the file without extension followed by an underscore and the original coverage name.

A Multi Simple Provider serving n-number of files can be seen as n-number of Simple Providers.

The configuration of a Multi Simple Provider is the same as the configuration of an Index Provider, except the JCLASS parameter.

Example of a MultiSimple Coverage Set Entry

```
<CREATE ID="ATLANTA_LIST"
JCLASS="com.ionicsoft.wmtmap.provider.coverage.MultiSimpleProvider" >
  <PARAM NAME="name" VALUE="ATLANTA_MOSAIC"/>
  <PARAM NAME="title" VALUE="Aerial imagery list over Atlanta"/>
  <PARAM NAME="abstract" VALUE="Aerial imagery list over City of Atlanta"/>
  <PARAM NAME="keywords" VALUE="atlanta,imagery,list,aerial"/>
  <PARAMBLOCK NAME="contact">
    <PARAM NAME="Organization" VALUE="ERDAS, Inc."/>
    <PARAM NAME="AddressType" VALUE="Postal"/>
    <PARAM NAME="AddressBody" VALUE="5051 Peachtree Corners Circle"/>
    <PARAM NAME="City" VALUE="Norcross"/>
    <PARAM NAME="PostCode" VALUE="30092-2500"/>
    <PARAM NAME="State" VALUE="GA"/>
    <PARAM NAME="Country" VALUE="USA"/>
    <PARAM NAME="Person" VALUE="Luc Donea"/>
    <PARAM NAME="Position" VALUE="Product Manager"/>
    <PARAM NAME="Voice" VALUE="+1 770 776 3400"/>
    <PARAM NAME="Fax" VALUE="+1 770 776 3500"/>
    <PARAM NAME="Email" VALUE="info@erdas.com"/>
    <PARAM NAME="OnlineResource" VALUE="http://www.erdas.com"/>
  </PARAMBLOCK>
  <PARAMBLOCK NAME="CoverageOffering">
    <PARAMBLOCK NAME="TIF">
      <PARAM NAME="name" VALUE="Grid_L2g_2d"/>
    </PARAMBLOCK>
  </PARAMBLOCK>
  <PARAM NAME="indexingProvider" VALUE="indexer.fac" />
  <PARAM NAME="indexingServer" VALUE="GML_COV" />
  <PARAM NAME="srs" VALUE="EPSG:4326" />
</CREATE>
```

The "CoverageOffering" parameter block has a different behavior in the case of a MultiSimple Provider. As the different Coverage Offerings are, by default, named using the following pattern: <file name>_<offering original name> a pattern-matching mechanism is used to rename the Coverage Offerings. For example:

The offering original name for Geotiff files is TIF. Therefore two files named file1.tif and file2.tif will be exposed as two coverage offerings named file1_TIF and file2_TIF. Using the above parameter block, they will be named file1_Grid_L2g_2d and file2_Grid_L2g_2d.

Hierarchical Coverages

When the amount of data becomes larger, better data management is required. The simplest and best data organisation model remains the simple hierarchical tree. That is why the WCS hierarchical provider enables users to expose their data in an hierarchical tree. This is far more powerful than the list model of the Multi Simple providers.



The hierarchical providers are not available in the ERDAS APOLLO Essentials product. The hierarchical provider is available in the ERDAS APOLLO Advantage/Professional product distribution.

The indexer is registry-object based, and the WCS provider references the registry through its providers.fac, and the WRS provider name. The indexing system WRS has to be set up on top of an Oracle database .

The hierarchical provider is able to:

- manage any type of data in a single instance. It is not unusual to visualize Hdf-Eos, GeoRaster, Jpeg2000 and Tif data under the nodes of the same tree.
- manage multiple coordinate systems in a single instance.
- compute dynamically the nodes metadata by aggregation of its children metadata.
- answer GetCoverage requests on the tree nodes by mosaicing the node children.

Example of an Hierarchical Coverage Set Entry

```
<CREATE ID="MODIS_TREE"  
JCLASS="com.ionicsoft.wmtmap.provider.coverage.HierarchicalProvider" >  
  <PARAM NAME="name" VALUE="MODIS_TREE"/>  
  <PARAM NAME="title" VALUE="hierarchical WCS"/>  
  <PARAM NAME="abstract" VALUE="MODIS Data"/>  
  <PARAM NAME="metacurl" VALUE="" />
```

```

<PARAM NAME="keywords" VALUE="Nasa,Terra,MODIS,MOD09GHK,Image
Archive"/>
<PARAM NAME="backgroundValue" VALUE="-1000" />
<PARAM NAME="srs" VALUE="EPSG:4326" />
<PARAM NAME="mode" VALUE="dynamic" />
<PARAM NAME="maxcache" VALUE="250" />
<PARAM NAME="maxStitch" VALUE="500" />
<PARAM NAME="gdalpath" VALUE="D:\resin-219\webapps\wcs\WEB-
INF\resource\GDAL\"/>
<PARAM NAME="hegpath" VALUE="D:\resin-219\webapps\wcs\WEB-
INF\resource\heg3\bin\ " />
<PARAM NAME="tmppath" VALUE="D:\resin-219\webapps\wcs\WEB-
INF\resource\temp\ " />
<PARAM NAME="indexingProvider"
VALUE="..\..\..\wrs\servlet\resource\providers.fac" />
<PARAM NAME="indexingServer" VALUE="WRSORA" />
<PARAM NAME="indexingType" VALUE="CATALOG" />
<PARAMBLOCK NAME="contact">
  <PARAM NAME="Organization" VALUE="ERDAS, Inc."/>
  <PARAM NAME="AddressType" VALUE="Postal"/>
  <PARAM NAME="AddressBody" VALUE="5051 Peachtree Corners
Circle"/>
  <PARAM NAME="City" VALUE="Norcross"/>
  <PARAM NAME="PostCode" VALUE="30092-2500"/>
  <PARAM NAME="State" VALUE="GA"/>
  <PARAM NAME="Country" VALUE="USA"/>
  <PARAM NAME="Person" VALUE="Luc Donea"/>
  <PARAM NAME="Position" VALUE="Product Manager"/>
  <PARAM NAME="Voice" VALUE="+1 770 776 3400"/>
  <PARAM NAME="Fax" VALUE="+1 770 776 3500"/>
  <PARAM NAME="Email" VALUE="info@erdas.com"/>
  <PARAM NAME="OnlineResource" VALUE="http://www.erdas.com"/>
</PARAMBLOCK>
</CREATE>

```

Oracle 10g GeoRaster Coverages

Oracle GeoRaster coverages can be served in various ways, depending on whether access to individual rows is expected or not. The following sections describe those different solutions.

The WCS Oracle GeoRaster Provider

The WCS GeoRaster provider (JCLASS=com.ionicsoft.wmtmap.coverage.GetRasterProvider) is very similar to the WMS RasterProvider. Except the JCLASS attribute, all the other parameters are the same.

Example of an Oracle GeoRaster WCS

```

<CREATE ID="TESTOGEOR"
JCLASS="com.ionicsoft.wmtmap.provider.coverage.GeoRasterProvide
r">
  <PARAM NAME="TITLE" VALUE="GeoRaster Simple WCS"/>
  <PARAMBLOCK NAME="LAYERS">
    <PARAMBLOCK NAME="0">
      <PARAM NAME="table" VALUE="DMTEST3"/>
    </PARAMBLOCK>
  </PARAMBLOCK>
</CREATE>

```

```

<PARAM NAME="SRS" VALUE="EPSG:26986"/>
<PARAM NAME="column" VALUE="GEORASTER"/>
<PARAM NAME="rasterdatatable" VALUE="RASTERTABLE3"/>
<PARAM NAME="id" VALUE="2"/>
<PARAM NAME="TITLE" VALUE="Layer0Title"/>
<PARAM NAME="ABSTRACT" VALUE="Layer0Abstract"/>
<PARAM NAME="NAME" VALUE="RASTER3_2"/>
</PARAMBLOCK>
<PARAMBLOCK NAME="1">
  <PARAM NAME="table" VALUE="DMTEST4"/>
  <PARAM NAME="SRS" VALUE="EPSG:26986"/>
  <PARAM NAME="column" VALUE="GEORASTER"/>
  <PARAM NAME="rasterdatatable" VALUE="RASTERTABLE4"/>
  <PARAM NAME="id" VALUE="1"/>
  <PARAM NAME="TITLE" VALUE="Layer1Title"/>
  <PARAM NAME="ABSTRACT" VALUE="Layer1Abstract"/>
  <PARAM NAME="NAME" VALUE="RASTER4_1"/>
</PARAMBLOCK>
</PARAMBLOCK>
<PARAM NAME="ABSTRACT" VALUE="My Georaster WCS Test"/>
<PARAM NAME="CONNECT"
VALUE="oracle://arcsde:1521/sid+test/user+RASTER/password+RASTER/defaultRowPrefetch+10"/>
</CREATE>

```

Inside the WCS SimpleProvider

The WCS SimpleProvider can be configured to serve Oracle GeoRaster data. This is mainly done through an extension to the PATH parameter and a set of GeoRaster-specific parameters. Those specific parameters are:

- **WCSProviderType:** It can be either **WCSGeoRasterProvider** or **WCSArcSdeProvider**
- **connect:** the connection string to the database or to the JNDI data source
- **table:** the table name holding the GeoRaster column(s)
- **name:** the name to give to the layer
- **column:** the GeoRaster column
- **id:** the row id
- **rasterdatatable:** the Raster Data Table containing the actual image
- **user:** the table owner (optional)

Those parameters can be either defined individually as <PARAM> elements, or grouped in the PATH parameter, or a mix. If set in the PATH parameter, the syntax is: (<name>=<value>)[(<name>=<VALUE>)]. Each parameter name is then case-sensitive.

Example of Individual Oracle GeoRaster Parameters

```
<CREATE ID="SIMPLE_OGEOR1"
JCLASS="com.ionicsoft.wmtmap.provider.coverage.SimpleProvider">
  <PARAM NAME="title" VALUE="ERDAS WCS server with GeoRaster"/>
  <PARAM NAME="abstract" VALUE="WCS over BOSTON imagery with
Oracle GeoRaster properties."/>
  <PARAM NAME="WCSProviderType" VALUE="WCSGeoRasterProvider" />
  <PARAM NAME="connect"
VALUE="oracle://geo.raster.com/user+myuser/password+mypwd/sid+M
YSID" />
  <PARAM NAME="table" VALUE="BOSTON_GEOASTER" />
  <PARAM NAME="name" VALUE="GEOR1" />
  <PARAM NAME="column" VALUE="georaster" />
  <PARAM NAME="id" VALUE="2" />
  <PARAM NAME="rasterdatatable" VALUE="rdt_boston" />
  <PARAM NAME="srs" VALUE="EPSG:26986" />
</CREATE>
```

The parameters can be grouped in a pseudo-path parameter, as soon as case-sensitivity (most of them must be lowercase) of parameter names is respected.

Example Grouped Oracle GeoRaster Parameters

```
<CREATE ID="SIMPLE_OGEOR2"
JCLASS="com.ionicsoft.wmtmap.provider.coverage.SimpleProvider">
  <PARAM NAME="title" VALUE="ERDAS WCS server with GeoRaster"/>
  <PARAM NAME="abstract" VALUE="WCS over BOSTON imagery with
Oracle GeoRaster properties."/>
  <PARAM NAME="path"
VALUE="(WCSProviderType=WCSGeoRasterProvider)

(CONNECT=oracle://arcsde:1521/sid+test/user+RASTER/password+RAS
TER/defaultRowPrefetch+10)

(table=DMTEST3) (name=RASTER) (column=GEORASTER) (id=2) (rasterdata
table=RASTERTABLE3)"/>
  <PARAM NAME="srs" VALUE="EPSG:26986" />
</CREATE>
```

Inside the WCS Index- or MultiSimple-Provider

The WCS Index Provider and MultiSimple Provider are serving data that are indexed in a WFS. As soon as indexing is achieved, the set of coverage offerings and layers exposed by the service depend on the data that have been indexed. For Oracle GeoRaster data to be indexed, the pseudo-path syntax described in the previous section fully applies. The "runwfsindexer" tool can be used as usual. The following example shows what parameters are necessary to successfully index an Oracle GeoRaster tile.

Example of Indexing a Oracle GeoRaster Tile

```
C:\Erdas\Apollo\tools\wcs> runwfsindexer -factory providers.fac
-name INDEX_GEOR -command addFile
-datapath
(table=DMTEST3) (column=GEORASTER) (id=2) (rasterdatatable=RASTERT
ABLE3)
(WCSProviderType=WCSGeoRasterProvider) (connect=oracle://arcsde:
1521/sid+test/user+RASTER/password+RASTER)
```

The Oracle GeoRaster tiles will be exposed as a single "GeoRaster" Coverage Offering and Layer. It can be overridden by setting the "name" parameter in a "CoverageOffering" PARAMBLOCK in the provider configuration.

HDF-EOS Coverages

This connector is deprecated but kept in the product and documentation of backward-compatibility purposes. Beware that it could disappear in a future release.

If the coverage data are in the HDF-EOS format, the framework will use the NASA HEG tool to parse the files and convert them into GeoTIFF. This HEG tool is free and can be downloaded from the web (see <http://eosweb.larc.nasa.gov/>).



The download and use of the HEG tool may be subject to restrictions that must be resolved with the providers of that tool. No assumption is made on the "right of use" of the tool in ERDAS products.

Obtain and install the HEG tool before configuring the provider. Do not forget to have the HEG environment variables set properly before starting the servlet engine or before running the command line scripts. Currently, the mandatory variables are PGSHOME, MRTDATADIR and MRTBINDIR (subject to change by NASA).

The configuration necessary to support this format is an addition to the standard configuration of the providers above (Simple, Index, Multi Simple or Hierarchical): The path to the HEG tool installation directory must be defined in the providers.fac file. This is done by defining the "hegpath" parameter. Moreover, as large temporary files are likely to be produced, add the "tmppath" parameter linking it to a directory for temporary files. If absent, the WCS uses the "hegpath" value as temporary directory.

Note that the framework makes the assumption that for each coverage, the metadata XML file is beside the HDF-EOS file, and has the same name plus a ".xml" extension. This metadata file, if present, will be converted "on-the-fly" into ISO 19115 in order to provide metadata information to the user.

The following example shows the configuration of a Simple Provider to handle a HDF-EOS coverage file. You will notice the "backgroundValue" parameter which contains the value by band that represents the absence of data. As an example, for images it is the background value that will be set as transparent. The value can be either a comma-separated list of values, one per band, or a single value that will apply to all bands.

Example of a HDF-EOS Simple Coverage Entry

```
<CREATE ID="MODIS"
JCLASS="com.ionicsoft.wmtmap.provider.coverage.SimpleProvider"
>
  <PARAM NAME="name" VALUE="MODIS"/>
  <PARAM NAME="title" VALUE="MODIS Coverage Server"/>
  <PARAM NAME="abstract" VALUE="MODIS Data over the USA"/>
  <PARAM NAME="metainfo"
VALUE="MOD09GHK.A2003189.h10v04.004.2003196000916.hdf.xml" />
  <PARAM NAME="keywords" VALUE="NASA,TERRA,MODIS,GRID"/>
  <PARAMBLOCK NAME="CoverageOffering">
    <PARAMBLOCK NAME="MOD_Grid_L2g_2d">
      <PARAM NAME="title" VALUE="MOD_Grid_L2g_2d data"/>
      <PARAM NAME="abstract" VALUE="MOD_Grid_L2g_2d data, one
granule"/>
    </PARAMBLOCK>
  </PARAMBLOCK>
  <PARAM NAME="hegpath" VALUE="/opt/heg/bin/" />
  <PARAM NAME="tmppath" VALUE="/opt/heg/tmp/" />
  <PARAM NAME="path"
VALUE="/home/Erdas/ApolloServer/data/coverages/heg/MODIS/MOD09G
HK.A2003189.h10v04.004.2003196000916.hdf" />
  <PARAM NAME="backgroundValue" VALUE="0" />
</CREATE>
```



It is possible to set up a Geotiff provider, starting from NASA HDF-EOS files. To do so, use the HEG tool to convert the HDF-EOS coverage into Geotiff. Additionally, ERDAS provides a Metadata Decoding tool to convert the NASA Metadata XML files into ISO 19115 XML files.

Pyramid Provider

For WMS connectors, it is possible to build a WCS Pyramid provider. The WCS pyramid provider behaves exactly the same way as the WMS pyramid provider. For more information, please see [Pyramid Provider Configuration](#).



All the proxied providers have to share the same coverage names, i.e. the same filenames, only the resolution should vary. In clear, it means a coverage name must be the same, which means the filename for base images at each decimation level must be the same.

The only differences in the Pyramid configuration are:

- The class that implements the WCS Pyramid Provider is `com.ionicsoft.wmtmap.provider.coverage.ScaleProvider` instead of `com.ionicsoft.wmtmap.provider.proxy.ScaleProvider`.
- The type of providers that can be proxied through the WCS Pyramid Provider, are the various WCS providers, i.e., `SimpleProvider`, `MultiSimpleProvider` or `IndexProvider`.

As with of the ERDAS WCS providers, the Pyramid Provider implements the WMS interface and can answer GetMap requests if the proper SLD rules are referenced in the Providers.fac configuration.

Below is an example of a WCS pyramid configuration:

```
<!-- The Pyramid Provider
This provider points to the other WCS providers defined in this
file -->
<CREATE ID="SCALE_WAL"
JCLASS="com.ionicsoft.wmtmap.provider.coverage.ScaleProvider">
  <PARAMBLOCK NAME="1">
    <PARAM NAME="minscale" VALUE="0"/>
    <PARAM NAME="maxscale" VALUE="50000"/>
    <PARAM NAME="id" VALUE="WAL4"/>
    <PARAM NAME="master" VALUE="true" />
  </PARAMBLOCK>
  <PARAMBLOCK NAME="2">
    <PARAM NAME="minscale" VALUE="50000.000001"/>
    <PARAM NAME="maxscale" VALUE="100000"/>
  </PARAMBLOCK>
</CREATE>
```



```

    <PARAM NAME="id" VALUE="WAL16"/>
  </PARAMBLOCK>
  <PARAMBLOCK NAME="3">
    <PARAM NAME="minscale" VALUE="100000.000001"/>
    <PARAM NAME="id" VALUE="WAL32"/>
  </PARAMBLOCK>
</CREATE>

<!-- Service exposing a layer named Wallonia -->
<CREATE ID="WAL4"
JCLASS="com.ionicsoft.wmtmap.provider.coverage.IndexProvider" >
  <PARAM NAME="name" VALUE="WAL4"/>
  <PARAM NAME="indexingProvider" VALUE="indexer.fac" />
  <PARAM NAME="indexingServer" VALUE="WAL4" />
  <PARAM NAME="srs" VALUE="31370" />
  <PARAM NAME="backgroundValue" VALUE="0" />
</CREATE>

<!-- Service exposing a layer named Wallonia -->
<CREATE ID="WAL16"
JCLASS="com.ionicsoft.wmtmap.provider.coverage.IndexProvider" >
  <PARAM NAME="name" VALUE="WAL16"/>
  <PARAM NAME="indexingProvider" VALUE="indexer.fac" />
  <PARAM NAME="indexingServer" VALUE="WAL16" />
  <PARAM NAME="srs" VALUE="31370" />
  <PARAM NAME="backgroundValue" VALUE="0" />
</CREATE>

<!-- Service exposing a layer named Wallonia -->
<CREATE ID="WAL32"
JCLASS="com.ionicsoft.wmtmap.provider.coverage.IndexProvider" >
  <PARAM NAME="name" VALUE="WAL32"/>
  <PARAM NAME="indexingProvider" VALUE="indexer.fac" />
  <PARAM NAME="indexingServer" VALUE="WAL32" />
  <PARAM NAME="srs" VALUE="31370" />
  <PARAM NAME="backgroundValue" VALUE="0" />
</CREATE>

```

GML Application Schema and Mapping to Databases

This chapter gives a detailed explanation on GML Application Schemas, and on how to configure vector data sources behind the wfs servlet.

- Concepts for GML Application Schemas
- Exposing GML Features in a WFS
- Definition of the configuration steps
- Feature Schema configuration
- Feature Mapping
- Additional configuration tasks

Introduction

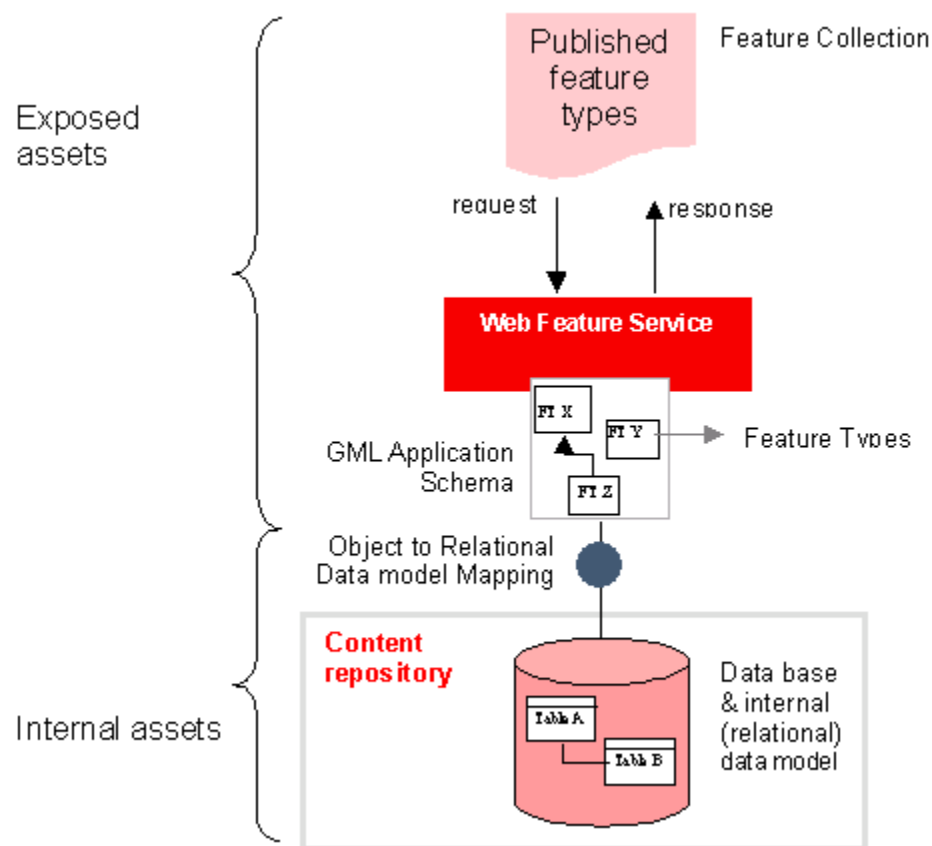
The Web Feature Server (WFS) makes it possible to expose data to the world, providing a view or an abstraction of the physical data. This is achieved through the definition of an Application Schema, that is interoperable, ISO-Compliant, and prevents users from gaining visibility into the nature of the computational infrastructure. For example, no one will know if the data is stored in data files, in a database or computed on the fly. A subset of the existing data can be presented while keeping confidential information out of reach. The exposed features are named Geospatial objects, and are a translation of the actual underlying data. The WFS responds to the requirements of a secure web service for interoperable exchange of geospatial objects, compliant with ISO, exposing one or multiple restricted views on an existing proprietary data resource.

Figure 1 below exposes the relationship between the data model used inside the content repository and the GML application schema exposed by the WFS. The relationship between those two models is achieved through a “mapping” mechanism which is configured at the time of setting up the service. The possibility of publishing a data model that is different from the internal one depends on the software used to deploy the nodes.

The mapping is the configuration that describes the link between the Feature Type definition and the objects stored in the underlying engine. The mapping document associated with a WFS presents the necessary information for the WFS to convert client requests to queries understood by the data server. It also converts the result into a compliant collection of features. Therefore, it makes the link between the internal data structure and the published information.

This chapter's purpose is to expose the different types of Feature Type relations, how to express these relations in a GML Application schema, how to implement them in a relational database and finally how to map this internal relational data model with the exposed object model.

Figure 125: Internal Data Model versus Exposed Feature Types



Key Concepts

To map an Application Schema onto a Database Model, it is necessary to understand clearly what each of those words mean. The scope of this section is to provide a short explanation.

Application Schema

The ISO 19109 standard "Rules for application schema" provides the following definition and purpose of an application schema:

An application schema provides the formal description of the data structure and content required by one or more applications. An application schema defines:

- The content and structure of data;
- Specifications of operations for manipulating and processing data by an application.

The purpose of an application schema is twofold:

- to provide a computer-readable data description defining the data structure, making it possible to apply automated mechanisms for data management;
- to achieve common and correct understanding of the data, by documenting the data content of the particular application field, and thereby making it possible to unambiguously retrieve information from the data.

ISO 19109 application schemas are defined in a conceptual schema language: the Universal Modeling Language (UML).

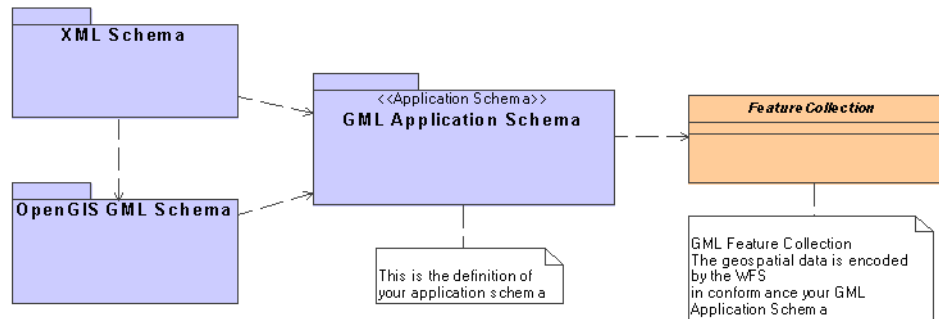
In the case of a geospatial application schema, the concepts of the General Feature Model (GFM) are mapped to the concepts of the UML conceptual language. The General Feature Model is a model of concepts required to classify a view of the real world expressed in UML. The objects that are classified are called features; relations between features are feature associations and inheritance. Feature Types have properties that are feature attributes, feature operations and feature association roles. The GFM is a meta model of feature types [ISO01b].

As an application schema deals with representing features, the structure of the GFM has to be kept in mind while creating the application schema. For more detailed information refer to the ISO19109 document.

GML Application Schema

A Geographic Markup Language (GML) application schema is an XML schema that describes one or more types of geographic objects [OGC03]. One can consider the GML application schema as a physical implementation of an ISO 19109 application schema. GML application schemas define the encoding of geospatial information. GML application schemas are used by Web Feature Servers (WFS) to encode geospatial data into GML and deliver it over the Web. User's Application Schemas have to extend the basic types defined in the GML Schema in order to be manageable by a WFS and be compliant with the GML specification (Figure below).

Figure 126: GML Schema Structure



Feature and Feature Type

A feature is an abstraction of real world phenomena. It is a geographic feature if it is associated with a location relative to the Earth. The state of a feature is defined by a set of properties. The number of properties a feature may have together with their names and types are determined by its type definition, the feature type.

An abstract feature type provides a set of properties common to several feature types. A concrete feature (by opposition to abstract) must derive from this type and may specify additional properties.

Mapping

GML Application Schemas are object-oriented while most databases are relational. The mapping consists of writing a correspondance between a GML Application Schema document and a Database Model, thus it is in an object-to-relational mapping.

The way the mapping is done is specific to a vendor. ERDAS writes the mapping in an XML document that can be built either using one of the command-line tools coming along with ERDAS APOLLO, or manually. In some cases, the mapping is implicit and does not need any document.

Configuration Overview

The ERDAS WFS is the component that delivers a Feature Service above geospatial engines. These engines can provide datasets on request. For example, imagine that a city planner has legacy data in Oracle about land parcels, the road network and point locations of trees in the city. The planner wants to publish this data online. Therefore, the planner needs to examine the following factors:

- Define which features to publish online for external users. For example, all of the roads in the city do not need to be exposed, especially private roads or maintenance roads. Therefore, expose only a subset of the information available.

- Express what attributes of a Feature to expose. There is a lot of information about roads - the length of the road, the type of road, what material the road is made out of and the age of the road. However, the users are only really interested in the name and types of road in the city. Therefore only expose the RoadName and RoadType as a subset of all the attributes available. This configuration is called the "FeatureType definition".
- Transform the information coming from the legacy data into the ERDAS WFS. This information expresses where the feature server gets the information. For example, the RoadName information will come from the field "ROAD_Naming" from table "Road" located in the Oracle database, MyCity. This part of the configuration is called "the mapping configuration".
- The source of the data and what connector is needed to get data from the legacy database into ERDAS servlets. Each legacy data type will have it's own ERDAS connector. However, for the same database, multiple connectors with specific behaviors may exist, as in the following example.

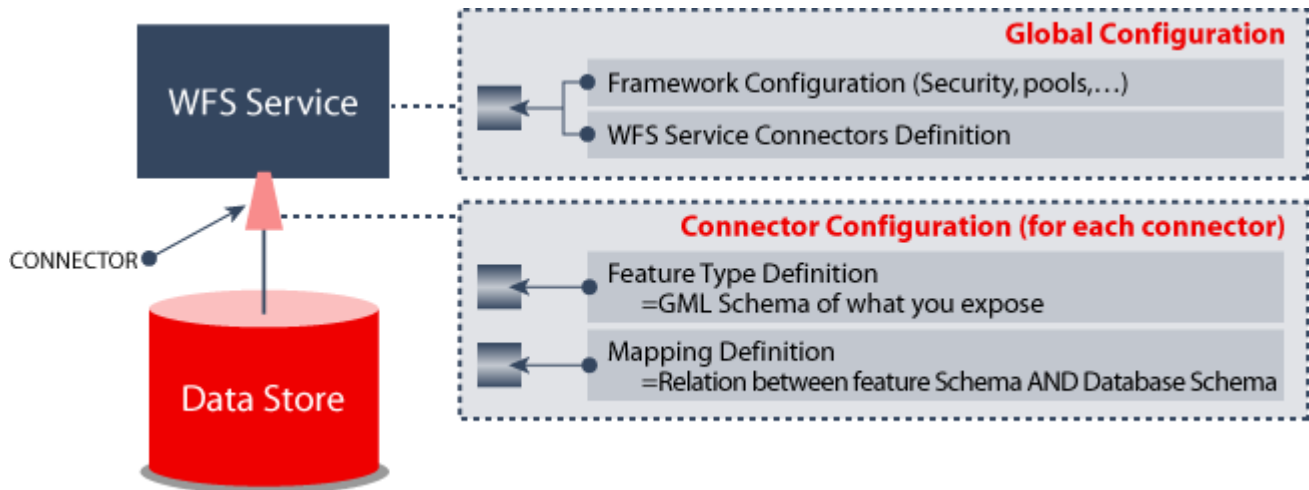
If accessing an Oracle database, there could exist an "Oracle OCI Connector" and an "Oracle thin Connector" using different technologies. Also, there might be a connector with some capabilities (e.g. Read Only) and in the future, maybe another connector that could also support transactions. This part of the configuration is called "the provider configuration".

So, to configure a WFS Service, the common configuration steps are the following:

1. Define a GML Application Schema definition of the feature types to expose.
2. Establish the corresponding mapping to the data store.(often a SQL mapping)
3. Configure the global framework parameters.
4. Indicate for each service the connectors to use in the providers file.

Done. The configuration can be tested.

Figure 127: WFS Configuration



The Feature type definition and the mapping definition can lie in the same file if needed, but it is recommended that they be maintained separately for ease of use.



For the following sections, it is necessary for the Administrator to have a basic knowledge of Java programming and a good understanding of the XML philosophy, structure and syntax. GML Application Schema notions are helpful.

Each of those three resources (`providers.fac`, feature type definition and mapping files) has to be reachable by the feature service, in one or more ways. The most common being via the CLASSPATH of the JVM running the feature services.



Common mistake: Remember that if modifying the feature type definition, the mapping needs to be updated too.

Feature Schema Configuration

The XML Schema associated with a WFS gives the GML Application Schema (structure) needed by the WFS to expose its feature types. The schema provides the type descriptions structure for each feature type and its properties.

GML Application Schema Construction

A XML Schema definition document should be provided for the feature types to be published as GML, either through a WFS or not. The syntax has to conform to the W3C XML Schema specification (see W3C [XML Schema](#)) and respect the GML constraints. This schema does not need to be built manually if it is provided by an authority. It could also be built with one of the tools provided in the ERDAS APOLLO distribution (see the Tools and Viewer chapter). Note that such a schema is independent from the vendor who provides an infrastructure to expose the features. Only the limitations of the vendor's product are to be considered.

Here's a sample of a GML feature Type named "Road".

Example:A Simple Feature Type Definition

```
<xsd:schema targetNamespace="http://www.erdas.com/wfs"
elementFormDefault="qualified" version="0.1">
<xsd:import namespace="http://www.opengis.net/gml"
schemaLocation="http://schemas.opengis.net/gml/3.1.1/base/feature.xsd"/>
  <xsd:element name="Road" type="wfs:RoadType"
substitutionGroup="gml:_Feature"/>
  <xsd:complexType name="RoadType">
    <xsd:complexContent>
      <xsd:extension base="gml:AbstractFeatureType">
        <xsd:sequence>
          <xsd:element name="streetName" type="string"/>
          <xsd:element name="centerLine"
type="gml:LineStringPropertyType"/>
        </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>
</xsd:schema>
```

The Steps to Construct the Feature Type Schema

These steps aim to help manually build the GML Application Schema for the information to publish. This information must be expressed in terms of feature types and properties of each feature type.



The following steps do not apply if a mapping of type "SQL" is applied. In that case, the feature type schema is created on-the-fly based on the database schema.

See the "Feature Mapping" section below for a complete explanation of the types of mappings available, and which one to use depending on the case.

1. Decide the names of the features types to publish.

The first task is to build the list of feature type names to publish. It often closely correlates with relational table names or entities of the real world (building numbers, land usage types, tree locations).

2. Decide the names, types and cardinalities of the properties.

For each of the feature types, decide which properties will be visible. These can be a sub-set of the existing data properties, or a super-set where calculated properties are defined, or a 1 to 1 matching with actual properties.

For each property, figure out its type (integer, string) and cardinality. It is strongly recommended to reuse existing types as often as possible. This is because when mentioning existing schema's in <import> clauses, the generic type "xsd:string" given to the property "STATION" can be replaced by a type (fictive):"ceos:StationType". The "ceos" prefix has to be defined in the header of the schema, with a syntax like:

```
xmlns:ceos="http://www.ceos.org/ceos" and this schema must be included in the deployment, with a clause like: <xsd:import namespace = "http://www.ceos.org/ceos" schemaLocation = "ceos.xsd" />.
```

For geometric properties, also analyse existing data to determine their type, and decide what WFS geometric property it matches. The most common geometries are Point, LineString (List of points that defines a line), Polygon and Text. Compound properties like MultiLineString, MultiPolygon are also supported.

If a property is a collection of other feature types, this feature type must be defined recursively and the proper type name set in the main feature type definition.

3. Decide the optionality of the properties. If a property can be empty, the server will either output it with an empty value, avoid to output it at all, or output it with no value. If the property has to be output, the attribute minOccurs="1" should be added to the schema. If it has to be output, even as a null string (looking like: <Prop1/>), the nillable="true" attribute should be added.

Note: the minOccurs="1" attribute should be added if it is expected that users will request output to the ShapeFile format (option outputFormat=SHAPE in the GetFeature request). If not, ShapeFile readers could fail due to absent properties in some features.

4. Encode the feature type and property definitions in the XML Schema encoding

Edit the document manually with a text editor or by using an XML editor. It is suggested use one of the sample XML Schemas provided with the product and replacing the sections with the specific information.

5. Mention the schema file in the providers.fac associated to the service.



Control the Schema correctness and GML FeatureType definition:

- Check whether the schema is syntactically correct by using any third-party commercial tool that performs XML and XML Schema validation (xalan or saxon).
- To see if the feature type schema is correct, wait until the WFS service is fully configured, and ask for a DescribeFeatureType on all the feature types of the WFS. It should provide an answer that matches the definitions.

Feature Mapping

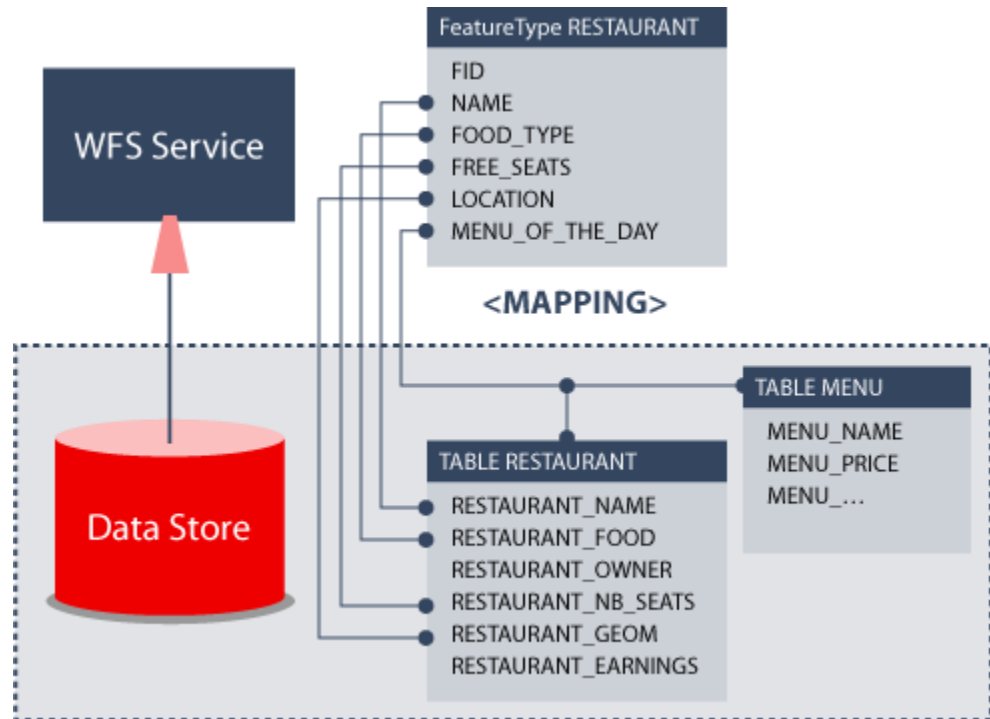
This chapter introduces the key aspects of the mapping configuration:

- What is mapping
- What are the different types of mapping
- How to choose what mapping to use
- Explicit Mapping
- SQL Mapping
- Automatic Mapping
- Relational (Explicit) Mapping
- Mapping of Enumerations
- Other Mappings

Mapping Concepts

The mapping is the configuration that describes the way ERDAS WFS achieves the link between the FeatureType definition and the objects returned by the underlying engine. The mapping document associated with an ERDAS WFS presents the necessary information for the WFS to convert client requests into queries understood by the data server. It also converts the result into a compliant collection of features. Therefore, it makes the link between the internal data structure and the published information.

Figure 128: WFS Mapping



The mapping file, written in XML, defines how the features types declared in the XML schema document are mapped onto the underlying data server. The mapping file name is either referenced by the "MAPPING" parameter of the provider (see the "Provider Configuration" chapter) or the mapping information is in the feature schema file.

The mapping information is contained in several tags: <MAPPING> is the main one. The <INFO>, <EXPORT>, <COLLECTION> and <OPTION> tags can be used to provide additional information such as Spatial Reference Systems, Bounding Box and additional dimensions.

Mapping Methodology

To map a set of Feature Types to database tables and relations, we propose several ways to achieve the mapping. Each applies in given situations and this section provide guidance in choosing the appropriate type of mapping.

Types of Mapping

The mapping can be explicit or implicit. If the mapping is explicit, then the file will contain all the information needed by the connector to manage the mapping. This is the case for the "Explicit Mapping" where for each element of the FeatureType, there will be an explicit link to an attribute of a table or a view. The mapping can also be implicit and completely managed by the connector or the framework itself. This is the case when the mapping is done by the code of the connector based on internal hypothesis or knowledge.

- **Explicit Mapping:** Explicit mapping is where a fully specified details of the feature are mapped to a database. With this mapping, define the mapping by hand in an explicit way, or use a tool to build it from the database model or from the GML Schema.
- **SQL Mapping:** The SQL mapping allows taking a table or multiple tables and asking the framework to make a 1:1 mapping from the table to a dynamically created feature. Implementing this, makes going from Table to Feature simple. There are no feature types to define and no complex mapping to do. The mapping from the tables to the feature type is implicit and managed by the framework.
- **Automatic Mapping:** This mapping is somehow the reverse of the SQL mapping. In automatic mapping, define a FeatureType in GML Application Schema and then submit the FeatureType to the engine that will create all the tables needed to support. The mapping from the FeatureType to the tables is implicit and managed by the framework.
- **Relational (Explicit) Mapping:** It is an explicit mapping, but where several tables are mapping onto a single feature type with complex properties.
- **Auto-Generated Mapping:** This mapping, derived from the Automatic Mapping, permits generating the mapping file instead of having the framework manage it. It allows a user to make changes in the mapping to fit the database model.
- **XML Enumeration and Relational Enumeration Mappings:** Allows mapping of enumerated values to an XML list or a table.

How to Choose a Mapping

To find out which mapping is best, look at these tips:

- If there are simple tables or Shapefiles be to publish in a minute, use the SQL mapping.

- If there are simple tables or Shapefiles, but well formatted, well named or retyped attributes are to be exposed, use the explicit mapping. Use one of the "runfromsqlxxx" script to build the schema and mapping files automatically from the data source.
- If there are complex GML FeatureTypes and the database is not yet built, use the Automatic mapping to create the tables.
- If there are a set of tables whose columns should appear in a single GML Feature Type, define a Relational Explicit Mapping.
- If there are a complex GML Application Schema with some of the data types being abstract in the GML Specification, the Auto-Generated Mapping is required to build the tables from the schema, and to be able to substitute those abstract types with real ones.
- If there are a set of feature types with enumerated values for some of the properties use the XML Enumeration Mapping if the values are listed in the schema or the Relational Enumeration Mapping if those values are stored in a table.

Mapping Tags Description

The mapping file is made of XML elements, named <MAPPING>, <INFO>, <EXPORT>, <COLLECTION> and <OPTION>. Please refer to the "Mapping tags" appendix for the list of XML tags that can appear in the mapping section document.

The <MAPPING> Tag

This tag describes the link between the FeatureType definition and the objects returned by the underlying engine.

See the "Feature Mapping Tags" Appendix for a complete description of the sub-tags or the next section for step-by-step building of the mapping.

The <INFO> Tag

As the data store may not provide all the information necessary for the service to be OGC-compliant, this tag contains additional information to complete it.

Spatial Reference Systems, Bounding Box, additional dimensions, allowed operations are some of the possible entries.

See the "Feature Mapping Tags" Appendix for a complete description of the sub-tags.

The <EXPORT> Tag

This tag defines which FeatureTypes are really presented to the world.

See the "Feature Mapping Tags" Appendix for a complete description of the sub-tags.

The <COLLECTION> Tag

This tag renames the root element produced by a getFeature request. The default is "featureCollection" but may be renamed if needed.

See the "Feature Mapping Tags" Appendix for a complete description of the sub-tags.

The <OPTION> Tag

This tag sets properties that apply to all feature types or just those related to the requests. Settings like output image resolution or Transactions response can be defined.

See the "Feature Mapping Tags" Appendix for a complete description of the sub-tags.

Explicit Mapping Definition Steps

Also named "specific mapping" this mapping makes no assumptions on the correspondance between the feature type structure and the actual data. Create the GML schema definition of the feature type (see previous chapter). For each of the properties of the feature type, mention in the mapping document which piece of data corresponds. The following section describes all the possible tags accepted in the mapping document. Most of them apply in the explicit mapping.



All of the connectors support this type of mapping.

This description explains how to fill the XML mapping document when the data source supports SQL requests and the mapping is explicit.



For non-relational or specific data sources, please call ERDAS to obtain the appropriate guide.

Follow these steps:

1. Build a <Mapping> ... </Mapping> block that contain the mapping tags, and the additional information.

2. Build one `<SQL> ... </SQL>` block and one `<Info> ... </Info>` block per feature type defined in the XML Schema.
3. The `<SQL>` element must have a "name" attribute whose value must be the name of the feature type (case-sensitive) preceded by the namespace prefix declared in the XML Schema header (commonly "wfs:").

The SQL value can also have a "generation" attribute with the value "specific" that notes that this type of mapping is used. This is the default value.

4. If a data server table is mapped onto a feature type, the `<Table>` element must contain the name of that table.
5. If a data server table is mapped onto a feature type, create as many `<Element>` entries as there are columns to map with a property.
6. Each feature type has a "fid" attribute that uniquely identifies it. The `<Primary>` element is required and it must contain the name of the primary key column in the data server table.
7. Other elements can be added in the `<SQL>` section. They are listed and described in the previous section.
8. In the `<Info>` section, a "name" attribute must be defined with the same rule as for the `<SQL>` element described above.
9. There must be at least one `<SRS>` that identifies the reference system of the data store. If more than one SRS is to be published in the capabilities, add all of them in the `<SRS>` tag separated by spaces. Being aware that the first item of the list is seen as the internal one.
10. Put at least one `<BoundingBox>` element giving the extents of the data.
11. If more request types are required beyond those supported by the Basic WFS, add an `<Operations>` element that lists the supported operations. See previous section for a complete list of operations.

Mention the mapping file in the providers.fac, associated to the service (see "Providers Configuration" section).

Example:Sample Explicit Mapping File

```
...
  <Mapping>
    <SQL name="wfs:RoadType">
      <Table nameSQL="ROAD"/>
      <Primary nameSQL="ROAD_ID" type="int"/>
      <Element name="wfs:streetName" nameSQL="NAME"/>
      <Element name="wfs:centerLine" nameSQL="GEOMETRY"/>
    </SQL>
  </Mapping>
```

```

<Info name="wfs:RoadType">
  <SRS>EPSG:4326</SRS>
  <BoundingBox SRS="EPSG:4326" minx="-180." miny="-90."
                maxx="180." maxy="90." />
  <Operations>Query</Operations>
</Info>
<Export generation="exportOnly">
  <Add name="wfs:RoadType"/>
</Export>
</Mapping>

```

SQL Mapping Definition Steps

This type of mapping allows a much lighter configuration, as the mapping manager achieves a one-to-one conversion from the SQL schema to the feature type schema. So, NO GML Schema is necessary when this mapping type is used. This type of mapping allows building the feature type schema from the database schema. Simply define the name of the table to publish and the framework does the job of building the feature type description. (Note: For non-relational or specific data sources connectors, the mapping may be implicit or explicit with specific behavior. Please call ERDAS to obtain the appropriate guide.) When this type of mapping is used, the <SQL> element can only contain a <Primary> property. None of the other tags belonging to the <SQL> section is allowed.



All vector connectors, except the "GML" connector, support this type of mapping.



It is also useful to note that, in the particular case of the Sql mapping, the table name provided can also be a pattern. The widest pattern is "%", and will lead to the mapping of each of the tables accessible by the user onto a feature type. This means that all tables found, including System tables, may be matched. This can be restricted either by giving a more restrictive pattern, for example "ADMIN%", or by adding the "schema" attribute to the <SQL> element to restrict its scope to the given database schema (or user). An alternative is to do an explicit mapping and adding the "user" attribute in the <Table> element.

For SQL mapping, follow these steps:

The process followed by the mapping manager is to analyze the SQL metadata and build the feature type schema accordingly. The minimum information to provide is the name of the table to map, and the script 'generation="sql' so that the manager knows it has to do the mapping. However, additional explicit information can be provided, like the primary key column name, so that some flexibility is retained.

1. Build a `<Mapping> ... </Mapping>` block that will contain the mapping tags, and the additional information.
2. Build one `<SQL> ... </SQL>` block and one `<Info> ... </Info>` block per feature type to define.
3. The `<SQL>` element must have a "name" attribute whose value must be the name of the table found in the database schema, or a pattern. The widest pattern is "%" and will lead to the mapping of each of the tables accessible by the user onto a feature type. Beware that if the user has privileges to access system tables, they will also be mapped.
4. It must also have a "generation" attribute, with the value "sql".
5. Each feature type having a "fid" attribute which uniquely identifies it. A `<Primary>` element must be created that contains the name of the primary key column in the data server table. If no `<Primary>` tag is defined, the "fid" value is built randomly and no persistence is insured for this value.
6. No other element can be added in the `<SQL>` section.
7. When starting the `<Info>` section, the element must have a "name" attribute whose value must be the name of the feature type (case-sensitive, equivalent to the name of the table) preceded by the namespace prefix "wfs:".
8. In the `<Info>` section, include at least one `<SRS>` element identifying the reference system of the data store. If publishing several SRSes in the capabilities, add all of them in the `<SRS>` tag separated by spaces. Being aware that the first item of the list is seen as the internal one.
9. Put at least one `<BoundingBox>` element giving the extents of the data.
10. If extending the request types supported beyond those of a Basic WFS, add an `<Operations>` element that lists the supported operations. See previous section for a complete list of operations.
11. Mention the mapping file in the providers.fac associated to the service (see "Providers Configuration" section).

Example:Sample SQL Mapping File

```

<Mapping>
  <SQL name="%" generation="sql" >
  </SQL>
  <Info>
    <SRS>EPSG:4326</SRS>
    <BoundingBox SRS="EPSG:4326" minx="-180" miny="-90"
      maxx="180" maxy="90" />
  </Info>
  ...
</Mapping>

```



The distribution contains a sample SQL mapping file, named `generic_sql_mapping.xml`, located in the same directory as the `WFS providers.fac` file. Use it as a template for the own mapping.

Automatic Mapping Definition Steps

This type of mapping is the way to convert a given feature schema onto a database schema based on feature type definitions. To achieve this mapping, it is first necessary to use the "Schema Generator" tool that builds a SQL script to generate the tables. As the mapping manager uses the same logic as the schema generator tool, the mapping is done automatically at run time. Note that this type of mapping still allows mapping rules since these rules are used when generating the SQL script. For example, a mapping rule can tell if a sub-collection is stored as a Blob, binary large object, or if it is expanded in each of its components in the data store. When the XML Schema document describing the feature types is the starting point, and the database structure can be created from it, this type of mapping applies.

Automatic mapping is also a convenient way to set-up storage for complex feature types. This mapping implies a two - step process: First, run the "Schema Generator" tool provided with the distribution. As entry parameters, it takes the name of the XML schema, and, optionally, the name of the mapping document. The output is a SQL script for the database schema generation (tables, indexes, or sequences). Then, at run time, the framework does the mapping automatically. In term of mapping directives, the <SQL> element must mention the attribute "generation" with the value "auto". Connectors supporting this type of mapping are Oracle Thin, Oracle OCI and PostgreSQL.

Follow these steps:

1. Take the XML schema file or build it according to GML rules.
2. Build a mapping file giving some mapping rules and input the value "auto" into the "generation" attribute.
3. Run the Schema Generator to obtain the SQL script.
4. Run this SQL script once to generate the tables.
5. Run the Schema Generator again, with the -delete option, to obtain the deletion SQL scripts. Backup the produced script, allowing for later removal of the tables. This script is useful when the number of tables is large.
6. If the WFS manages transactions, make sure the LOCKTIMEOUT table is created, as it will store locking information along with the "LK" field generated by the tool. The SQL script to generate the LOCKTIMEOUT table can be found in the distribution CD under `data/db/lock.sql` .



You can name the Lock-Id field (predefined as "LK") differently, as soon as you add a <Lock> tag in the mapping file to tell the service what field is used to manage locking.



If you do not want to manage transactions, it could be necessary to explicitly disable the "Lock" operation in the <Operations> tag for your feature type(s).

7. At this stage, insert and then retrieve a feature to ensure the structure corresponds to the expectations. If not, modify the schema file or the mapping rules, run the del.sql script to remove the tables, and return to step 3.

Example: Sample Automatic Mapping File

In the following example, the feature type is named AnnotationListType, defined in an XML Schema whose prefix has been set to "xima". The <Element> items in the mapping allow restricting the type of values in the given properties. The <Info> tag contains an <Operations> element that explains which types of requests are allowed on this feature type. The "*" sign means ALL and corresponds to "Insert, Delete, Update, Lock and Query". Note that the sample is for illustration purposes only and, therefore, the additional information necessary is not provided (such as "xima" schema, "iap" schema, and the definitions of the other feature types mentioned in the example).

```
<Mapping>
  <SQL name="xima:AnnotationListType" generation="auto">
    <Element name="xima:Metadata">
      <Type name="iap:AnnotationMetadataType"/>
      <Type name="iap:ImageMetadataType"/>
    </Element>
    <Element name="xima:Content">
      <Type name="iap:SimpleContentType"/>
    </Element>
    <Element name="xima:ImageReference">
      <Type name="iap:ImageURLType"/>
    </Element>
  </SQL>
  <Info name="xima:AnnotationListType">
    <Operations>*</Operations>
    <SRS>EPSG:4326</SRS>
    <BoundingBox SRS="EPSG:4326" minx="-180." miny="-90."
      maxx="180." maxy="90." />
  </Info>
  ...
</Mapping>
```

Relational (Explicit) Mapping Definition Steps

It starts with defining a simple Explicit Mapping for each of the tables to map with one feature type per table.

Then, changes have to be done like described below in order to map the relations.

The first case described is for a composition where a single feature type gets its properties from a nested table. Another case is the association where several feature types have their own existence but are related (like the Parcel/Person relationship).

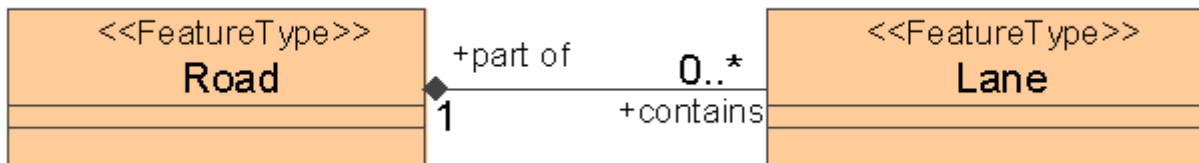
By definition, "The aggregation is a special type of association used to represent a "part of" relationship between two feature types. A key indicator of an aggregate relationship is feature types that share a lifetime. If the containing feature type is destroyed the contained feature type is destroyed with it."

And "A composition is the strongest relationship between two feature types. Composition is a special form of aggregation that indicates not only lifetime association but typically exclusive containment as well."

Composition

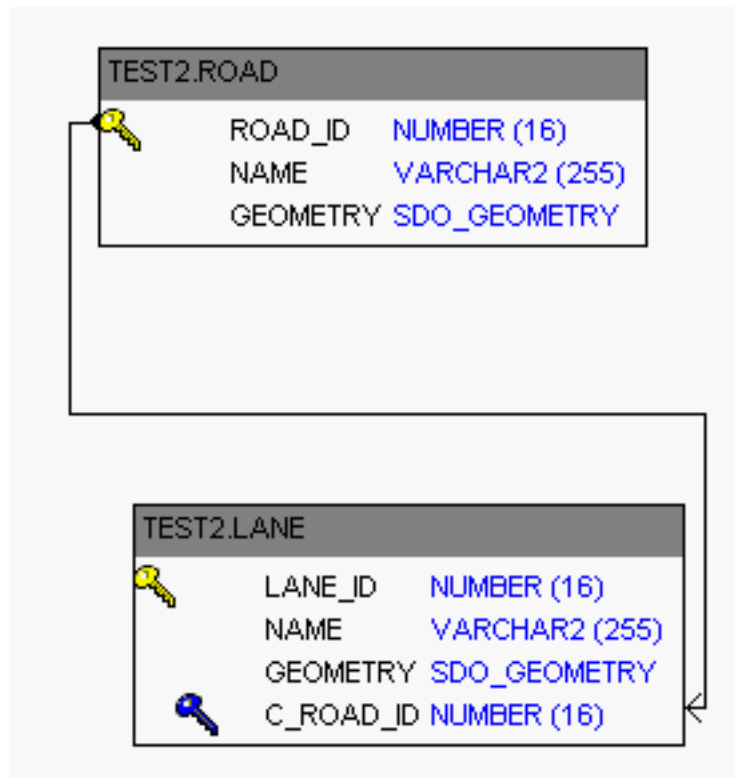
The explanation is based on the Road vs. Lane relationship where a Road is composed of one or more Lanes as shown in the UML diagram below.

Figure 129: Road-Lane UML Diagram



It is assumed that the underlying database is Oracle and the relationship between the ROAD and LANE tables is based on a primary key - foreign key relationship as in the diagram below.

Figure 130: Road-Lane Relational Diagram



As soon as an Explicit Mapping has been produced for the Road and Lane feature types, there should be a GML Application Schema with those two feature types defined, as in the "Simple Feature Type Definition" example above. Note that the GML3 rules are being used to define complex properties. Avoid relying on the old-fashioned GML2 syntax. Review the "Moving to GML3" section below for guidelines on how to get familiar with GML3 and how to migrate GML2 schemas to GML3.

Example: Road and Lane Feature Types

```
<xsd:schema targetNamespace="http://www.erdas.com/wfs"
elementFormDefault="qualified" version="0.1">
<xsd:import namespace="http://www.opengis.net/gml"
schemaLocation="http://schemas.opengis.net/gml/3.1.1/base/feature.xsd"/>
  <xsd:element name="Road" type="wfs:RoadType"
substitutionGroup="gml:_Feature"/>
  <xsd:complexType name="RoadType">
    <xsd:complexContent>
      <xsd:extension base="gml:AbstractFeatureType">
        <xsd:sequence>
          <xsd:element name="streetName" type="string"/>
          <xsd:element name="centerLine"
type="gml:LineStringPropertyType"/>
        </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>
```

```

</xsd:complexType>
<xsd:element name="Lane" type="wfs:LaneType"
substitutionGroup="gml:_Feature"/>
<xsd:complexType name="LaneType">
  <xsd:complexContent>
    <xsd:extension base="gml:AbstractFeatureType">
      <xsd:sequence>
        <xsd:element name="geometry"
type="gml:PolygonPropertyType"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

```

In this example, the Road feature type has a "centerLine" geometric property and a "streetName". Of course, as it inherits from gml:AbstractFeatureType, it has a set of additional properties like "gml:name" and "gml:description". Those properties will appear in the mapping file if they are mapped with a table column in the database.

The Lane feature type has a Polygon-type geometric property, and the gml:name inherited property will be mapped.

To explicitly map those feature types onto SQL tables, a mapping file similar to the "Sample Explicit Mapping File" example above is needed. That same mapping extended to include the Lane feature type will look like the example below.

Example:Road and Lane Mapping

```

...
<Mapping>
  <SQL name="wfs:Road">
    <Table nameSQL="ROAD"/>
    <Primary nameSQL="ROAD_ID" type="int"/>
    <Element name="wfs:streetName" nameSQL="NAME"/>
    <Element name="wfs:centerLine" nameSQL="GEOMETRY"/>
  </SQL>
  <SQL name="wfs:Lane">
    <Table nameSQL="LANE"/>
    <Primary nameSQL="LANE_ID" type="int"/>
    <Element name="gml:name" nameSQL="NAME"/>
    <Element name="wfs:geometry" nameSQL="GEOMETRY"/>
  </SQL>
</Mapping>
<Info name="wfs:Road">
  <SRS>EPSG:4326</SRS>
  <BoundingBox SRS="EPSG:4326" minx="-180." miny="-90."
maxx="180." maxy="90." />
  <Operations>Query</Operations>
</Info>
<Info name="wfs:Lane">
  <SRS>EPSG:4326</SRS>
  <BoundingBox SRS="EPSG:4326" minx="-180." miny="-90."
maxx="180." maxy="90." />

```

```

    <Operations>Query</Operations>
  </Info>
  <Export generation="exportOnly">
    <Add name="wfs:Road"/>
    <Add name="wfs:Lane"/>
  </Export>
</Mapping>

```

In that mapping example, notice that the mapped elements are the feature type names (Road and Lane) and not the complex type names (RoadType and LaneType) . This is a constraint relating to GML3 model. For GML2 feature types, the complex type names have to be mapped.

Now, express each road as composed of one or more lanes. First extend the GML schema to add a "hasLanes" property (cardinality 1..n) to the RoadType type and define the "HasLanesType" type that will hold the Lane features. The schema become like in the example below.

Example:Related Road and Lane Feature Types

```

<xsd:schema targetNamespace="http://www.erdas.com/wfs"
  elementFormDefault="qualified" version="0.1">
  <xsd:import namespace="http://www.opengis.net/gml"
  schemaLocation="http://schemas.opengis.net/gml/3.1.1/base/feature.xsd"/>
  <xsd:element name="Road" type="wfs:RoadType"
  substitutionGroup="gml:_Feature"/>
  <xsd:complexType name="RoadType">
    <xsd:complexContent>
      <xsd:extension base="gml:AbstractFeatureType">
        <xsd:sequence>
          <xsd:element name="streetName" type="string"/>
          <xsd:element name="centerLine"
  type="gml:LineStringPropertyType"/>
          <xsd:element name="hasLanes" type="wfs:HasLanesType"
  minOccurs="1" maxOccurs="unbounded"/>
        </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>
  <xsd:complexType name="HasLanesType">
    <xsd:sequence>
      <xsd:element ref="wfs:Lane"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:element name="Lane" type="wfs:LaneType"
  substitutionGroup="gml:_Feature"/>
  <xsd:complexType name="LaneType">
    <xsd:complexContent>
      <xsd:extension base="gml:AbstractFeatureType">
        <xsd:sequence>
          <xsd:element name="geometry"
  type="gml:PolygonPropertyType"/>
        </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>

```

```

    </xsd:complexContent>
  </xsd:complexType>

```

The next step consists in expressing the relation in the mapping file and mapping it to the database tables and foreign keys. The relation itself will be modeled using a <Relation> element in the <Mapping> tag as defined in the "Feature Mapping Tags" appendix. For our example, the content of this tag will be:

```

<Relation sourceType="wfs:Road"
  targetType="wfs:Lane"
  source="wfs:hasLanes"
  targetSQL="C_ROAD_ID"
  ownership="true"
/>

```

"wfs:Road" and "wfs:Lane" are respectively the parent and child ends of the relation. "wfs:hasLanes" is the source property name. "C_ROAD_ID" is a column of the "LANE" table that models the target field of the relation and the source field being the ROAD_ID in the ROAD table. The "ownership" set to true means that when a Road feature is deleted and that, the underlying Lanes will be removed as well.

Finally, extend the mapping of the wfs:Road type to tell that it is involved in a relation. This is done first by changing its "generation" attribute from its default value to "relational" and second by assigning it a "wfs:hasLane" property which is the source of the relation, through the ROAD_ID key in the ROAD table. The mapping file now looks like below.

Example:Road and Lane Relational Mapping

```

...
  <Mapping>
    <SQL name="wfs:Road" generation="relational">
      <Table nameSQL="ROAD"/>
      <Primary nameSQL="ROAD_ID" type="int"/>
      <Element name="wfs:streetName" nameSQL="NAME"/>
      <Element name="wfs:centerLine" nameSQL="GEOMETRY"/>
      <Element name="wfs:hasLanes" nameSQL="ROAD_ID" />
    </SQL>
    <SQL name="wfs:Lane">
      <Table nameSQL="LANE"/>
      <Primary nameSQL="LANE_ID" type="int"/>
      <Element name="gml:name" nameSQL="NAME"/>
      <Element name="wfs:geometry" nameSQL="GEOMETRY"/>
    </SQL>
    <!-- Relation between Road and Lane -->
    <!-- Note that C_ROAD_ID is a column of LANE -->
    <Relation sourceType="wfs:Road"
      targetType="wfs:Lane"
      source="wfs:hasLanes"
      targetSQL="C_ROAD_ID"

```



```

        ownership="true"
    />
</Mapping>
<Info name="wfs:Road">
  <SRS>EPSG:4326</SRS>
  <BoundingBox SRS="EPSG:4326" minx="-180."
miny="-90." maxx="180." maxy="90." />
  <Operations>Query</Operations>
</Info>
<Info name="wfs:Lane">
  <SRS>EPSG:4326</SRS>
  <BoundingBox SRS="EPSG:4326" minx="-180."
miny="-90." maxx="180." maxy="90." />
  <Operations>Query</Operations>
</Info>
<Export generation="exportOnly">
  <Add name="wfs:Road"/>
<!-- We do not expose wfs:Lane anymore -->
</Export>
</Mapping>

```

Below is a sample GML output from this WFS for the Road feature type. It is clear that one or more Lane types are linked to the parent Road.

Example:Road and Lane Sample GML3 Output

```

<wfs:Road id="RoadType.1">
  <gml:streetName>Main Street</gml:streetName>
  <wfs:centerLine>
    <gml:LineString srsName="EPSG:4326">
      <gml:pos>-112.09332 33.5698333</gml:pos>
      <gml:pos>-111.95549 33.666975</gml:pos>
      <gml:pos>-111.76598 33.480425</gml:pos>
      <gml:pos>-111.90496 33.3826028</gml:pos>
      <gml:pos>-112.09332 33.5698333</gml:pos>
    </gml:LineString>
  </wfs:centerLine>
  <wfs:hasLanes>
    <wfs:Lane>
      <gml:name>this is a Lane belonging to Road 1</gml:name>
      <wfs:geometry>
        <gml:Polygon srsName="EPSG:4326">
          <gml:exterior>
            <gml:LinearRing srsName="EPSG:4326">
              <gml:pos>-116.09332 33.5698333</gml:pos>
              <gml:pos>-111.95549 33.666975</gml:pos>
              <gml:pos>-111.76598 33.480425</gml:pos>
              <gml:pos>-111.90496 33.3826028</gml:pos>
              <gml:pos>-116.09332 33.5698333</gml:pos>
            </gml:LinearRing>
          </gml:exterior>
        </gml:Polygon>
      </wfs:geometry>
    </wfs:Lane>
  </wfs:hasLanes>
</wfs:Road>

```

Association

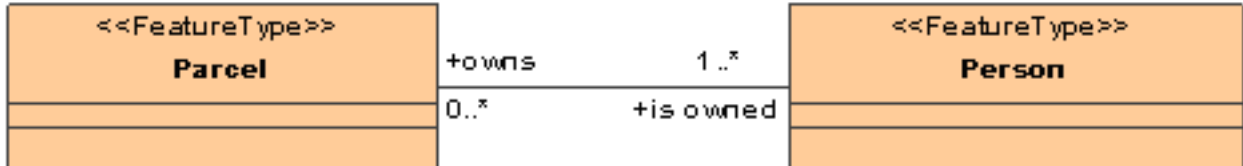
We use associations (which are neither aggregations nor compositions) when there is no ownership between the object types, and when the relation is 1..many or many..many. It will be modeled with an intermediary table, like in the example below.

The example describes the ownership relation between Parcels and Persons, like illustrated in the diagram below. A person can own zero, one or many parcels while a parcel belongs to at least one or many persons. If a person owning a parcel dies the parcel continues to exist, only the relation between that specified person and the parcel will disappear. Also once a parcel is deleted (because of a split or merge for instance), the person continues to exist.



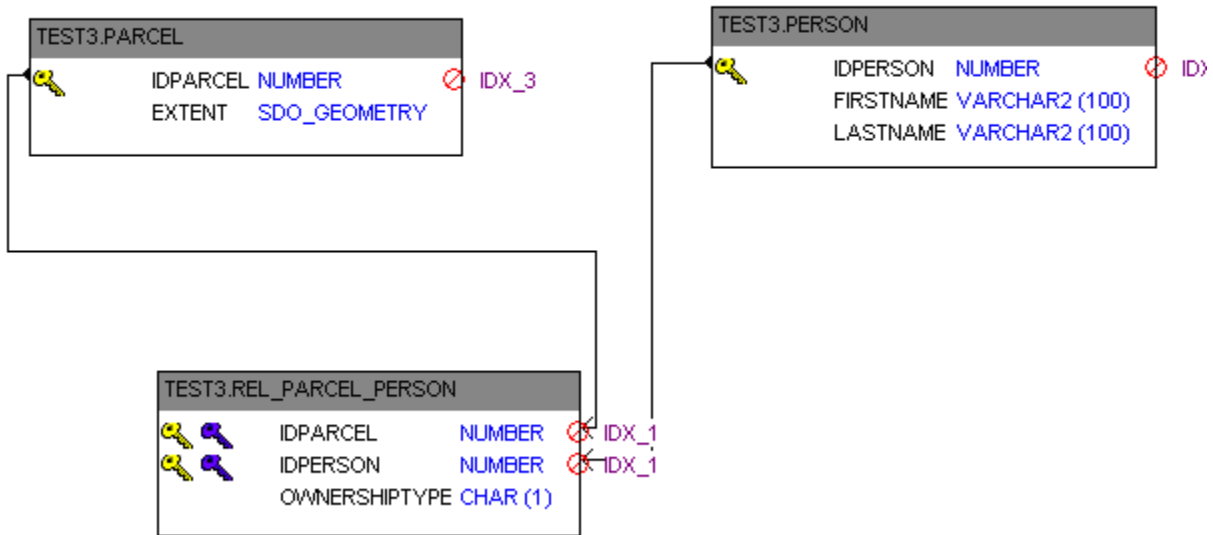
The complete mapping and schema files for this example are available in the distribution, under the data/erdas-apollo/db/oracle/ParcelPerson directory. They should be copied to config/erdas-apollo/providers/vector so that they can be referenced by their name in the providers.fac . The SQL instructions to build the tables and data are also provided.

Figure 131: Parcel-Person UML Diagram



It is assumed that the underlying database is Oracle and the relationship between the PARCEL and PERSON tables is based on a many..many relationship, the REL_PARCEL_PERSON holding the relations, as in the diagram below.

Figure 132: Parcel-Person Relational Diagram



As soon as an Explicit Mapping has been produced for the Parcel and Person feature types, a GML Application Schema appears with those two feature types defined, as in the "Simple Feature Type Definition" example above. Note that the GML3 rules are being used to define complex properties. Avoid relying on the old-fashioned GML2 syntax. Review the "Moving to GML3" section below for guidelines on how to get familiar with GML3 and how to migrate GML2 schemas to GML3.

Example: Parcel and Person Feature Types

```
<xsd:schema targetNamespace="http://www.erdas.com/wfs"
elementFormDefault="qualified" version="0.1">
<xsd:import namespace="http://www.opengis.net/gml"
schemaLocation="http://schemas.opengis.net/gml/3.1.1/base/feature.xsd"/>
<xsd:element name="Parcel" type="wfs:ParcelType"
substitutionGroup="gml:_Feature"/>
<xsd:complexType name="ParcelType">
<xsd:complexContent>
<xsd:extension base="gml:AbstractFeatureType">
<xsd:sequence>
<xsd:element name="IDParcel" type="integer"/>
<xsd:element name="extent"
type="gml:SurfacePropertyType"/>
</xsd:sequence>
</xsd:extension>
</xsd:complexContent>
</xsd:complexType>
<xsd:element name="Person" type="wfs:PersonType"
substitutionGroup="gml:_Feature"/>
<xsd:complexType name="PersonType">
<xsd:complexContent>
<xsd:extension base="gml:AbstractFeatureType">
<xsd:sequence>
```

```

        <xsd:element name="IDPerson" type="int"/>
        <xsd:element name="firstName" type="string"/>
        <xsd:element name="lastName" type="string"/>
    </xsd:sequence>
</xsd:extension>
</xsd:complexContent>
</xsd:complexType>

```

In this example, the Parcel feature type has an "extent" geometric property and an ID. Of course, as it inherits from gml:AbstractFeatureType, it has a set of additional properties like "gml:name" and "gml:description". Those properties will appear in the mapping file if they are mapped with a table column in the database.

The Person feature type has an ID, and a set of firstName and lastName properties.

To explicitly map those feature types onto SQL tables, a mapping file similar to the "Sample Explicit Mapping File" example above is needed. That same mapping extended to include the Person feature type will look like the example below.

Example:Parcel and Person Mapping

```

...
<Mapping>
  <SQL name="wfs:Parcel" schema="TEST3">
    <Table nameSQL="PARCEL"/>
    <Primary nameSQL="IDPARCEL" type="int"/>
    <Element name="wfs:extent" nameSQL="EXTENT"/>
  </SQL>
  <SQL name="wfs:Person" schema="TEST3">
    <Table nameSQL="PERSON"/>
    <Primary nameSQL="IDPERSON" type="int"/>
    <Element name="wfs:firstName" nameSQL="FIRSTNAME"/>
    <Element name="wfs:lastName" nameSQL="LASTNAME"/>
  </SQL>
</Mapping>
<Info name="wfs:Parcel">
  <SRS>EPSG:4326</SRS>
  <BoundingBox SRS="EPSG:4326" minx="-180."
miny="-90." maxx="180." maxy="90." />
  <Operations>*</Operations>
</Info>
<Info name="wfs:Person">
  <SRS>EPSG:4326</SRS>
  <BoundingBox SRS="EPSG:4326" minx="-180."
miny="-90." maxx="180." maxy="90." />
  <Operations>Query</Operations>
</Info>
<Export generation="exportOnly">
  <Add name="wfs:Parcel"/>
  <Add name="wfs:Person"/>
</Export>
</Mapping>

```

In that mapping example, notice that the mapped elements are the exported type names (Parcel and Person) and not the complex type names (ParcelType and PersonType). This is a constraint relating to GML3 model. For GML2 feature types, the complex type names have to be mapped.

Now, express each parcel as owned by one or more persons. First extend the GML schema to add a "owner" property (cardinality 1..n) to the ParcelType type and define the "PersonPropertyType" type that will hold the Person features. The schema becomes like in the example below.

Example:Related Parcel and Person Feature Types

```
<xsd:schema targetNamespace="http://www.erdas.com/wfs"
elementFormDefault="qualified" version="0.1">
<xsd:import namespace="http://www.opengis.net/gml"
schemaLocation="http://schemas.opengis.net/gml/3.1.1/base/feature.xsd"/>
  <xsd:element name="Parcel" type="wfs:ParcelType"
substitutionGroup="gml:_Feature"/>
  <xsd:complexType name="ParcelType">
    <xsd:complexContent>
      <xsd:extension base="gml:AbstractFeatureType">
        <xsd:sequence>
          <xsd:element name="IDParcel" type="xsd:integer"/>
          <xsd:element name="extent"
type="gml:SurfacePropertyType"/>
          <xsd:element name="owner" type="wfs:PersonPropertyType"
minOccurs="1" maxOccurs="unbounded"/>
        </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>
  <xsd:complexType name="PersonPropertyType">
    <xsd:sequence>
      <xsd:element ref="wfs:Person" minOccurs="0"/>
    </xsd:sequence>
  </xsd:complexType>

  <xsd:element name="Person" type="wfs:PersonType"
substitutionGroup="gml:_Feature"/>
  <xsd:complexType name="PersonType">
    <xsd:complexContent>
      <xsd:extension base="gml:AbstractFeatureType">
        <xsd:sequence>
          <xsd:element name="IDPerson" type="xsd:integer"/>
          <xsd:element name="firstName" type="xsd:string"/>
          <xsd:element name="lastName" type="xsd:string"/>
        </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>
```

The next step consists in expressing the relation in the mapping file and mapping it to the database tables and foreign keys. The relation itself will be modeled using a <Relation> element in the <Mapping> tag as defined in the "Feature Mapping Tags" appendix. For our example, the content of this tag will be:

```
<Relation sourceType="wfs:Parcel"
  targetType="wfs:Person"
  source="wfs:owner"
  association="REL_PARCEL_PERSON"
  sourceSQL="IDPARCEL"
  targetSQL="IDPERSON"
  ownership="false"
  reverse="false"
/>
```

"wfs:Parcel" and "wfs:Person" are respectively the parent and child ends of the relation. "wfs:owner" is the source property name. The association table is named "REL_PARCEL_PERSON". The sourceSQL parameter holds the name of the field in that table corresponding with the primary key in PARCEL. The targetSQL parameter holds the name of the field in that table corresponding with the primary key in PERSON. The "ownership" set to false means that when a Parcel feature is deleted, the linked Person entries will not be removed.

In a way similar to the one..many relations not using an association table, you need to add a mapping for the source property, like it was done for the "hasLanes" property in the previous example. Finally, in order to indicate that the wfs:Parcel it is involved in a relation, it is still necessary to changing its "generation" attribute from its default value to "relational".

The final mapping file now looks like below.

Example:Parcel and Person Relational Mapping

```
...
  <Mapping>
    <SQL name="wfs:Parcel" schema="TEST3"
generation="relational">
      <Table nameSQL="PARCEL"/>
      <Primary nameSQL="IDPARCEL" type="int"/>
      <Element name="wfs:extent" nameSQL="EXTENT"/>
      <Element name="wfs:owner" nameSQL="IDPARCEL" />
    </SQL>
    <SQL name="wfs:Person" schema="TEST3">
      <Table nameSQL="PERSON"/>
      <Primary nameSQL="IDPERSON" type="int"/>
      <Element name="wfs:firstName" nameSQL="FIRSTNAME"/>
      <Element name="wfs:lastName" nameSQL="LASTNAME"/>
    </SQL>
    <Relation sourceType="wfs:Parcel"
      targetType="wfs:Person"
```

```

        source="wfs:owner"
        association="REL_PARCEL_PERSON"
        sourceSQL="IDPARCEL"
        targetSQL="IDPERSON"
        ownership="false"
        reverse="false"
    />
</Mapping>

<Info name="wfs:Parcel">
  <SRS>EPSG:4326</SRS>
  <BoundingBox SRS="EPSG:4326" minx="-180."
miny="-90." maxx="180." maxy="90." />
  <Operations>*</Operations>
</Info>
<Info name="wfs:Person">
  <SRS>EPSG:4326</SRS>
  <BoundingBox SRS="EPSG:4326" minx="-180."
miny="-90." maxx="180." maxy="90." />
  <Operations>Query</Operations>
</Info>
<Export generation="exportOnly">
  <Add name="wfs:Parcel"/>
  <Add name="wfs:Person"/>
</Export>
</Mapping>

```

Below is a sample GML output from this WFS for the Parcel feature type. It is clear that one or more Person types are linked to the parent Parcel.

Example:Parcel and Person Sample GML3 Output

```

<wfs:Parcel gml:id="3">
  <wfs:IDParcel>3</wfs:IDParcel>
  <wfs:extent>
    <gml:Surface srsName="EPSG:4326">
      <gml:patches>
        <gml:PolygonPatch>
          <gml:exterior>
            <gml:LinearRing srsName="EPSG:4326">
              <gml:pos>-71.2994186401367 42.4900390625</gml:pos>
              <gml:pos>-71.2994338989258 42.4896957397461</gml:pos>
              <gml:pos>-71.2962524414063 42.4898750305176</gml:pos>
              <gml:pos>-71.2963287353516 42.4903060913086</gml:pos>
              <gml:pos>-71.2994186401367 42.4900390625</gml:pos>
            </gml:LinearRing>
          </gml:exterior>
        </gml:PolygonPatch>
      </gml:patches>
    </gml:Surface>
  </wfs:extent>
  <wfs:owner>
    <wfs:Person>
      <wfs:IDPerson>3</wfs:IDPerson>
      <wfs:firstName>Bernard</wfs:firstName>
      <wfs:lastName>Snyers</wfs:lastName>
    </wfs:Person>
  </wfs:owner>
</wfs:Parcel>

```

```

    </wfs:Person>
  </wfs:owner>
</wfs:Parcel>

```

Note: In such an association of feature types, you can run WFS GetFeature requests with a filter applying on a property of the child type. The request below searches for Parcels owned by Persons among which at least one is named "Dimitri".

```

<ogcwfes:GetFeature maxFeatures="20"
  xmlns:ogc="http://www.opengis.net/ogc"
  xmlns:ogcwfes="http://www.opengis.net/wfs"
  xmlns:wfs="http://www.erdas.com/wfs"
  version="1.1.0"
  service="WFS" >
  <ogcwfes:Query typeName="Parcel">
    <ogc:Filter>
      <ogc:PropertyIsEqualTo>

<ogc:PropertyName>owner/Person/firstName</ogc:PropertyName>
      <ogc:Literal>Dimitri</ogc:Literal>
    </ogc:PropertyIsEqualTo>
    </ogc:Filter>
  </ogcwfes:Query>
</ogcwfes:GetFeature>

```

The complete mapping and schema files for this example are available in the distribution, under the data/erdas-apollo/db/oracle/ParcelPerson directory. They should be copied to config/erdas-apollo/providers/vector so that they can be referenced by their name in the providers.fac . The SQL instructions to build the tables and data are also provided.

Limitations and Extensions

WFS Transactions on Complex Features:

When a feature has a complex property (i.e. a property of cardinality larger than 1, or a property which has sub-properties), an Update transaction cannot apply on a part of that property (i.e. an item of the collection or a sub-property). It is necessary to update the whole property: the property will be deleted and re-inserted with its new content.

Mapping of Enumerations

Regardless of the type of mapping used, it is useful to have properties which values are taken from a list instead of being a "wild" string. ERDAS supports two types of enumeration: XML Enumerations when the list of values is defined in the GML Application Schema and Relational Enumerations when the list of values is stored in a table in the database. This sections explains how to configure your enumeration type so that it will be transparently treated as such.

XML Enumerations

The three key elements to configure an XML Enumeration property are:

- Defining a type holding the list of values
- Declaring the type of the property, with a multiple cardinality

uTelling the mapping file that the property is an enumeration

Based on a basic example, here is the illustration on the configuration steps needed for setting up an XML Enumeration. The example used the BOSTON_ORA "BUSINESS" feature type on which a "Nature" property is added. The allowed values for this property are: SCHOOL, HOSPITAL, JAIL, HOME, Restaurant, RESTAURANT, Library and LIBRARY. So, we define a simple XML type deriving from "String", which values are those listed. The XML Schema code to add into the boston_ora.xsd file is:

```
<xsd:simpleType name="NatureType">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="SCHOOL"/>
    <xsd:enumeration value="HOSPITAL"/>
    <xsd:enumeration value="JAIL"/>
    <xsd:enumeration value="HOME"/>
    <xsd:enumeration value="Restaurant"/>
    <xsd:enumeration value="RESTAURANT"/>
    <xsd:enumeration value="Library"/>
  </xsd:restriction>
</xsd:simpleType>
```

Then, you have to add a property to your feature type, using the above type, and with a cardinality of more than 1. For the BUSINESS feature type, we give the name "Nature" to this property and we define it as below:

```
<xsd:element name="Nature" maxOccurs="unbounded" nillable="true"
type="wfs:NatureType">
```



That constraint on the cardinality can look artificial, but it is necessary in order to let the service consider the property as needing an additional resolution step when storing or retrieving it. If the cardinality is left to 1, Inserts will fail due to insufficient column width and retrieval will show the key instead of the value. In fact, after inserting enumerated values in the data source, you can notice that it will store a single character. This also means that you are limited to a list of 86 values for the enumeration, corresponding to the following character sets: A..Z, 0..9, a..z, !\$&()+,-./:;<=>?@[{}]*

Now, you have to map your property to a database field, and tell it that it is an Enumeration. Adding a property to be mapped is classically achieved by adding an <Element> tag. Declaring it as an enumeration is done by adding a <Enumeration/> to the Element. For the Nature property of the BUSINESS feature type, mapped to a "NATURE" column, the declaration looks like:

```
<Element name="wfs:Nature" nameSQL="NATURE">
  <Enumeration/>
</Element>
```

The "NATURE" field is defined in the database as a string with a size of 1 or more.

Various limitations:

Transactions (WFS-T operations): you can insert, update or delete features from your WFS but when inserting or updating, you have to use values that are part of the enumeration.

Filters: property values which are defined in an XML Enumeration can be used in search criteria, but only for exact match operators: Equals or Differs.

Relational Enumerations

This type of enumeration uses a database table to store the enumeration values. This solution is generally applied either when you already have sets of values stored in a separate table, or when the set of values goes beyond the limit for XML Enumerations.

The main steps needed to configure a Relational Enumeration property are:

- Declare a simple XML Type in your schema, for that property
- Add the mapping of this property in your mapping file
- Add a <RelationalEnumeration> element in the mapping file to link with the database

Once again, we take the BOSTON "BUSINESS" feature type example to explain and illustrate the configuration. If we consider the BUSINESS feature type is mapped with a relational table in a database, we can as well consider that a second table, named CITIES, exists, to hold the State and City code of the cities. Our job is to add a "CityCode" property to the BUSINESS feature type, ensuring that the values for that property are taken from the CITIES table. The SQL code to create the CITIES table can be:

```

CREATE TABLE CITIES
(
  CITY_ID    VARCHAR2(20 BYTE)  NOT NULL,
  CITYCODE   INTEGER
);

```

Declaring a `CityCodeType` simple type in the schema and adding a "CityCode" property with this type to the `BUSINESS` feature type is rather straight forward:

```

...
<xsd:simpleType name="CityCodeType">
  <xsd:restriction base="xsd:integer">
<!-- those enumeration elements are fake -->
    <xsd:enumeration value="108"/>
    <xsd:enumeration value="132"/>
  </xsd:restriction>
</xsd:simpleType>
...
<xsd:element name="CITY" minOccurs="0" nillable="true"
type="xsd:string">
  </xsd:element>
  <xsd:element name="CityCode" minOccurs="0" nillable="true"
type="wfs:CityCodeType">
  </xsd:element>
  <xsd:element name="TELEPHONE" minOccurs="0" nillable="true"
type="xsd:string">
  </xsd:element>
...

```

In the database, a new column has to be defined for the `BUSINESS` table. We name it "CITYLINK". Its must hold the same type as the key column of the `CITIES` table. And if your RDBMS supports constraints, you can link the `BUSINESS.CITYLINK` field with the `CITIES.CITY_ID` key column.

Then, that `CITYLINK` column is mapped with the "CityCode" property in the mapping file:

```

...
<Element name="wfs:CITY" nameSQL="CITY"/>
<Element name="wfs:CityCode" nameSQL="CITYLINK"/>
<Element name="wfs:TELEPHONE" nameSQL="TELEPHONE"/>
...

```

The last operations consists in telling the WFS that the `CityCode` property contains a Relational Enumeration stored in the `CITYCODE` column of the `CITIES` table linked by the `CITY_ID` key column. This is achieved by adding a `<RelationalEnumeration>` element in the mapping file, with four attributes holding the types or columns mentioned above. For the `CityCode` property, it can look like:

```

<RelationalEnumeration

```

```
name="wfs:CityCodeType"  
nameSQL="CITIES"  
pkey="CITY_ID"  
column="CITYCODE"  
</>
```

If the mapping file does not declare any <Export> tag, all the types defined in the XML schema will be disclosed in the WFS capabilities document. Adding an Export clause prevents the simple types from being exposed:

```
<Export generation="exportOnly">  
  <Add name="wfs:BUSINESS"/>  
</Export>
```

Remarks:

In the <RelationalEnumeration> tag, the "name" attribute addresses a simple XML type, not a feature type. So, you HAVE to define a simple type as one side of the link, and then use that type for one or more properties in your feature type.

The type must be the same between the simple type (here, an integer) and the column types in the corresponding table columns (here, the CITYCODE column is INTEGER).

The type of the column in the feature type table must NOT have the type of the values. It must have the type of the key field in the values table (here, String(20)).

Limitations and Extensions

Various limitations:

Transactions (WFS-T operations): you can not execute transactions on a feature type containing one or more properties defined by a Relational Enumeration if that property is not of type String. This is due to the fact that the value is used as key for reverse-resolving the row ID and that key is a string.

Filters: property values which are defined in an Relational Enumeration cannot be used in search criteria, except if they are defined as strings. And in all cases, the filters can only use exact match operators: Equals or Differs.

Extension:

The table containing the values can also contain other columns which can participate in the search. This feature allows to give alterate values to each row (a string in another language, for example) and have those values taken into account in the search phase. Note that the output value is taken from the first column.

In practice, you just need to mention one or more additional columns in the `RelationalEnumeration.column` attribute, so that it could look like: `column="CITYCODE,EUCODE,USCODE"` .

How to Control Mapping Correctness

If a syntax error is encountered by the WFS service in its mapping file, either the global service will refuse to start or the provider will not be accessible.

Requesting the capabilities of the provider and checking its content verifies that the content of the `<Info>` section is valid.

To check whether the `<SQL>` section of the mapping file is valid, run a `getFeature` request on the given feature type. A request failure or a wrong result (missing properties, invalid ID, wrong geometries) will denote incorrect mapping information. This error message may be: `java.sql.SQLException : ORA-00904 : invalid column name.`

If a single document defining the feature type schemas and the mapping is required, be aware that the definition of the feature type must be done before the mapping for a given feature type. Otherwise, an error message of type: "Unable to generate template based on `wfs:RoadType` or the type `wfs:RoadType` is unknown. (Think to check the prefix and the default namespace)." will be received

Moving to GML3

This section aims to clarify what parts of GML3 is supported in ERDAS APOLLO, and how to adopt it or migrate services to that new specification.

ERDAS APOLLO support of GML3

There are five major reasons why ERDAS made the huge investment to support the new OGC GML3 specification:

- More and more data providers expressed the wish to publish WFS services with complex data types and GML2 does not provide the appropriate framework to achieve it
- The OGC WFS 1.1.0 specification provides a good set of new features, but that specifications makes GML3 support mandatory
- GML3 defines new types of geometries which were not standardized before, e.g., Arc, Curve and Surface.

- GML3 defines new feature types in order to increase the standardization of vertical models: temporal, units of measure and coverages are just a few examples
- The new property-value model, at the heart of GML3, allows a much clearer semantic for readers of GML3 documents especially for complex properties.

GML3 Concepts and Schemas

Before reading the GML 3.1 specification, be familiar with the GML concepts as the document has more than 500 pages. Specification can be downloaded from the OGC web site.

There are some shortcuts to understanding the GML3 and being ready to use it:

- From the GML 3.1 specification (numbered 03-105r1), read chapters 21 and 23, which are guidelines to using GML3 schemas.
- ERDAS has configured a sample GML3-based WFS, named ATLGML3_SHAPE, relying on the usual configuration files (types and mapping). Browse those files and send WFS 1.1 requests to that WFS to see how the schema is defined and how the GML3 output is structured.
- When comfortable, migrate the services to the GML3. The next section explains how to achieve this.

Setting Up a ERDAS WFS Serving GML3

There are several ways to prepare a GML3-based WFS.

The first situation is whether a GML3 schema already exists, and a WFS is to be built on top of it. This is fast, as only the mapping file and the SQL scripts need to be built in order for the database to be generated. This can be done using one of the "Schema Generator" command-line tools provided as part of the ERDAS APOLLO distribution. It currently only applies to Oracle and Postgres provider types.

The second situation is if a database ready, and the mapping and the schemas to be are to be constructed. The "From SQL Generator" command-line tools will do the work. Just make sure to provide the "-gml3" option to the script so that GML3-compliant schemas will be build.

The third situation is if a set of WFS providers is up and running, and they are to be transformed so that they will properly respond to GML3-based requests. This migration can be done in a few steps explained in the next section.

Migrating a GML2 WFS to GML3

Assuming that there is an entry in the WFS providers.fac and this entry references a schema file (through the "types" parameter) and a mapping file (through the "mapping" parameter). If the provider is to become GML3-compliant, the providers.fac does not have to change, the servlet will analyze the types and mapping files and decide to enable GML3 or not.

The types file should be changed:

- The import clause for the "feature.xsd" schema has to be relocated. Its value is probably one of:

```
<xsd:import namespace="http://www.opengis.net/gml"
schemaLocation="http://www.opengis.net/namespaces/gml/core/feature.xsd"/>
```

or

```
<xsd:import namespace="http://www.opengis.net/gml"
schemaLocation="http://schemas.opengis.net/gml/2.1.2/feature.xsd"/>
```

It should become:

```
<xsd:import namespace="http://www.opengis.net/gml"
schemaLocation="http://schemas.opengis.net/gml/3.1.1/base/feature.xsd"/>
```

Note that the following import is also GML3-valid:

```
<xsd:import namespace="http://www.opengis.net/gml"
schemaLocation="http://www.opengis.net/namespaces/gml/base/feature.xsd"/>
```

- If the feature types hold complex properties, i.e., properties whose value is not a simple one like string, date, integer, those property definitions have to be adapted to match the GML3 property model. Such a property cannot have more than one value and this value is likely to be remote. So, it has to be split into a 2-level definition. The top level is the property name and the underlying level is the property value. It will often lead to defining a new complexType in the schema.

The mapping file has to be adapted to match the following rules:

- If the gml namespace is included, it has to be relocated to the same URL as for the types file.
- The <SQL> tag holding the feature type name has to map the global element name not its complex type as with GML2 mapping. So, if there is a feature type named "Parcel" whose XML type is "ParcelType" the GML2 mapping file mentions "ParcelType" in the SQL name attribute. Replace it with "Parcel".

- If mapping the gml:name and/or the gml:description properties of the feature type to a column in a table, add the compatible="gml2" attribute in the <SQL> element to by-pass the GML3 redefinition of those property types.

This action is needed because with GML3 the gml:name property has a cardinality of 0..n and has an optional "codeSpace" attribute. To be able to map it to a simple string, tell the servlet should be instructed to keep that the GML2 behavior. The GML3 output of the service will remain valid. That action also has to be taken if the gml:description property is to be mapped to a simple string, as its GML3 definition allows it to be either a simple string or a reference to a remote value.

- If the mapping does not take advantage of any of the properties inherited from AbstractFeatureType, set the "compatible" attribute to "pure".

Some more advanced rules also apply, but as they relate to particular case, they will be treated on a case-by-case basis.

Setting Up a ERDAS WFS Serving GML-SF (Simple Feature)

For a WFS to server features matching the GML Simple Feature Profile (also known as Level 0), you need to import the feature schema ending with "gmlsf.xsd". So, the import clause could look like:

```
<xsd:import namespace="http://www.opengis.net/gml"
schemaLocation="http://schemas.opengis.net/gml/3.1.1/profiles/gmlsfProfile/1.0.0/gmlsf.xsd"/>
```

You can build that schema manually or produce it with the "From-SQL Generator" tool, taking care of setting both parameters: -gml3 and -cl <n> where <n> is the level of GML-SF compatibility. <n> being any of 0, 1 and 2 currently produce the same result.

Feature Mapping Tags

This appendix presents the detail of the configuration XML tags. They are divided in several sections, for each type of information:

- The <MAPPING> section
- The Metadata <INFO> section
- The Capabilities Feature Type section <EXPORT>
- The <COLLECTION> section
- The <OPTION> section
- The <UserFunction> section
- The <UnitDefinition> section
- The <UnitAssociation> section
- The <WMS> section

Mapping Section <MAPPING>

This section lists and briefly describes each of the XML tags that can appear in the mapping section document.

Table 43: The Mapping Tag

Element (* means multiple occurrences)	Allowed Parent	Attributes	Description
Mapping	document root		the root element of the mapping (1)

Table 44: The SQL Tag

SQL	Mapping		defines the mapping associated to a feature type. (2)
		name	the name of the feature type to map or the template name to use for 'sql' generation. If using "%", all tables found in the associated source will be mapped, including System tables if the user is allowed to view them.

Table 44: The SQL Tag (Continued)

		generation	<p>specifies the kind of mapping.</p> <ul style="list-style-type: none"> • specific: the specified elements are mapped (default) • auto: the element mapping is automatically generated • sql: the type is created from the database • relational: to map related tables onto a single feature type • autogen: same as "auto" mapping but can be changed manually • simple: same as specific without capability of mapping complex types onto a complex SQL type.
		schema	used when generation=sql. It specified a template for the schema/user to address. If not specified, the null value is used leading to processing all schemas available to the user.
		compatible	used to force the mapping be compatible with GML2. Allowed values are pure or gml2. 'pure': the parent gml:AbstractFeatureType is considered empty. 'gml2': the gml3 type is made compatible with GML2. (gml:name and gml:description properties are considered simple strings with cardinality maximum of 1)

Table 45: Sub-Elements of the SQL Tag

Table	SQL		defines the mapping of a type to a table
		nameSQL	the table name
		alias	a possible table alias to use in query generation
		user	a schema name if the table does not belong to the current schema (Oracle only)
TableScale	Table		defines the set of alternative tables to invoke based on the scale from the request. There can be 0 to n instances of <TableScale> elements.
		scaleMax	defines the maximum scale at which the current Table is invoked (inclusive). Default is infinity.
		scaleMin	defines the minimum scale at which the current Table is invoked (inclusive). Default is 0.
		nameSQL	the table name
		user	A schema name if the table does not belong to the current schema
		alias	a possible table alias to use in query generation
Primary	SQL		(3)

Table 45: Sub-Elements of the SQL Tag (Continued)

		name or nameSQL	the column name of the primary key. Several columns can be given (comma separated)
		fid	<p>can have one of the following values:</p> <ul style="list-style-type: none"> • true: the primary key matches the XML ID requirements, so no encoding is needed • auto: the primary key is generated by the database (for insert operation) <p>The fid="auto" option is not supported for PostgreSQL/PostGIS as this DBMS does not support the unique identifier auto-generation option. Oracle does, based on UUIDs.</p> <ul style="list-style-type: none"> • false: the primary key doesn't match the XML ID requirements • generated: the primary key is generated by the wfs (as a long or a string)
		type	a comma separated list of the type of each column (schema type). If not specified, default to xsd:string, unless 'auto' is specified for fid
		allowInsertId	if set to true (default is false), the feature IDs are kept (if present) during Insert operations. This is mainly intended to transfer features from one WFS to another.
NoPrimary	SQL		Sets no mapping for the primary key. It will lead to absence of "fid" in the outputs.
Foreign	SQL		defines the mapping of the foreign key in auto mapping
		nameSQL	the column name of the foreign key (used in auto mapping)
Element *	SQL		defines the mapping of one element
		name	the name of the element to map (the feature element name)
		nameSQL	the column name (not required in auto mapping)
		typeSQL	the type of the column (only required for Object and Array types)
		force	Optional field. The only allowed value is "mandatory". It forces the element to have a minimum cardinality of 1 and to become nullable. This is mainly used when there are several optional geometries in a type to force the mapped ones to become mandatory. This helps retrieve the default geometry property.

Table 45: Sub-Elements of the SQL Tag (Continued)

		measure	Optional measure unit name. See <UnitDefinition> and <UnitAssociation> sections below.
		nameSQLCol2	the name of the second column (not required in automatic mapping) used to map the end time of a TimePeriod.
Attribute *	SQL, Element		defines the mapping of one attribute
		name	the name of the attribute to map (the feature element name)
		nameSQL	the column name
Geometry *	SQL		defines the mapping of a geometry element (used in databases that do not have geometry data types)
		name	the name of the geometry element
		nameSQL	the comma separated list of column names (this defines the mapping of a point)
		dontCheckName	if set to true (the default is false) the column names are not checked for characters requiring the use of quote
		filterName	the column name to use in filter expression, default is to use the nameSQL value
		functionName	the name of a function which is used on the select part and enclosed the nameSQL (mainly applies to MySQL, the function name being "asText")
Lock	SQL		defines the mapping of the lock field
		nameSQL	the column name
MapGen	SQL		defines the specific mapping for generation of GetMap (see the Advanced Configuration appendix). There can be 0 to n instances of <MapGen> elements.
		scaleMin	defines the minimum scale at which the current MapGen applies (inclusive). Default is 0.
		scaleMax	defines the maximum scale at which the current MapGen applies (inclusive). Default is infinity.
MapGenFormat	SQL	PCDATA	the content of this element is a list of formats for which the MapGen capability is disabled. Several separators can be used - among them are comma, blank, carriage return and line feed. Formats are case-insensitive and are expressed as HTML, GML, SVG, XML, ...
Field *	MapGen		defines the properties to output.

Table 45: Sub-Elements of the SQL Tag (Continued)

		name	the name of a database column name to use.
		PCDATA	a string that will replace the column name in the underlying request. Optional.
Where	MapGen		a where clause which will be added (with an And) to the underlying request. Optional, and only applicable to providers using a SQL query language (Shapefiles do not!)
Last	MapGen		an ending clause that will be added to the underlying request (typically, a group by or a sort order). Optional, and only applicable to providers using a SQL query language (Shapefiles do not!)
NoldGeneration	MapGen		prevents computation of the feature identifier, when it is irrelevant.
NoQueryGeneration	MapGen		if specified, no query will be sent to the data source for the specified scale range
Relation	Mapping		defines a relationship between two types (a source type has a relation with a target type). The relation is not used in the "auto" mapping. The relation must be defined after the mapping of both types. It can be noticed that in case of a many-to-many relationship, targetSQL and sourceSQL contain the primary key of the sourceType and targetType.
		sourceType	the name of the source type of the relation
		targetType	the name of the target type of the relation
		source	the name of the source element in the source type
		target	the name of the target element in the target type -if not specified, in a one-to-x relationship, the targetSQL field is used and if not defined, it defaults to the primary key
		association	the name of the association table (used only for many-to-many relationships). In this case, the types of the field must be the types of the primary keys of the source and target types.
		targetSQL	the name of the target field either in the target type or in the association table.
		sourceSQL	the name of the source field in the association table (used only for many-to-many relationships)

Table 45: Sub-Elements of the SQL Tag (Continued)

		ownership	indicator of ownership. Implies cascaded deletions of the related objects. (In case of a many-to-many relationship, the delete occurs only in the association table) The possible values are: false = no automatic delete occurs true = the source owns the related objects, so they are deleted.
		reverse	a boolean used in a many-to-many relationship to automatically add the reverse relationship (the default is false)
		targetTable	the name of another table holding the relation data
RelationalEnumeration	SQL		build a mapping between a simple type and a table of values. The table of values has a primary key and several columns with at least one containing values. The logic is to store the primary key of a value in the feature table instead of the value. Note that the SQL type of the primary key and the SQL type of any elements of the mapped type must match. As the type is mapped, this is valid for all elements of this type. If you have a element of the enum type which has a max cardinality > 1, it will not work
		name	the name of the simple type to map. It is usually an enumeration type.
		nameSQL	the name of the SQL table containing the value
		pkey	the name of the primary key
		column	a comma-separated list of column names containing some values

Table 46: Sub-elements of the Element tag

Type *	Element		defines the types used in a collection
		name	the type name
Blob	Element		mark the element to be stored as a blob (the element should be a collection)
HandleFID	Element		mark the collection elements to handle its own FID. It implies that the WFS does NOT recognize this FID as a valid FID of get feature request

Table 46: Sub-elements of the Element tag

Enumeration	Element		mark the collection elements to use an enumeration mapping where the collection is mapped onto a string. Each character of the string is one element of the collection. This tag has no effect if the element is not a collection. This reduces the query done to insert/retrieve the feature and allows to use the property in the filter like a simple property.
--------------------	----------------	--	--

Metadata Section

<INFO>

Expresses the metadata and helps the system to be compliant.

"PCDATA" means "Any String" e.g., a title can be any free text.

Table 47: The Info Tag

Element (* means multiple occurrences)	Allowed Parent	Attributes	Description
Info	Document root		the root element of the information section (2)
		name	the type name linked to the information (7)
		base	the name of the parent type whose info should be inherited by this type info

Table 48: Sub-Elements of the Info Tag

Title	Info	PCDATA	defines the title associated to the type and exported in the capabilities
Abstract	Info	PCDATA	defines the abstract associated to the type and exported in the capabilities
SRS	Info	PCDATA	defines the list of supported SRS, separated by a space. The first SRS is the internally used one (4).
		overwrite	boolean value (true, false) defining if the list of SRSs exposed in the capabilities document is exclusively composed of the current list or if the SRSs coming from the data are added. Default: false.
BoundingBox *	Info		defines the bounding box
		SRS	the box SRS

Table 48: Sub-Elements of the Info Tag (Continued)

		minx	the X value of the lower left point
		miny	the Y value of the lower left point
		maxx	the X value of the upper right point
		maxy	the Y value of the upper right point
Operations	Info	PCDATA	<p>defines the supported operations as a comma separated list. Allowed values are *, # , Insert, Delete, Update, Lock, Query and Native. Warning: some operations may not be supported by all connectors. "*" means all operations. # means Query, Insert, Delete, Update.</p> <p>If not defined, the default is "Query". (5)</p>
Dimension *	Info		<p>as ERDAS WFS Servlet is also able to support the WMS Specification, this tag describes the behavior of the getMap and publishes the dimension and extent for conformance to WMS Specification. (6)</p>
		name	the dimension name that will be published in the WMS capabilities document. It must be prefixed with DIM_ in the requests, except for TIME and ELEVATION
		property	the feature property name (default is name)
		unit	the unit name (mandatory attribute, value cannot be empty)
		symbol	the symbol name
		default	the default value (optional)
		PCDATA	the extent: see (6)
Metadata	Info	PCDATA	<p>the template url of the metadata url (see MetadataURL section in Advanced Configuration appendix). If this string is empty, the DefaultMetadata option is used. If this one is also empty, the provider global default is used.</p>
		type	<p>the type of the metadata (default is TC211) Acceptable values are TC211, FGDC, 19115, 19139.</p>

Table 48: Sub-Elements of the Info Tag (Continued)

Legend	Info	PCDATA	the template url of the legend url (see LegendURL section in Advanced Configuration appendix). If this string is empty, the DefaultLegend option is used. If this one is also empty, the provider global default is used.
LegendConfig	Info,Option		This options allows to configure the output produced by the GetLegendGraphic through SLD rules. This element can be either in the Info section for a single feature type, or in the Option section for global application. <LegendConfig outline="true" width="30" height="40" fontsize="40"/>
		width, height	the width and height of the drawing area (default width is 18, default height is 12)
		widthOffset	the offset between the image border and the drawing area (default 5)
		offset	the y space between two drawing areas
		textRightOffset	the offset between the text and the image border (default 5)
		textBottomOffset	the offset between the bottom of the drawing area and the text (default 3)
		fontSize	the text font size in points (default 10)
		fontName	the text font name (default Arial)
		fontStyle	the text font style (use bold for a bold font, defaults to normal)
		outline	boolean. If set, an outline is drawn around the drawing area (default false)
		outlineColor	the color of the outline (default black)
		textColor	the text color (default black)
		backColor	the image background color (default is white). Used only if not present in the request
		scale	the scale factor to apply when drawing (default 4). It means that the rule is drawn scaled up and down , e.g the marker size will be reduced by the scale factor.

Table 48: Sub-Elements of the Info Tag (Continued)

Queryable	Info	PCDATA	defines if the WMS layer built on top of the feature type can be queried. The default value is "true". So, this tag clears the queryable flag for the given layer.
ScaleHint	Info		defines the scale hint to give to WMS layers. Not that the <ScaleHint> element will be automatically computed if a <MapGen> element exists for this layer. However this one always takes precedence <ScaleHint scaleMin="10000" scaleMax="50000"/>
		scaleMin	defines the minimum scale (default 0)
		scaleMax	defines the maximum scale (default is max)

* means multiple occurrences

- An occurrence of the "Mapping" element can appear for each feature type. A single element can group several feature type mappings.
- For a given feature type, if its schema is in the same file as the mapping, the <SQL> and <Info> tags for this feature type must appear **after** the schema.
- In common situations, the "Primary" element is mandatory. If not mentioned, the "fid", which value is based on this element, will not be permanent. Moreover, this element is the only one allowed for a mapping of type "sql".
- In the case of an Oracle Spatial database, the value of the first SRS will be matched with the SRID given in the USER_SDO_GEOM_METADATA table, whatever value it has.
- Operation "Native": This declaration indicates that in the capabilities document the <Native> element (see WFS 1.0.0 spec) is supported by the WFS.

In the case of ERDAS, the attributes vendorId and safeToIgnore for this element are ignored. The content of the element is directly passed through as a Transaction to the JDBC source. No response is returned. That means that for a SQL request, a SELECT order will provide no data back. However, a CREATE, an INSERT or a DROP will do something.

This operation does not enforce the interoperability concept and is a threat as it allows the database to be altered.

For the ERDAS WFS, the JDBC-type data sources are the only ones that support that "native" operation, i.e. Oracle, Oracle OCI, Postgres, DBF, ODBC and MS-Access. The Shapefiles and ArcSDE connectors do not support it.

- Dimension:

First, it allows the publishing in the WMS capabilities document one or more <Dimension> and <Extent> tags for one or more of the layers. Then based on this information the client application can insert into its WMS requests (GetMap and GetFeatureInfo) other parameters that provide additional filters to the data. It somehow resembles the WFS <Filter> for a WMS.

Extract from the WMS 1.1.1 capabilities DTD for the <Dimension> and <Extent> tags:

```
<!ELEMENT Dimension EMPTY >
<!ATTLIST Dimension
  name CDATA #REQUIRED
  units CDATA #REQUIRED
  unitSymbol CDATA #IMPLIED>
<!-- The Extent element indicates what _values_ along a dimension
are valid. -->
<!ELEMENT Extent (#PCDATA) >
<!ATTLIST Extent
  name CDATA #REQUIRED
  default CDATA #IMPLIED
  nearestValue (0 | 1) "0">
```

Configuration: The <Dimension> and <Extent> tags will appear in the WMS capabilities, as soon as a <Dimension> tag is declared in the <Info> block of the WFS mapping file. The attributes to give to this <Dimension> tag are:

- name: The name that will appear in the capabilities document for the elements <Dimension> and <Extent>. It is also the name to use in the WMS requests, prefixed with DIM_ (except for TIME and ELEVATION).
- property: the property name of the feature type.
- unit: A character string indicating the type of unit of the property. This value is not checked by the ERDAS WFS. It is only used to provide the necessary information in the capabilities document as the "units" attribute. This attribute is mandatory. Example: kilometer, m, second, time, etc.

- **symbol:** will produce the "unitSymbol" attribute in the <Dimension> element of the capabilities. This attribute is optional.
- **default:** A default value to use if the request does not mention this dimension as parameter. If no value is defined, the request will do no filtering on this property. This value will also be used for the "default" attribute of the <Extent> element in the capabilities. This attribute is optional.

The element to insert as PCDATA in the <Dimension> tag is a character string representing the list or range of allowed values. It uses the syntax described in Appendix C of the WMS 1.1.1 spec: value or value1,value2,value3 or min/max/resolution or min1/max1/res1,min2/max2/res2. For example, a list of values "A,C,F,G,W" or a range of values: "10/100/2". The value "?" has no particular meaning and should not be used. This value will appear in the <Extent> element in the capabilities. It is not checked by the WFS.

Examples:

```
<Dimension name="FC" property="GNS_FC"
unit="">A,H,L,P,R,S,T,U,V, </Dimension>
```

```
<Dimension name="SCALELEVEL" property="LEVEL" unit=""
default="0">0,1,2</Dimension>
```

```
<Dimension name="TIME" property="TIME" unit="ISO8601"
default="1999-01-01/1999-01-07/P1D">1999-01-01/2003-12-
31/P1D</Dimension>
```

- **SRS and BoundingBox priority rules:**

The <SRS> tag defines the reference system(s) supported by the WFS. The first in the list refers to the internal SRS. The <BoundingBox> tag defines the server box. It is used to calculate the LatLonBoundingBox found in the capabilities document. The information given in the <Info> tag for an explicitly named feature type has priority and replaces the information extracted from the database. The SRS and BoundingBox tags are treated independently from each other. This means that if a BoundingBox but no SRS is given in the database, the bounding box will be associated with the SRS given in the <Info> tag.

An anonymous <Info> tag, i.e., with no feature type name, will complement the information obtained from the database, but does not replace it.

If the SRS and the BoundingBox are not mentioned in the database or in the <Info> tag, the values taken are WGS84 (EPSG:4326) and (-180,-90,180,90).

Capabilities Feature Type Section: <EXPORT>

This sets the FeatureTypes presented to the world.

Table 49: The Export Tag

Element (* means multiple occurrences)	Allowed Parent	Attributes	Description
Export	Document root		the export section defines the types which are really declared in the capabilities document
		generation	the value can be: <ul style="list-style-type: none"> • mapped: mapped types are exported if they are not explicitly removed, added types are also exported • exportOnly: only explicitly exported types are exported

Table 50: Sub-Elements of the Export Tag

Add *	Export		
		Name	the type name to add
Remove *	Export		
		name	the type name to remove

Collection Section: <COLLECTION>

This tag permits the remaining of the root element produced by a getFeature request. The default is "featureCollection", but may be renamed, if needed using this tag.

Table 51: The Collection Tag

Element (* means multiple occurrences)	Allowed Parent	Attributes	Description

Table 51: The Collection Tag

Collection	Document root	Name	<p>the type name to use as collection or as member element.</p> <ul style="list-style-type: none"> If the type name is not of type Association, the type name will be used as collection. The typename can be a global element or a type. (The first element whose type is Association will be used as feature Member) If the type name is of type Association, the collection element is unchanged, and the member element is replaced with this one.
-------------------	---------------	------	--

Options Section:
<OPTION>

The tags belonging to this section have several goals, but apply to all feature types.

Table 52: The Option Tag

Element (* means multiple occurrences)	Allowed Parent	Values	Description
Option	Document root		A set of options applying to all feature types
AgressiveSQL-Mapping	Option	gml2, gml3	This option specifies the use of an aggressive method to compute the geometry type from SQL. "gml2" is for GML2 types, "gml3" for GML3 types. The option may also be specified in the providers.fac file as a provider property. This is not always an accurate method. (Default: disabled) <AgressiveSQLMapping>gml2</AgressiveSQLMapping>
AllowDeprecatedGML3	Option	true, false	Allows the output of deprecated GML3 geometries, lighter than the new ones, when the actual geometry complies. Default: false. <AllowDeprecatedGML3>true</AllowDeprecatedGML3>
AllowFullDistanceEmulation	Option	true, false	This option, if set, emulates the distance sort for the output of the WMS GetFeatureInfo request. If not set, the system assumes the closest feature is returned first by the provider. (Default: false)
AllowLaxGMLModel	Option	true, false	If set, the model verifier will be a little more permissive. Among others, it will allow attributes for GML3 properties. (Default: false)
AllowLaxParsing	Option	true, false	This option allows the GML reader included in the WFS to tolerate some minor errors (like entering invalid enumeration values...). (Default: false). <AllowLaxParsing>true</AllowLaxParsing>

Table 52: The Option Tag (Continued)

AlwaysGenerateLegend	Option	true, false	If set, the default legend template is used even if no Legend tag is defined: the LegendURL tag is forced in the capabilities document (Default: false). <AlwaysGenerateLegend>>true</AllowGenerateLegend>
AlwaysGenerateMetadata	Option	true, false	If set, the default metadata template is used even if no Metadata tag is defined: the MetadataURL tag is forced in the capabilities document (Default: false). <AlwaysGenerateMetadata>>true</AllowGenerateMetadata>
CheckSomethingLocked	Option	true, false	It applies to WFS transactions. If set, the WFS will check that something is locked (or unlocked) before processing a delete or update request. The default value is false. Warning: if set, any request to delete unexisting features while no other feature is locked will return an error message.
ComputeBoundedBy	Option	true, false	This allows the WFS to compute the <gml:boundedBy> content for each feature if not provided. (Default: false). <ComputeBoundedBy>>true</ComputeBoundedBy>
ConformToSimpleFeatureProfile	Option	0,1,2	This option forces the schema produced with an SQL Mapping to conform to the OGC-GML Simple Feature Profile. (Default: not set). The value of the option is the conformance level (0,1,2). <ConformToSimpleFeatureProfile>0</ConformToSimpleFeatureProfile>
DefaultDPI	Option	positive number	The default dpi to use in the renderer for GetMap operations
DefaultLegend	Option	string	Defines the default path template for legend files location
DefaultMetadata	Option	string	Defines the default path template for metadata files location
DoNotRewriteSchemaLocation	Option	true, false	This option prevents the WFS from rewriting the schema location with a location relative to itself. Default is false.
DontEmbedSVG	Option	true, false	If true, requests not to embed SVG images/symbols but produce external url instead
DoStreamGML	Option	true, false	This options allows the WFS to output the produced GML as a continuous stream without doing internal copy. So It requires less memory and there is no need for the MaxMemorySize option to produce large output. Restriction: It implies to envelope computation. (Default: false). <DoStreamGML>>true</DoStreamGML> (This option is provided in Beta state)
EncodingCharset	Option	encoding	This specifies the charset used to encode the GML response, the GetCapabilities response and the DescribeFeatureType response. (Default: utf-8). <EncodingCharset>iso-8859-1</EncodingCharset>

Table 52: The Option Tag (Continued)

ShapeEncodingCharset	Option	encoding	This specifies the charset used to encode the SHAPE response to a GetFeature. (Default: utf-8). <ShapeEncodingCharset>iso-8859-1</ShapeEncodingCharset>
FilterAlphaMin	Option	positive integer	This option is used during generation of 8 bits images (GIF or PNG8). It defines the minimum alpha value to be non-transparent. (Default: 128). <FilterAlphaMin>80</FilterAlphaMin>
FilterDistanceMin	Option	positive integer	This option is used during generation of 8 bits images (GIF or PNG8). It defines the minimum distance to be distinct from the reference color. (Default: 1000). <FilterDistanceMin>500</FilterDistanceMin>. Note: the value is the square of the distance and will be compared with $dR*dR + dG*dG + dB*dB$.
FollowEPSGDef	Option	true, false	This option forces the WFS to follow the order of axis defined by EPSG when an EPSG SRS is used. It mainly applies to geographic projections, for which EPSG sets the latitude-longitude order and some OGC specifications set the longitude-latitude order. (Default: false) <FollowEPSGDef>>true</FollowEPSGDef>.
GenerateUpper-Case	Option	true, false	If true, sets the use of uppercase in AUTO mapping mode
Generate8BitsPNG	Option	true, false	If true, requests to generate PNG with 8-bits depth instead of 24-bits. This makes the PNG output compatible with Microsoft Internet Explorer versions <= 6.0, which cannot deal with RGBA PNG. This option always overrides the QUALITY parameter
GMLOutputFollowModel	Option	true, false	Request to have the default GML output format to be GML2 or GML3 depending of the actual model used. If not set, the default is GML2 for WFS 1.0 and GML3 for WFS 1.1 or above.
IgnoreDataSRS	Option	true, false	If set, this option allows you to use and expose the SRS defined in the mapping file and ignore the SRS defined in the data (such as a prj file forshapefiles). It currently only applies to shapefiles. Default value is false. See also the Info "SRS" element, "overwrite" attribute.
LegendConfig	Info, Option	true, false	This option allows to configure the output produced by the GetLegendGraphic through SLD rules. This element can be either in the Info section for a single feature type, or in the Option section for global application. See that same element in the "Info" section for detail of each attribute. <LegendConfig outline="true" width="30" height="40" fontsize="40"/>
MaxMemorySize	Option	number	The max memory size to allocate per request in bytes. This is a hint. It is currently used in the GML generation. A negative or zero value puts everything in memory. (Otherwise a backup file is used , well it is slower but safer...) <MaxMemorySize>10000000</MaxMemorySize>

Table 52: The Option Tag (Continued)

OGCStrictConformance	Option	true, false	If set, this option forces the output of the WFS to strictly obey the OGC-published schemas. It mostly impacts the set of FilterCapabilities in the WFS 1.1 capabilities document. Keeping this value set to false allows you to extend the set of FilterCapabilities to other operators.
OptimizeGMLOutput	Option	true, false	If true, requests not to compute the box of the resulting feature collection. This prevents from rewriting the GML output stream to put the computed box at the beginning
RemoveDeleteCheck	Option	true, false	It applies to WFS "Delete" transactions. If set, the WFS will never check if something has been deleted. Otherwise (the default behavior), the WFS will check if there has been a deletion. If there was a deletion and the transaction still exists it will throw an error.
RemoveUpdateCheck	Option	true, false	It applies to WFS "Update" transactions. If set, the WFS will never check if something has been updated. Otherwise, the WFS will check if there has been an update (this is the default behavior). If there was an update and the transaction still exists, it will throw an error.
SchemaLocation	Option	string	This option specifies the schema location attribute written into GML output. It is written as-is, no correctness check is done. The default behaviour of the WFS is to generate one relative to itself, but sometimes it may be convenient to force the location of the various schemas. <SchemaLocation> http://www.opengis.net/ogc http://schemas.opengis.net/gml/2.1.2/feature.xsd</SchemaLocation>
SRSOutputFormat	Option	none, SHORT, URN, URNCONVERT	This option defines how the WFS outputs the SRS. The default is as the user typed it, "SHORT" forces the code:value notation (e.g. EPSG:26986), "URN" forces the URN notation, "URNCONVERT" sets the URN notation converted into the OGC codes (e.g. EPSG:4326 gives OGC:84). <SRSOutputFormat>URN</SRSOutputFormat>
UsePixelPatternV2	Option	true, false	Requests to generate a pixel space pattern using a method more compatible with Adobe Viewer. Default is false. <usePixelPatternV2>>true</UsePixelPatternV2>
UseUTCDate	Option	true, false	requests to output UTC dates instead of GMT ones (UTC = GMT + 12h). (Default: false) <UseUTCDate>>true</UseUTCDate>

User Functions Section: <UserFunction>

The tags belonging to this section allow publishing of functions which are implemented in the underlying database.

Table 53: The UserFunction Tag

Element (* means multiple occurrences)	Allowed Parent	Attributes	Description
UserFunction	Document root		defines a user function implemented in the backend database
		name	the user function name as exported in the capabilities and used in the filter
		nameSQL	the database function name (or procedure name)

Table 54: Sub-Elements of the UserFunction Tag

Parameter *	UserFunction		Defines the nth parameter type
		type	the parameter type (default to string)
		nbr	the number of parameter to repeat (default is 1)
Return	UserFunction		optional element to define the function returned type. If not specified, the returned type is not a string.
		type	defines the return type of the function

Units Definition Section: <UnitDefinition>

The tag belonging to this section allows reference to unit dictionaries for use with Measurement types and unit associations, as soon as the predefined units do not match the need. In fact, ERDAS WFS comes with a set of predefined units, namely Angles in Degrees (deg), Angles in Grads (grad), Angles in Radians (rad), Distance in Kilometers (Km), in Meters (m), in Centimeter (cm), in Millimeter (mm) and in Inches (in).

The <UnitDefinition> element has no sub-element. It sets the URL of the dictionary used for measures. The referenced document must contain a GML dictionary of unit definitions.



In the mapping file, the association has to be declared AFTER the unit and the property are known. So, the <UnitDefinition> element has to be set before the <UnitAssociation> tag if the association uses this form, or before the <Mapping> tag if the "measure" attribute is declared in an <Element> tag.

Table 55: The UnitDefinition Tag

Element (* means multiple occurrences)	Allowed Parent	Attributes	Description
UnitDefinition	Document root		References a dictionary used for measures
		PCDATA	The URL to the units dictionary

**Units Association Section:
<UnitAssociation>**

As soon as a feature property is a measure, its value is given in a unit. There are two ways to associate a feature property with a unit. Either you can use the "measure" attribute of the "Element" tag or you can define a Unit Association using the current tag.



In the mapping file the association has to be declared AFTER the unit and the property are known. We recommend the <UnitAssociation> element to be set at the end of the mapping file.

Table 56: The UnitAssociation Tag

Element (* means multiple occurrences)	Allowed Parent	Attributes	Description
Unit Association	Document root		associates a measure unit to a property of a feature type
		type	the feature type
		name	the property name (defined by an Element tag)
		measure	the unit name (one of the internal ones, or one defined in the unit dictionary given by the UnitDefinition tag)

**WMS Layer Hierarchy Section:
<WMS>**

The tags belonging to this section allow exposing the layers as a hierarchy, so that the WMS capabilities document discloses an organized structure of layers, and so that aggregates of layers can be requested.

Table 57: The WMS Tag

Element (* means multiple occurrences)	Allowed Parent	Attributes	Description
WMS	Document root		defines some behaviour of the WMS interface built on top of the WFS

Table 58: Sub-Elements of the WMS Tag

Element (* means multiple occurrences)	Allowed Parent	Attributes	Description
Layer	WMS, Layer		Defines a node in the layer structure of the WMS
		title	the title that will appear in the WMS capabilities document
		type	for a layer corresponding to an actual feature type, the name of that feature type
		name	only applies to layers which are not feature types. It will allow the category layer to be used in requests.

Coverage and Image Servers

Image Server Concepts

The ERDAS Image Server is a Java component that manages and delivers an Image Service on geo-enabled raster files. The Image Server is able to manage one single image or multiple images organized as a layer in a directory. The Image Server is wrapped by the ERDAS WMS Framework and allows the client to discover and query geo-rasters through a coherent and standard-based WMS interface. The ERDAS WMS Framework is a Java component that offers OGC Web Map Server interface.

The configuration of the image server is done by configuring an ImageProvider. The raster database may be one single image (SimpleProvider), multiple images (tiles) organized as a layer, geospatially indexed, and stored in a directory (LayerProvider) or a set of images with one layer per image (MultiSimpleProvider). The image files can be of multiple types but the engine is mostly designed to produce good results on uncompressed TIFF images (including GeoTIFF) or uncompressed BIL images. Formats like Jpeg, Gif or others are supported but do not provide adequate performance.

In the case of many image layers, the distribution provides a tool to index the image database. See "Tools and Viewers" chapter for an explanation of the ImageLayerIndexer tool.

Image Provider Types

There are several types of Image providers from which to choose:

- The SimpleProvider. It is used to publish a single image.
- The LayerProvider. It is used to create a layer using a set of tiles. The images do not need to have the same size, the same format and may even overlap.
- The MultiSimpleProvider. It is used for a collection of images that do not form a layer. This is useful if the same image is used at a different time of the year, the same place at different scales or even unrelated images.

Configuring Individual Coverages/Images

The configuration file where a given image is exposed is called the `providers.fac` file. (See [Provider Types](#) for a description of this file). To access a single image, configure a "SimpleProvider" in the `providers.fac` and insert as JCLASS the attribute of the CREATE element, the

`com.ionicsoft.wmtmap.provider.imageProvider.SimpleProvider` class. The syntax of the `<CREATE>` tag is as follows: the mandatory element is the PARAM element having "path" as NAME attribute and the path to the image file as VALUE attribute. For example, to serve the orthophotoplan of Brussels:

Figure 133: Brussels Orthophotoplan



whose image file is named `D:/Erdas/data/images/Brussels.bil` and whose header file is named `D:/Erdas/data/images/Brussels.hdr`, the entry in the `providers.fac` is:

Example:A Simple Image Provider

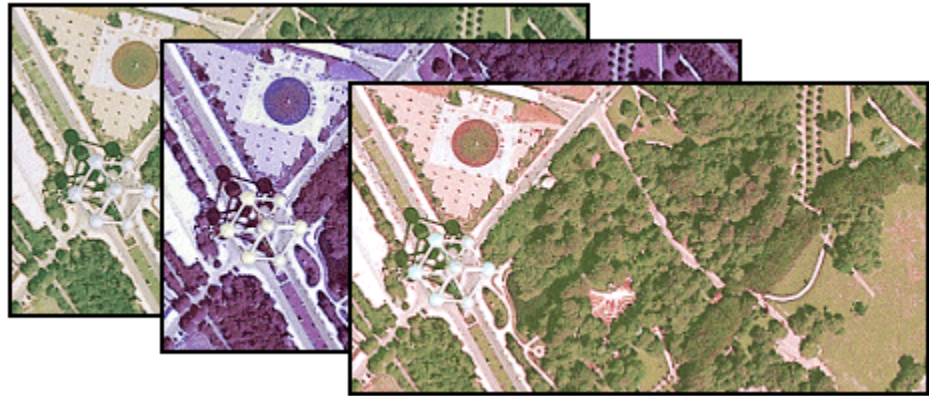
```
<CREATE ID="Brussels"

JCLASS="com.ionicsoft.wmtmap.provider.imageProvider.SimpleProvider"/>
  <PARAM NAME="path" VALUE="D:/Erdas/data/images/Brussels.bil" />
  <PARAM NAME="SRS" VALUE="EPSG:4326" />
  <PARAM NAME="name" VALUE="Brussels" />
</CREATE>
```

Other parameters can be added to the provider definition. Please refer to the appendix "Provider parameters" for more information.

To access a set of images, configure a MultiSimpleProvider in the `providers.xml` file, and to mention, as JCLASS attribute of the CREATE element, the `com.ionicsoft.wmtmap.provider.imageProvider.MultiSimpleProvider` class. The syntax of the <CREATE> tag is as follows: the mandatory element is the PARAM element that have "path" as NAME attribute. The path to the directory where the image files are is the VALUE attribute.

Figure 134: A Set of Images on Brussels



For example, to serve a collection of TIFF images on Brussels which image files are named

`D:/Erdas/data/images/brussel/spotjanuary.tif` to
`D:/Erdas/image/brussel/spotdecember.tif` , the entry in the `providers.fac` file is:

Example:Provider for a MultiSimpleProvider

```
<CREATE ID="BrusselThisYear"  
  
JCLASS="com.ionicsoft.wmtmap.provider.imageProvider.MultiSimple  
Provider"/>  
  <PARAM NAME="path" VALUE="D:/Erdas/data/images/brussel" />  
  <PARAM NAME="SRS" VALUE="EPSG:4326" />  
</CREATE>
```

This provider permits seeing each image as a layer in the capabilities document. They can be invoked individually, but only through a single entry in the configuration file. This is the same rule for an individual image apply. The layers will be named with the full path to the image.

Configuring a Mosaic or a list of Coverages/Images

To access a collection of images and see them as a single layer, configure the `LayerProvider` in the `providers.fac` input as a JCLASS attribute of the CREATE element the `com.ionicsoft.wmtmap.provider.imageProvider.LayerProvider` class. The mandatory element to put in the CREATE attribute is the PARAM element that has "path" as the NAME attribute. The path to the directory of the index file is the VALUE attribute. For example, in order to serve a layer of BIL images whose image files are named `D:/Erdas/data/images/dallas/spot1.bil` to `D:/Erdas/data/images/dallas/spot10.bil` and whose corresponding header files exist, the entry in the `providers.fac` file is:

Example:Sample LayerProvider

```
<CREATE ID="Dallas"

JCLASS="com.ionicsoft.wmtmap.provider.imageProvider.LayerProvid
er"/>
<PARAM NAME="path" VALUE="C:\\image\\dallas"/>
<PARAM NAME="name" VALUE="Dallas" />
<PARAM NAME="SRS" VALUE="EPSG:4326" />
</CREATE>
```

As for the previous types of image providers, other parameters can be added to the provider definition. Please refer to the appendix "Provider parameters" for information.

Image Layers Index File

The index file, named "*PRIME_IDX*", must be present in the directory and have the structure as described below. If there is a layer to be accessed in a continuous way, georeferenced images are required. If not, the images location will be automatically set to 0,0 in their respective SRS and the locations of the images will all be incorrect. Create an index file. This index file can be automatically created by an indexing tool that inspects existing files in a given directory and creates the index file, named *PRIME_IDX*, on them. Also an index can be created by hand. This index file must be in the same directory as the layer images and has the following format:

- Index file organization: *PRIME_IDX*
- One header line: `H_IONIC_LAYER V <version_nr> C <image_count> <date>`
- One line per image: `<width> <height> <xmin> <ymin> <xmax> <ymax> <filename>`

Date is an integer and can be set to 0 if created by hand.
`H_IONIC_LAYER` is a fixed string and is the header indicator

For example, this is the listing of the file *PRIME_IDX* of one image layer

Example:Sample PRIME_IDX file

```
H_IONIC_LAYER V 1 C 4 200201111333
5941 7609 410764.34 4573631.48 416705.34 4581240.48
harveys_lake_pa_ne.bil
5946 7613 405532.77 4573693.41 411478.77 4581306.41
harveys_lake_pa_nw.bil
5946 7609 410679.44 4566692.97 416625.44 4574301.97
harveys_lake_pa_se.bil
5951 7613 405442.88 4566754.9 411393.88 4574367.9
harveys_lake_pa_sw.bil
```

The index is used by the image server to quickly select the images to process to create a view for the "getmap" requests.

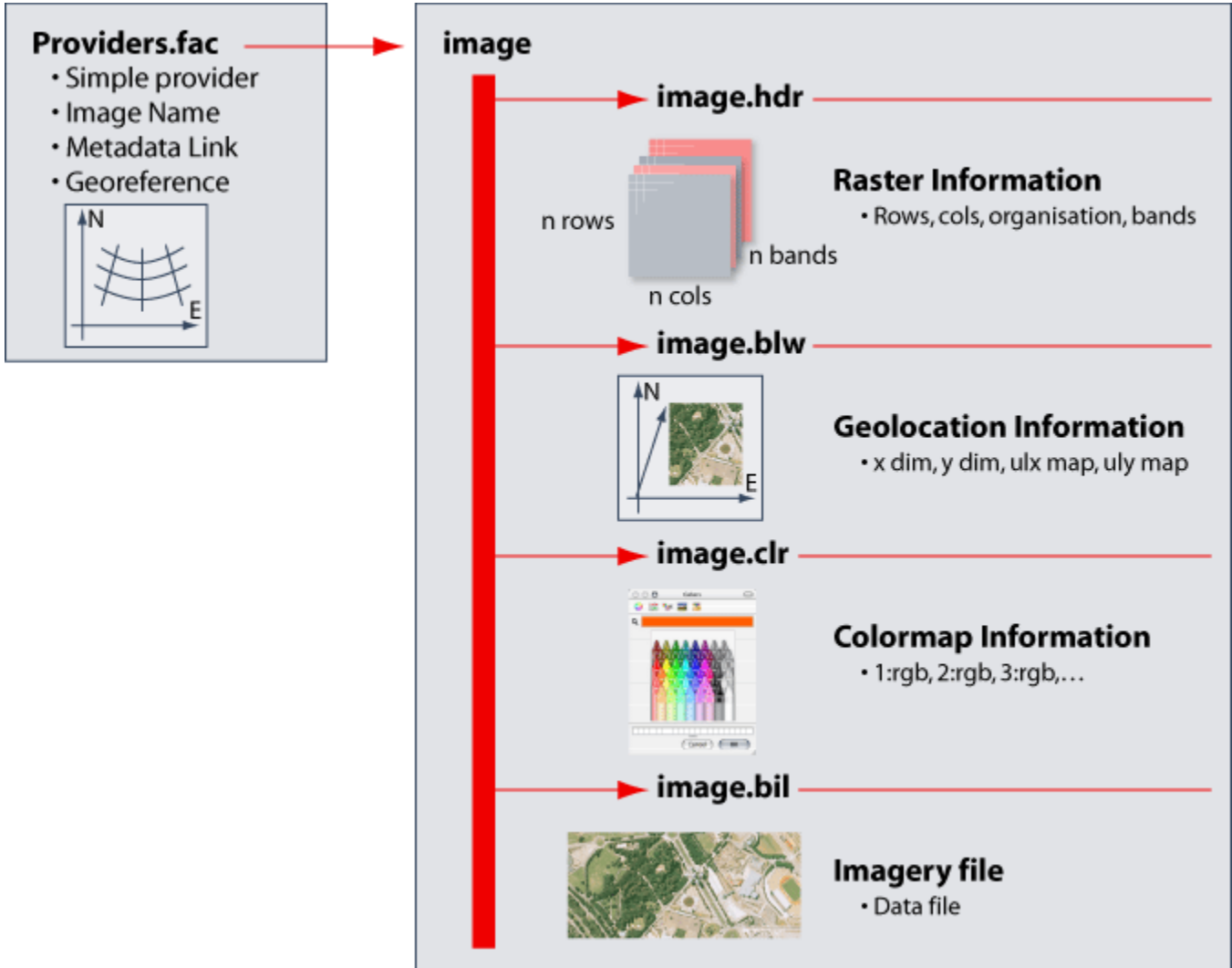
The Image Data Model

The Image server handles images retaining some of the rules for the images data model. Each image is composed of multiple parts:

- The "header" or raster information giving the image size, organization, number of bands, etc., and is often stored in a ".hdr" file, or in the image file itself.
- The "world file" or "geolocation" expressing the location of the image (its coordinate), is often stored in the world file (.tfw, .blw, etc.), in the ".hdr" file, or in the image file itself in the case of GeoTIFF.
- In the case of indexed color images, the "colormap" can be stored in the ".clr" file or in the image file itself.
- The "data" representing the samples of the images is often stored in a BIL file, or any other format (TIFF, GIF, etc.).

Each file format has its own model, header and format. The Image Server is mainly oriented to support TIFF and BIL images.

Figure 135: Example of a Data Model Organization



Each file associated with the image must have the same name as of the image file plus a well-defined extension. The rules for creating header files are described in next section.

The HDR File Organization

Except for newly defined formats like GeoTIFF, existing image file formats (SPOT, BIL, JPEG, GIF, PNG, TIFF) do not contain georeferencing information in the data file. This means that some header files have to be created or obtained to allow correct positioning of the image in a reference coordinate system. Additionally, for BIL images, the data file contains only raw data and the image information and metadata is found in the header files which is a text file with ".hdr" extension.

Example: Example of a Bil Image Setup

there is raw image, e.g., named myimage.raw. Rename it with the ".bil" extension. Create a new myimage.hdr text file, and enter the following image properties in that file:

```
nrows 8568
ncols 9576
nbands 3
nbits 8
layout bil
```

- nrows and ncols mention the height and width in pixels of the raster image
- nbands gives the depth of the image. nbits give the size of one pixel in bits
- layout indicates the organization and can be "bil", "bip", "bib" or "bsq"

Figure 136: Bil Bands

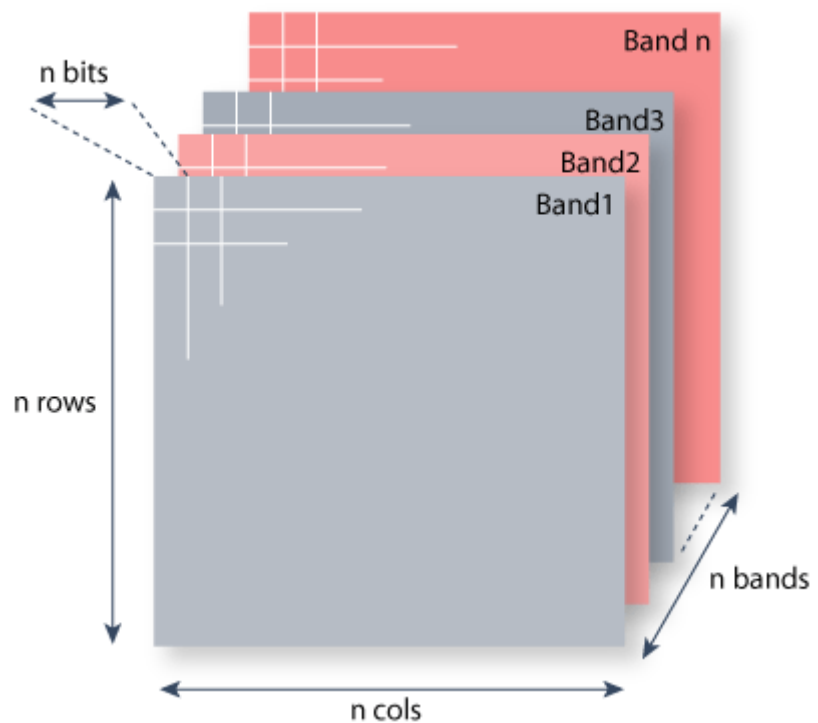


Table 59: Parameter Names and Descriptions

Parameter name	Type	Description	Example	Version
nrows	Integer	Number of rows of this raster image. This record provides the number of rows (lines) in image raster file. It is also often referred to as the height of the image.	8568	All
ncols	Integer	Number of columns of this raster image. This record provides the number of pixels (samples, columns) in each row of the image raster file. It is often also referred to as the width of the image.	9576	All
Nbands	Integer	Number of spectral bands of this raster image. This record defines the number of spectral bands present in the raster image file. Typical values are: <ul style="list-style-type: none"> • 1 for black and white or greyscale images (e.g. SPOT Pan) • 3 for RGB color images (e.g. SPOT XS images or aerial photography) • 4 for SPOT Xi acquisitions or Vegetation images • 7 for Landsat images 	3	All
Nbits	Integer	Number of bits per pixel of this raster image. This record provides the number of bits used for each pixel of each band of the raster image. By default nbits=8 (a byte). It is recommended to provide this value, even if the value is 8.	8	All
Layout	String	<i>Bands layout for multiple band rasters.</i> This keyword describes the internal storage scheme used for the raster data. In case the number of spectral bands (nbands) is higher than 1, it is necessary to describe the scheme used to mix the bands of the raster. Currently there are three possibilities: <ul style="list-style-type: none"> • BIL: Bands Interleaved per Lines • BIP: Bands Interleaved by Pixels • BIB or BSQ: Bands Interleaved by Band or Band Sequential When using standard file formats such as TIFF or JPEG this information is stored into the raster file and there is no need to use this keyword. A special paragraph is dedicated to this topic.	Bil	All

Table 59: Parameter Names and Descriptions (Continued)

Skipbytes or header	Integer	Number of bytes to skip at the beginning of the raster file. This keyword provides the number of bytes to be skipped at the beginning of the raster file to get to the first row of pixels. Both names are valid and have the same meaning and exist due to some format support. It is not good to have both, but if both are defined, the current behavior is to use max(skipbytes, header)	0	All
Color_mapping	Struct	Parameter to express the spectral bands that are to be used as red green and blue. If the raster is composed of 4 bands, numbered 0 1 2 3, designate which bands are going to be used by the system as R G B bands. If 3 0 1 are to be used, write "color_mapping 3 0 1" in the file. Sometimes, there is a gain and bias associated with a band. This means that the value in the raster has to be corrected for display. The value used can be updated by the following formula: Result = (pixel value / gain) + bias. If the values are not specified, the values are 1.0 for the gain and 0.0 for the bias. To give the gain and bias, look at the following example "color_mapping 3 0 1; 1.2 1.4 1.1; 0 0 0" here, the gain for red is 1.2, the gain for green is 1.4, the gain for blue is 1.1; all bias are at 0 This parameter is new and may not be available in the current version.	0 1 2	2.1

Layout

The layout describes the internal storage scheme used for the raster data. In the case where the number of spectral bands (nbands) is higher than 1, it is necessary to describe the scheme used to mix the bands of the raster. Currently, the layout can be one of "bil", "bip", "bib" or "bsq". The following table explains the type of organization with 3 bands of red green blue. If there were 5 bands of temperature or vegetation, it would be the same schema.

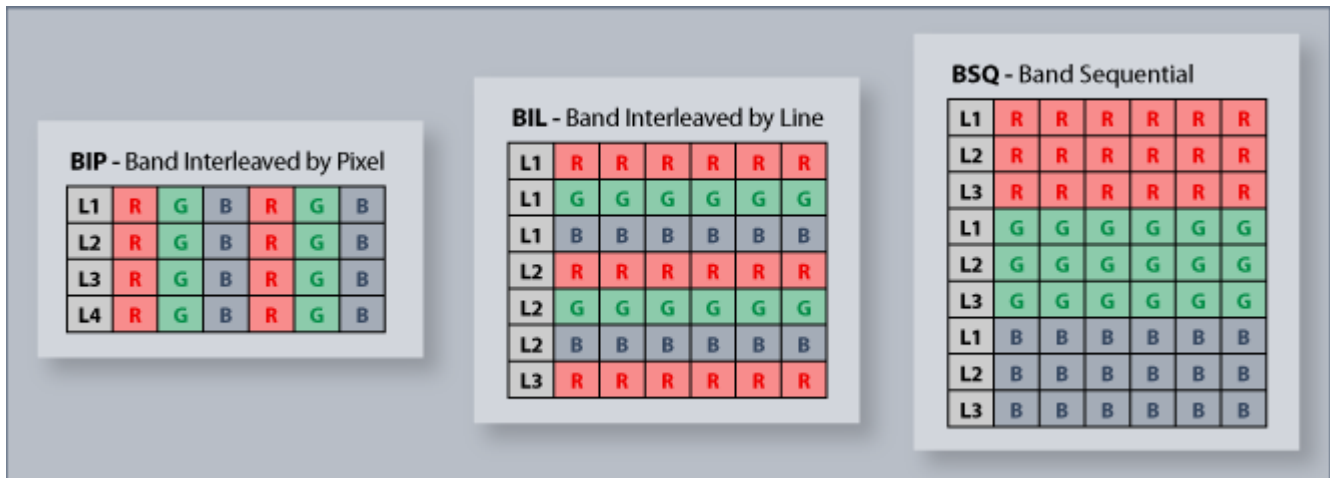
Table 60: Layout Table

layout	Means	Example	Simple description
BIL	Band Interleaved by Line	RRRRRRRRR GGGGGGGG BBBBBBBBB RRRRRRRRR GGGGG.	For the first line, all the red samples then all the samples for green, then all the samples for blue, then we have the second line.
BIP	Band Interleaved by Pixel	RGBRGBRG RGBRGBRG RGB..	For all of the line, all samples of the first pixels, then all samples of the second pixel, etc

Table 60: Layout Table (Continued)

BIB BSQ	Band Interleaved by Band Band SeQuential	RRRRRRRRRR RRRRRRRRRR GGGGGGGGG GGGGGGGGGG BBBBBBBBBBB BBBBBBBBBB	All the samples of the first band, then all the samples of the second band, etc. BIB and BSQ means the same.
------------	--	---	--

Figure 137: BIL Layout



The World Coordinate File Organization

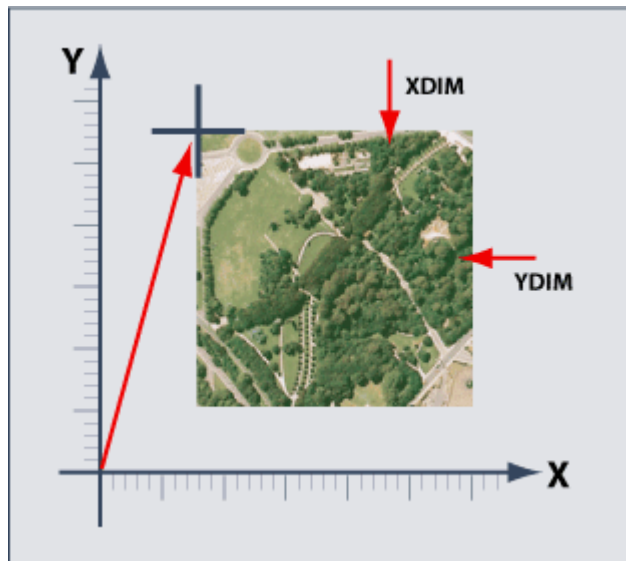
Create a new myimage.blw text file, and enter georeferencing information in it, for example:

```
1.000000000
0.0000000000
0.0000000000
-1.000000000
237583.750000000000
153205.050000000000
```

The first value is the width of a pixel in the project units (meter, degree, ...), and also called XDIM. The second and third values are for the rotation, and should be 0 because rotation is currently unsupported. The fourth value is the height of a pixel, but must be negative (YDIM) because the y pixel axis starts at the top and goes down and the coordinate axis starts at the bottom and goes up. The last two values are the x and y position of the centre of the upper left pixel. (ULXMAP and ULYMAP).

This method is called a "Geopositioning by insertion point" and the file containing the 6 values is often referred to as a world coordinate file. Geopositioning by insertion point is performed by a unique point (upper-left) and dimensions of pixel cells. The relationship between raster and coordinates is: - $X = ULXMAP + XDIM * i$ - $Y = ULYMAP - YDIM * j$ where (i,j) are floating point pixel coordinates starting from (0.0, 0.0). These equations are valid for standard axis orientations (eastwards and northwards).

Figure 138: World File



By default, the coordinate (ULXMAP , ULYMAP) is the centre of the upper-left pixel.

Naming Organization of World File

The world file must have a name that fits the type of the file, e.g., for myimage.tif, the world file is myimage.tfw; for myimage.bil, the world file is myimage.blw; for myimage.gif, the world file is myimage.giw. See the table at the end of this section, for the complete list of supported formats and the corresponding world file extensions.



For GeoTIFF images, the header information is included in the images. There is no need to create a world file. However, the SRS parameter in the providers.fac file still must be set even if it is set in the GeoTIFF header.

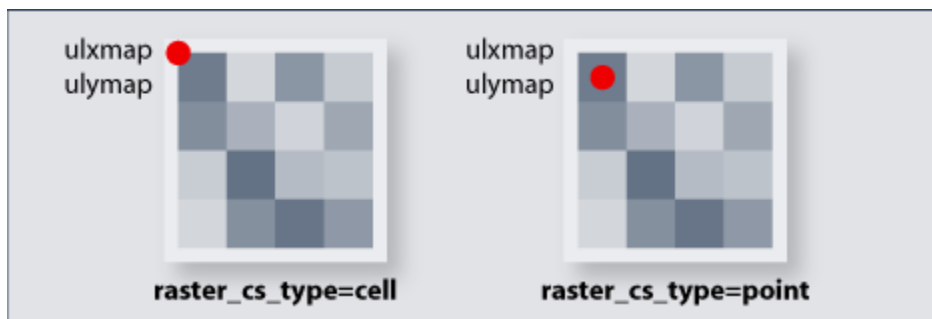
Coordinates in the Hdr File

Sometimes, these 4 values are found directly in the ".hdr" file under the tags of the same name (xdim, ydim, ulxmap, ulymap). Therefore, there is no need for the world file. To use these tags in the hdr file, look at the following table.

Table 61: HDR File Tags

Parameter name	Type	Description of parameters and their name in the hdr file	Example	Version
ulxmap	double	Upper-left pixel System X coordinate in the Coordinate Reference System. This record provides the CRS X coordinate of the upper-left raster image corner. It is expressed in the Coordinate Reference System. The sub-pixel location pointed to by ulxmap is dependant upon the "raster_cs_type" keyword value. By default, it is the centre of the pixel.	55.3	All
ulymap	double	Upper-left pixel System Y coordinate in the Coordinate Reference System. This record provides the CRS Y coordinate of the upper-left raster image corner. It is expressed in the Coordinate Reference System. The sub-pixel location pointed to by ulymap is dependant upon the "raster_cs_type" keyword value. By default, it is the centre of the pixel.	4.21	All
xdim	double	Pixel dimension along the column axis (x) in the map coordinate space. This keyword provides the (x) column axis dimension of each pixel of the raster image. The value is expressed in the CRS. This value is positive, whatever the orientation of the coordinate axis may be, e.g, a value of 20.0 would mean that one pixel has an horizontal length of 20(meters in a metric system or degrees if the CRS is geographic)	20	All
ydim	double	Pixel dimension along the row axis (y) in the map coordinate space. This keyword provides the (y) row axis dimension of each pixel of the raster image. The value is expressed in the CRS. This value is positive, whatever the orientation of the coordinate axis may be, e.g., a value of 20.0 would mean that one pixel has an vertical length of 20(meters in a metric system or degrees if the CRS is geographic)	20	All
raster_cs_type	String	Raster Coordinate System type. The raster pixel array can be considered as a series of points or as a series of adjacent cells depending on the type of data the raster is representing. This has a direct consequence on the interpretation of ULXMAP and ULYMAP keywords. This location can be located at the centre of the top-left most pixel (raster_cs_type=point) or at the upper-left of that same pixel (raster_cs_type=cell). By default, raster_cs_type=point	point	2.1

Figure 139: Raster CS Type



The Color File Organization

In some cases, the raster image is a file of indexed pixels. This means that each pixel is in a byte which value is an index in a colormap where the system can find the RGB colors to use to display that pixel. The file containing the RGB values is called the "colormap" and can be stored in an ASCII file with a ".clr" extension.



This should only work with images stored as 1 band of indexed pixels. (nbands=1, nbits=8)

Table 62: Color File Format Table

Color File format
<pre>PaletteHeader 0100 256 0 255 0 38 54 90 57 43 76 55 57 93 66 43 76</pre>
<p>Each line of 3 values is a valid entry of the palette file. The first data line (here 0 255 0) is for the index 0. The next one is for index 1. The three values represent the R G B component for the pixel index. (0 255 0 stands for green). Only the 3 values lines are required. The lines at the beginning are optional and not used, but if there are lines at the beginning, they can be made of only one word per line.</p>



If using PaintShopPro, the colormap of an image is easily seen. In the Color menu, there is a "Save palette" menu. If the palette is saved as a Jasc Palette file, a file is named "image.pal". Rename it to "image.clr" and a valid palette file will exist for the image. This tool can be used to define, read and write the palette.

Header Files Summary Table

The following table indicates the name of the header file(s) to adopt for each image format.

Table 63: Header Files Table

File format (extension)	World Geo reference file format (extension)	Description	Capabilities for Compression level	Transparency built in	Colormap
apf	apw	Deprecated	Uncompressed	Yes	
bil	blw or hdr	Fast passing Uncompressed raw format	Uncompressed	No	clr
bmp	bmw	Windows format	Uncompressed	No	
gif	gfw	! Unisys license needed. Contact Unisys.	Gif compression	Yes	Inside
jpg	jpw or jgw	Jpeg image	Variable	No	No: 24 bits
jpeg	jpw or jgw	Jpeg image	Variable	No	No: 24 bits
pcx	pcw	Pc paintbrush		No	
png	pnw	Portable Network graphics		Yes	No
ppm	ppw	Portable pixmap	No	No	
tga	tgw	Targa		No	
tif	tfw (or none for GeoTIFF)	TIFF or GeoTIFF	Uncompressed	No	Inside

USGS Metadata

If there is a TIFF or a BIL file with the HDR file header coming from USGS for Digital Orthophoto Quadrangle (DOQ), it is unnecessary to do any other processing and the ImageServer should support it directly. To see if the file comes from the USGS, open the hdr file with any text editor. If there are tags named QUADRANGLE_NAME, QUADRANT and PRODUCTION_SYSTEM, it means the files are probably coming from the USGS.

Limitations and Constraints

- File separators can be given either in the Unix form, i.e. "/" or in the Dos form "\" in which case it must be backslash-headed "\\". This rule does not apply to URLs taking only "/".
- Compressed TIFF images are currently not supported.
- Rotated images are currently not supported.
- Bands split in multiple files are not supported.

Imagery Connectors

The GDAL Tool

This section introduces the GDAL tool and how to use it in the WCS framework.

Description

GDAL Tool is a connector on the GDAL C++ library. It is a small binary that is able to decode metadata from source files and convert/subset these source files to produce WCS results. This chapter describes its installation, configuration, usage and the supported formats.

Supported formats

The GDAL Tool is plugged in the WCS, and configured to be called for specific extensions using the decoder.txt configuration file. When the WCS does not find a decoder from the source extension, or if all decoders fail, the WCS will also call the GDAL tool as a last chance attempt.



You can use the OpenEV package to view your images and check their formats: dted, geotiff, ...

Currently tested and supported input (R) and output (W) formats:

Table 64:
Table 65: GDAL-based Source Formats by Platform

Format Name	Windows	Linux	Solaris
JPEG2000	R/W	R/W	R/W
NITF	R/W	R/W	R/W
GeoTIFF	R/W	R/W	R/W
Compressed TIFF	R/-	R/-	R/-
TIFF with non-contiguous strips	R/-	R/-	R/-
ECW	R/W (500m)	R/W (500m)	-/-
MrSID	R/-	R/-	R/-
DTED	R/W	R/W	R/W
NetCDF	R/-	R/-	R/-
HDF4	R/-	R/-	R/-

Notes of formats:

- NITF: supported versions are 1.1, 2.0, 2.1
- JPEG2000 (jp2): supported through the Kakadu library. Georeferenced header are supported.
- DTED1 & DTED2: the "srs" parameter must be present in the `providers.fac`, as the srs is not always defined in DTED header.
- HDF4: providing the sub-data tiles order is defined in a .til file.

The GDAL library supports many other input formats. They should be working with the WCS/IAS framework but they have not been tested. For a complete list of GDAL supported formats, please see [GDAL Raster Formats](#). These formats include:

- VRT, HFA, ELAS, AAIGrid, PNG, JPEG, MEM, GIF, XPM, BMP, PCIDSK, PNM, ENVI, EHDr, PAux, MFF, MFF2, BT, FIT, USGSDEM..

The following commercial libraries have been added to the original GDAL package:

- The JP2KAK (Kakadu) add-on provides support for JPEG2000 with no size limitation. Note that Kakadu is a commercial library bundled with our product.
- The ECW decoding library from EARTH RESOURCE MAPPING.

Known limitations

- A NITF file is not georeferenced, it only has 4 Ground Control Points, real precision only comes with pixel space seeking (Image CRS). The SRS will be taken from the "srs" parameter in the `providers.fac` file.
- Metadata tags decoded by GDAL are not mirrored in the capabilities or in a ISO 19139 file.
- GDAL cannot encode Geotiff files with different data types in different bands, for example, all 8bits data except 1 band
- While it is possible to create a Geotiff file using 8,16,32 and 64bits data it is not possible to create:
 1. ECW > 8bits data, only byte data is supported
 2. JPEG2000 > 16bits data, only byte, short or ushort data is supported

3. DTED > 16bits data, only byte, short or ushort data is supported

- The gdal_tool stitching and resampling capabilities are not used so that heterogeneous providers that mix GDAL and non-GDAL-based formats can produce proper output. This is also the case for the IndexProvider. If a SimpleProvider or a MultiSimpleProvider is set up, each tile or granule appears individually in the capabilities and must be requested individually in the GetCoverage request, as the name of this tile will constitute the value to the COVERAGE parameter in the request. Therefore, the request will never receive hybrid formats. Furthermore, it is possible to ask GDAL to immediately produce other supported formats instead of just GeoTIFF. This is why the Capabilities document for those provider types publish additional output formats such as JPEG2000, ECW (on Windows) and NITF.

Linux and Solaris Configuration

Multiple GDAL builds are available in the Linux installation

- one static build
- one dynamic build dependant of the libstdc++.so.6 library (Linux systems compiled with gcc 3.4 et after). This is the one configured by default at installation.
- one dynamic build dependant of the libstdc++.so.5 library (for older Linux systems).

Several GDAL builds are available in the Solaris installation

- one dynamic build compiled on Solaris 8 with the Sunpro compiler and successfully running on Solaris 8, 9 and 10. This is the one configured by default at installation.



On Solaris 8, it assumes system patches 108434-22 (C++) and 112757-01 (Math) are installed. On Solaris 9, it assumes system patches 111711-16 (C++) and 111722-04 (Math) are installed.

- one dynamic build compiled on Solaris 10 with the Sunpro compiler but only running on Solaris 10.

On Linux and Solaris, a wrapper called gdal_tool is used to configure which build to use and its library dependancies (LD_LIBRARY_PATH variable configuration).

Hierarchical data sets

HDF4 data sets are composed of sub-datasets (tiles) that must be organised to create multiple bands of data on the whole set extent.

For example, Landsat HDF4 data are composed of $8*8*7 = 448$ subdatasets. In order to serve this data, the ERDAS WCS must know how these 448 sub-datasets can be organized in 7 bands of $8*8$ tiles. Indeed, GDAL does not provide this critical information.

It is then necessary to provide and configure a new metadata file for each hdf4 file, each hdf4 dataset can then be described by three files, for example:

1. Geocoded_landsat_tm193_22_890707.hdf: the hdf4 data file
2. Geocoded_landsat_tm193_22_890707.hdf.xml: the xml metadata file, encoded in ISO19115/19139 or other proprietary format
3. Geocoded_landsat_tm193_22_890707.hdf.til: the file that provide the tiles order in the hdf4 file.

The last file must have the .til extension and should contain following information:

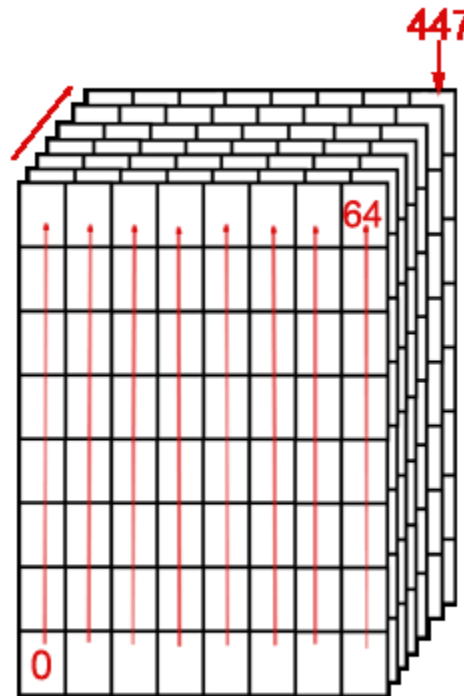
```
TILES_METADATA 2
DATA_TYPE HDF4_Landsat
714826.2,5949126.5,958730.5,6154977.5
EPSG:32632
Y -8
X 8
CHANNEL 7
```

In the above example:

- The file format is "TILES_METADATA" and its version is "2"
- The "DATA_TYPE" is "HDF4_Landsat", this data type will appear in the granule metadata and identify the SLD rule that should be used to render the granule.
- The granule bounding box is "714826.2,5949126.5,958730.5,6154977.5", this is the data complete extent when all the tiles are assembled
- The bounding box spatial reference identifier is "EPSG:32632"

- From the last three lines, we can say that the granule is composed of $8 \times 8 \times 7$ tiles, and each of the 7 bands is 8 tiles width and 8 tiles height. We can also determine the tiles order within the hdf4 data file:
 1. The tiles are first stored along the Y axis, from the lowest tile until the uppest
 2. The columns are then stored along the X axis, from the left column until the right one.
 3. The bands are then stored from the first band until the band number 7.

Figure 140: Tiles order in Landsat hdf4 dataset



GDAL Installation Notes

The installation of the GDAL Tool is automatic in the distribution. The `gdal_tool` binary is in `APOLLO_HOME/tools/native/gdal`, along with the appropriate dynamic libraries.

A set of dynamic libraries (.dll or .so files) are provided along with the tool to support proprietary formats like ECW or MrSID. On Windows, no additional configuration is needed to access those libraries when they are located in the same directory as the gdal_tool executable. On Linux, the LD_LIBRARY_PATH environment variable must be set to that same path in order for the .so files to be found. That variable should be set in the servlet engine startup script to ensure proper propagation up to the gdal_tool executable. To avoid that dependency for users not interested in particular data types, the static executable is activated by default. To use the dynamic one instead, rename gdal_tool as gdal_tool_static, and rename gdal_tool_dyn as gdal_tool. It may also be necessary to create symbolic links from .so.1 files to the existing .so files.

ERDAS APOLLO Image Manager and ERDAS APOLLO currently provide binaries for the following platforms:

- Linux (static and dynamic)
- Windows
- Solaris

GDAL Configuration

1. Servlet Configuration

The `providers.fac` entry has to mention:

- the GDAL directory path: parameter '`GDALPath`', the directory where the gdal_tool binary is copied
- the temporary directory path: parameter '`tmpPath`', the directory where the temporary directories will be created

If some of the configuration is missing, error messages arise:

- If `GDALPath` is not set: `javax.servlet.ServletException: Cannot construct provider NITF1`
`com.ionicssoft.api.util.PropertyNotFoundException: Property Not Found: GDALPath`
- If `GDALPath` is set but does not point to a directory where the binary is: `javax.servlet.ServletException: Cannot construct provider NITF1`
`com.ionicssoft.api.wcs.CoverageConfigurationException: Error : gdal metadata decoder tool`
`C:/Erdas/Apollo/tools/native/gdal/gdal_tool.exe can not be found !`

2. Sample providers.fac File

```
<CREATE ID="GDALTEST"
JCLASS="com.ionicsoft.wmtmap.provider.coverage.SimpleProvider"
>
  <PARAM NAME="name" VALUE="GDALTEST"/>
  <PARAM NAME="title" VALUE="NITF Coverage Server"/>
  <PARAM NAME="abstract" VALUE="NITF Data"/>
  <PARAM NAME="metainfo" VALUE="" />
  <PARAM NAME="keywords" VALUE="nitf,gdal,test,Coverage"/>
  <PARAM NAME="GDALPath" VALUE="D:\resin-219\webapps\wcs\WEB-
INF\resource\GDAL\" />
  <PARAM NAME="tmppath" VALUE="D:\resin-219\webapps\wcs\WEB-
INF\resource\GDAL\temp\" />
  <PARAM NAME="path" VALUE="
  \Server\DataArchive\CoverageData\NITF\03SEP01022335-M1BS-
000000102191_01_P002.NTF.rv2" />
  <PARAM NAME="backgroundValue" VALUE="0" />
</CREATE>

<CREATE ID="GDALTEST1"
JCLASS="com.ionicsoft.wmtmap.provider.coverage.SimpleProvider"
>
  <PARAM NAME="name" VALUE="GDALTEST1"/>
  <PARAM NAME="title" VALUE="DTED Coverage Server"/>
  <PARAM NAME="abstract" VALUE="DTED Data"/>
  <PARAM NAME="metainfo" VALUE="" />
  <PARAM NAME="keywords" VALUE="dted,gdal,test,Coverage"/>
  <PARAM NAME="GDALPath" VALUE="D:\resin-219\webapps\wcs\WEB-
INF\resource\GDAL\" />
  <PARAM NAME="tmppath" VALUE="D:\resin-219\webapps\wcs\WEB-
INF\resource\GDAL\temp\" />
  <PARAM NAME="path"
VALUE="\Server\DataArchive\CoverageData\DTED\N32.DT2" />
  <PARAM NAME="srs" VALUE="EPSG:4326" />
  <PARAM NAME="backgroundValue" VALUE="-32767" />
</CREATE>
```

3. Using GDAL encoding abilities to enable new output formats

The output formats advertised in the DescribeCoverage document of the WCS interface defines the possible encoding offered by this interface. The list of the available output formats results from GDAL encoding abilities and a configuration file telling which formats to expose. That file can be modified to offer more output formats. The file to modify is called 'gdal_formats.prop' and should be found in the WCS servlet classpath, in the package `com.ionicsoft.keys.resource` to take effect. Below is an example of `gdal_formats.prop` file:

```
# list of supported GDAL output formats,
# the formats names match GDAL driver short names

# already tested formats
```

```
FORMAT1=GTiff
FORMAT3=NITF
FORMAT2=JP2KAK
FORMAT4=DTED
FORMAT5=ECW

# not tested formats

# FORMAT6=USGSDEM
# FORMAT7=VRT
# FORMAT8=HFA
# FORMAT9=ELAS
# FORMAT10=AAIGrid
# FORMAT11=MEM
# FORMAT12=XPM
# FORMAT13=PCIDSK
# FORMAT14=PNM
# FORMAT15=ENVI
# FORMAT16=EHdr
# FORMAT17=PAux
# FORMAT18=MFF
# FORMAT19=MFF2
# FORMAT20=BT
# FORMAT21=FIT
# FORMAT22=HDF4Image
```

The five first output formats are the one usually available through the WCS interface. These encoding are tested and supported. The other formats are other encoding formats proposed by GDAL. These encoding have never been tested and ERDAS does not guarantee that these will work if they are enabled.

To enable a new output format, just remove the '#' sign in front of the desired format line.

Using GDAL Library

The decoding framework of ERDAS servlets will first try to use ERDAS's own decoders based on the file extensions. If no known extension is found or if the files cannot be parsed the framework makes a call to **gdal_tool**.

Testing the gdal_tool Binary

The tool has two functionalities:

- decoding metadata with the metadata tool
- converting/subsetting data with the convert tool

1. metadata tool

The following command will create an ASCII file (`destFilePath`) with the source (`sourceFilePath`) metadata:

```
gdal_tool -meta -s sourceFilePath -o destFilePath
```

Here is a sample output:

```
Source=aurora-1010010002DFDB00.tif

Driver=GTiff
DriverName=GeoTIFF

PixelSize=(913,789)

Srs=EPSG:4326

GridOrigin=(-105.662293,40.052810)

PixelRes=(0.00139209,-0.00139209)

Box_UpperLeft=(-105.662293,40.052810)
Box_LowerLeft=(-105.662293,38.954448)
Box_UpperRight=(-104.391312,40.052810)
Box_LowerRight=(-104.391312,38.954448)

NBands=3
BlockSize=(913,8);(913,8);(913,8)
DataType=Byte,Byte,Byte
ColorInterp=Red,Green,Blue

HasMetadata=false
```

2. convert tool

The convert tool is able to convert/subset/transform source files via a simple command like:

```
gdal_tool -convert -srcwin 0,0,srcPixelWidth,srcPixelHeight -
outside 400 400
-s sourceFilePath -o destFilePath
```

This will create a 400*400 Geotiff file (`destFilePath`) with the same BBOX and the same number of bands as the source data (`sourceFilePath`).

Testing in the WCS servlet

Depending on the decoder definitions set in the `decoder.txt` configuration file located in the `ionic_cots.jar` in `com/erdas/coverage/decoder/resource/`, some file extensions are automatically managed by GDAL. Moreover, if the extension is finally not recognized, GDAL will be invoked as last-chance decoder. If you want to check that GDAL is properly configured, set up a Simple coverage provider on top of a coverage file which extension is changed to `.foo`.

GDAL is also used to produce outputs in NITF, DTED, JPEG2000 and ECW. If you successfully run a GetCoverage request asking for one of those formats, you can be sure that GDAL is properly invoked for outputs.

Very Large Coverage Manager

Usually, the size of data that can be processed at a time is limited to 2 or 4Gb in most applications.

Thanks to the Very Large Coverage Manager, the ERDAS WCS is able to process data of any size through the whole chain, from input, through subsetting, mosaicing, ..., portraying, until the output. The working size is only limited by the hard drive storage capacity.

The ERDAS WCS is able to output a 50Gb Geotiff files by mosaicing thousands of big source files.

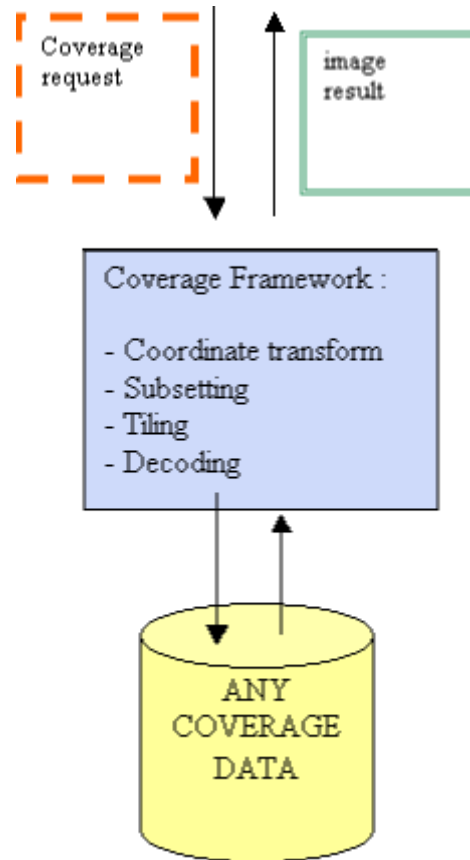
Very Large Coverage Management Description

The WCS framework is based on J2ee Raster API. The Raster object manages its data using a single array per band. This means that the number of pixels per band is limited to the Integer maximum value, as the index of the array is an integer, so:

```
(pixel width * pixel height) < integer maximum value (2147483647)
```

For example, a Java Raster of pixel sizes [46340*46340] is possible but a Java Raster of pixel sizes [46341*46341] is impossible. The (VLCM) has been created to manage bigger coverage grids through the whole coverage processing chain. Usual WCS request processing chain is following:

Figure 141: WCS Process Chain



Any WCS provider can call the VLCM if the request output sizes are bigger than a defined number. The following guard decides the VLCM call:

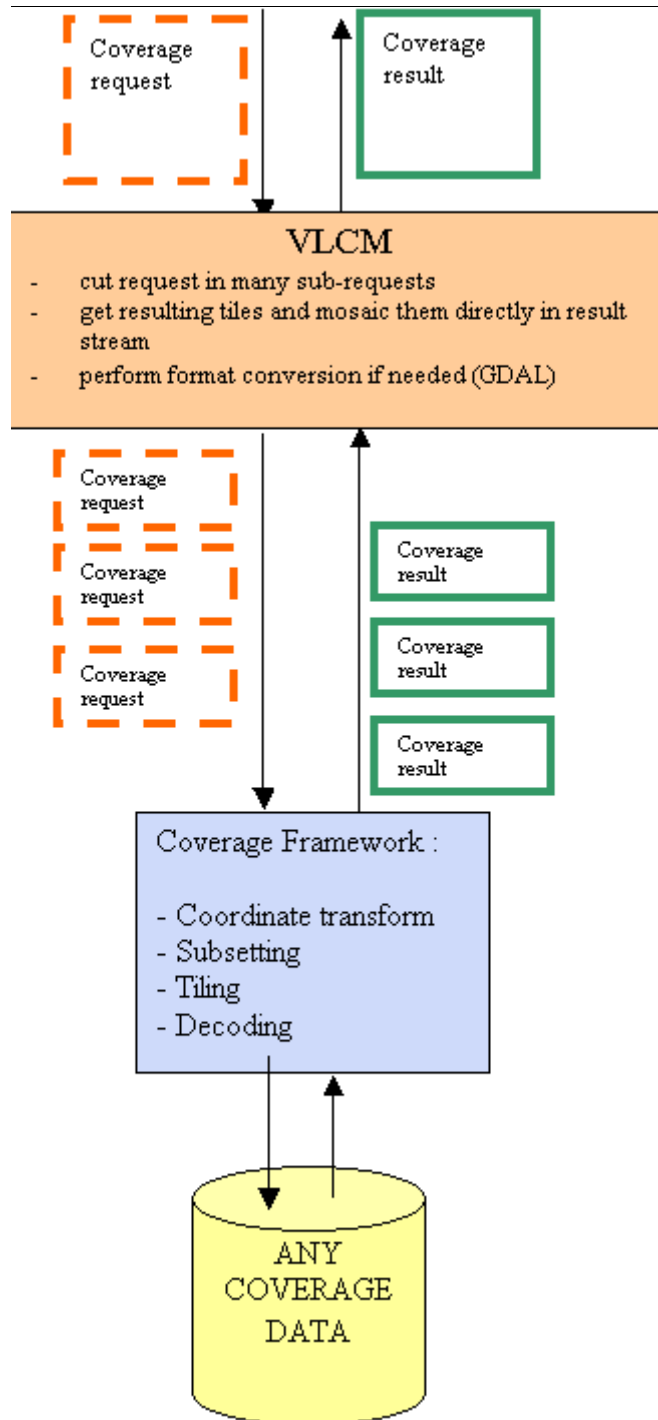
```
If(pixel width * pixel height > VERYLARGETRIGGER *  
VERYLARGETRIGGER)  
Call VLCM;
```

That number can be configured in pixels using the VERYLARGETRIGGER parameter in the WCS `providers.fac` entry (default is 5000):

```
<PARAM NAME="VERYLARGETRIGGER" VALUE="2500" />
```

The VLCM is working on top of the coverage framework, so that any of its abilities (coordinate transform, stitching, decoding,...) are available for very large coverage processing:

Figure 142: Very Large Coverage Processing



The VLCM will perform multiple requests on the WCS framework to get small tiles (tiles with less than `VERYLARGETRIGGER * VERYLARGETRIGGER` pixels each). Each tile will be directly stitched in the result stream. If the requested output format is GeoTIFF, the result stream is directly returned through the servlet engine. If not, the stream is written onto disk as a Geotiff (+vrt) file. GDAL is then called on that file to produce the JPEG2000 or NITF file. Finally GDAL's result is streamed back through the servlet engine.

The VLCM will work with any WCS provider (SimpleProvider, IndexProvider, ...) on any type of coverage data (GeoTIFF, NITF, ...).

If the parameter STORE is set to TRUE in the WCS request, the WCS will return an XML file with the result description and an URL pointing on that result.

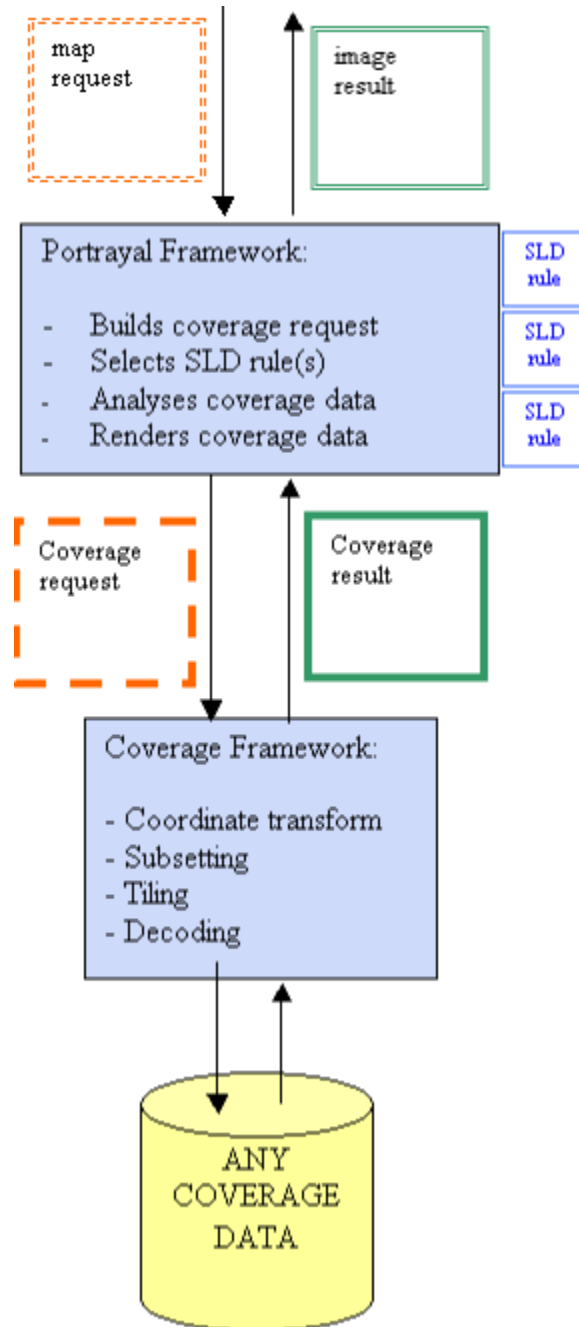
Per WCS request:

- The VLCM will not consume more memory than the one needed for two or three tiles (three if coordinate transform is needed), i.e. $3 * \text{VERYLARGETRIGGER} * \text{VERYLARGETRIGGER} * \text{bits_per_sample} * \text{number_of_bands}$ bytes.
- The VLCM will not consume more hard-drive space than the one needed for two or three times the uncompressed result (three if STORE=TRUE is used), i.e. $3 * \text{pixel_width} * \text{pixel_height} * \text{bits_per_sample} * \text{number_of_bands}$ bytes.

Very Large Image Management

The Very Large Image Manager (VLIM) has been created to manage very big map requests on WCS/CPS, through the whole coverage processing and rendering chain. Usual WCS/CPS processing/rendering chain is following:

Figure 143: WCS/CPS processing/rendering chain



The VLIM will be called by any WCS provider if the map request output sizes are bigger than a defined number (`VERYLARGETRIGGER` parameter in `providers.fac`, default is 5000).

The VLIM is working on top of the portrayal framework, so that any of its abilities (channel selection, contrast enhancement, look-up tables,...) are available for very large coverage processing and rendering.

The VLIM will perform multiple requests on the CPS framework to get small tiles (tiles with less than `VERYLARGETRIGGER * VERYLARGETRIGGER` pixels each). Each tile will be directly stitched in the result stream. If the requested output format is GeoTIFF (Geotiff image format), the result stream is directly returned through the servlet engine. If not, the stream is written onto disk as a Geotiff (+vrt) file. GDAL is then called on that file to produce the PNG, JPEG, GIF or BMP image file. Finally GDAL's result is streamed back through the servlet engine.

If some contrast enhancement is requested to render the coverage data (Normalize or Histogram), some control must be applied on the CPS to avoid inhomogeneous rendering on the stitched result, as shown in the example below:

Figure 144: Without rendering control

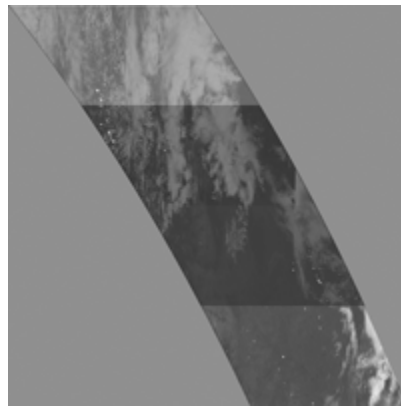
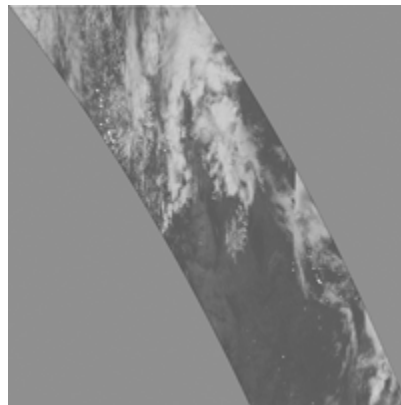
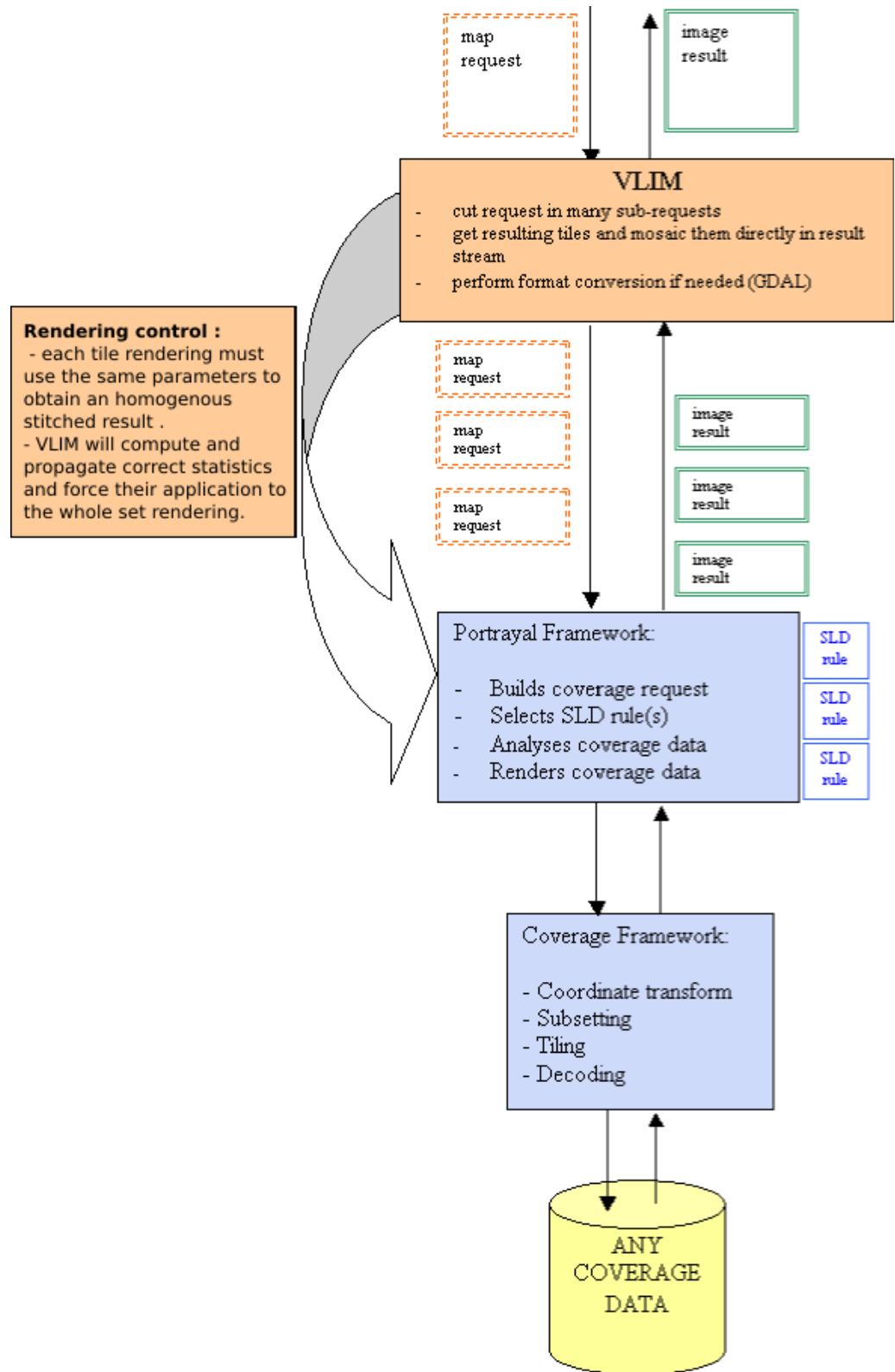


Figure 145: With rendering control



To avoid this problem, each tile rendering must use the same parameters. The VLIM will compute and propagate the correct statistics and force their application to the whole set rendering.

Figure 146: Very Large Image Management process



The VLIM will work with any WCS provider (SimpleProvider, IndexProvider, ...) on any type of coverage data (GeoTIFF, NITF, ...).

Per WMS request:

- The VLIM will not consume more memory than the computed using the following formula, i.e. $2 * \text{VERYLARGETRIGGER} * \text{VERYLARGETRIGGER} * \text{coverage_bits_per_sample} * \text{coverage_number_of_bands} + 2 * \text{VERYLARGETRIGGER} * \text{VERYLARGETRIGGER} * 4 \text{ bytes}$.
- Note that the VLIM consumes more memory than the VLIM for the same pixel sizes requested. If an OutOfMemory exception is thrown, the `VERYLARGETRIGGER` value must be reduced.
- The VLIM will not consume more hard-drive space than the one needed for two times the uncompressed image result, i.e. $2 * \text{pixel_width} * \text{pixel_height} * 4 \text{ bytes}$.

If the data set is homogeneous (bands count, data type), it is possible to accelerate and enhance the quality of the Normalize contrast-enhancement filter, it is possible to specify the minimum and maximum values of the whole coverage data set in the provider entry, for example:

```
<PARAM NAME="DataStatistics" VALUE="7,1,-100,16000" />
```

Where

- 7 is the number of bands of the data set
- 1 is the data type (unsigned short)
- -100 is the data minimum value
- 16000 is the data maximum value

The data value is found in the following table:

Data type	value
byte	0
unsigned short	1
short	2
integer	3
float	4

double	5
--------	---

Doing this, the same parameters will always be applied to the Normalize filter and the statistics gathering will be avoided, speeding up the rendering.

Note that, it is not possible to specify the Histogram filter parameters, as they are totally dependant of the data to render.

If no contrast enhancement is necessary to render the source coverage data, it is better to remove the normalize filters from the SLD rules, this will speed up the rendering.

Temporary Files Can Be Very Big

To ensure no disk full will arise, make sure your temporary directories are located on a disk with enough free space:

- Set `tmppath` parameter in your WCS provider: `<PARAM NAME="tmppath" VALUE="E:/storage/">`. This will be used to store GDAL results and the result Geotiff when the output format is not GeoTIFF.
- Set `TEMPMANAGER` to the properly sized disk in the configuration part of your `providers.fac`: `<TEMPMANAGER DIR="E:\storage2\" />`. This will be used to store the result if the `STORE=TRUE` option is used (any format).
- Set the Java temporary directory: launch the servlet engine using following option: `-Djava.io.tmpdir="E:/storage3/"`. This will be used to store the result Geotiff stream when the output format is GeoTIFF.
- If `TEMPMANAGER` is not set, the temp files could be in `TOMCAT_HOME/work/.../erdas-apollo`.
- You can verify that the `java.io.tmpdir` VM variable is properly set. Its value is visible in the output of the `request=debug&cmd=env` request on the servlet.
- If you choose to use the `STORE=TRUE` option, the final image will be copied to the final location. Meaning that it will be found twice on the disk during a few seconds.

Configuration

The choice of the `VERYLARGETRIGGER` parameter value is critical:

- If `VERYLARGETRIGGER` value is too small, the manager will be called when it is not necessary. The processing time is bigger when the manager is involved as the work is done on the hard drive.

- If VERYLARGETRIGGER value is too big, some OutOfMemory exception will be thrown if the allocated RAM is smaller than $(n * \text{VERYLARGETRIGGER}^2)$ bytes (see A. and B.).

Limitations

- We do not use the internal GDAL stitching capabilities. It means that for n granules, n calls will be made to GDAL.
- For very large output (larger than 5000x5000 pixels), the WCS uses a disk-based output production, to avoid OutOfMemory errors. In that case, the number of calls to GDAL will be larger, with at least one per granule. More precisely, by default, the WCS will extract pieces of granules of less than 25000 pixels.
- The VLIM GIF output will work only if the source coverage data has only one band. The VLIM cannot output colored GIF files.
- Known VLCM output sizes limits:

format	limits
NITF	2GB

- Known VLIM output sizes limits:

format	limits
BMP	4GB
GIF	2GB
JPEG	4GB (max dimensions 65500x65500)

Issues

- Resin and Tomcat don't seem to be able to output very large images provided by our servlets. They rapidly produce a OutOfMemory error. We currently found no other turnaround than requesting STORE=TRUE, and then retrieving the file manually.
- When the output is produced in the temporary JVM directory, it is kept for a while. We still have to investigate if it is a fixed period, or until next query, or other.

Examples

- All the CIB NITF granules (468) requested (all extent, 35328x52223 pixels) in GeoTIFF on a heavy-loaded Windows 2000 machine, with option STORE=TRUE. 40 minutes (including 5 minutes for copying the file into the final directory). The output is 1.8 GB.
- Single 200 MB Geotiff, on a Windows 2000 machine. The output format is JPEG2000, 50000x50000 pixels. 15 minutes (including 10 minutes for GDAL to convert the produced Geotiff into JPEG2000 by GDAL). The output is 214 MB.

- Single 200 MB Geotiff, on a Windows 2000 machine. The map request output format is PNG, 35000x35000 pixels (3 bands). 18 minutes (including 7 minutes for GDAL to convert the produced Geotiff into PNG by GDAL). The output is 93 MB.

Advanced Configuration

This appendix describes some advanced data configuration aspects for the Advanced Scenarios chapter. As the scenarios chapter details step-step procedures, this chapter describes the additional configuration necessary to set up these scenarios.

Metadata URL

As soon as Metadata are loaded or encoded using your preferred ISO 19139 Metadata Encoder tool, it is possible to link them to the WMS or WFS through MetadataURL entries in the Capabilities document. This section explains the details of this configuration.

Templates

The template will be a standard template using {} to define parameters. The following parameters will be defined

- {name}: replaced by the layer name, the feature name
- {metaname}: same behavior as {name}, kept for backward compatibility
- {host}: replaced by the servlet host name
- {port}: replaced by the servlet port
- {context}: replaced by the context path and always starts with a / or is empty
- {servlet}: replaced by the servlet path and always starts with a / or is empty
- {id}: replaced by the provider id
- {server}: the server path that is equivalent to `http://{host}:{port}{context }{servlet}/{id}`

This definition is used in all references to metadata/legend templates in this document (in the default specification the association to a WMS layer, and the association to a WFS feature type).

GLOBAL TEMPLATE

The provider factory file may contain a METADATA section that may define the default metadata document location template used by all providers. This default will be defined through the TEMPLATE attribute.

Storage

The metadata can be stored on a specific database or in a file system. The servlet uses different mechanisms to handle file system and database storage:

For *Database storage*, it is up to the user to define the appropriate URL. An example can be "http://{host}/GetMetadata ID={id}&TYPE={name}". A WFS can be a good candidate for this kind of (remote)storage.

For *File storage*, the metadata associated with one element (FeatureType, Layer) are supposed to be defined in a single file. This file is identified by the provider ID and the element name(Note: the file system(s) is defined for all providers).

This file can be stored onto a user defined directory (absolute path on the file system). The absolute directory path should be provided with the DIR attribute of the METADATA and the template should be defined as {absolute}{id}/{name}.xml. The {absolute} is mandatory to mark the use of the global file system, it will be replaced by "". In order to work, the user running the container (servlet) engine must read the permissions on that directory.

This file can also be stored onto a servlet relative directory, i.e., relative to the resource package under the servlet package. The template should be defined as {relative}{id}/{name}.xml. (the .xml is part of the template). The {relative} is mandatory to mark the use of the relative file system. It will be replaced by "". In both cases of file storage, the final URL (the one written in the capabilities) will be an http request using an internal mechanism to retrieve those files.



The servlet will check if the files are available in the expected directory with the expected name. If absent, the tag is not set in the capabilities document, except if the "AlwaysGenerateMetadata" tag is set in the mapping file.

Metadata Configuration in the WMS and WCS Servlets

Each provider can have a METAURL entry which specifies the metadata template. If this entry is empty or equals to "inherit", the global template will be used. The defined template will be used to define the metadata URL associated to each layer of the provider. It is not possible to associate a metadata to specific layers only.

Example:Sample Providers.fac for Metadata URL

```
<CREATE ID="PROXYNASA"
  JCLASS="com.ionicsoft.wmtmap.provider.proxy.ProxyProvider">
  <PARAM NAME="URL" VALUE="http://cost.gsfc.nasa.gov:8250/viz-
bin/wmt.cgi" />
  <PARAM NAME="METAURL" VALUE="" />
  <!-- used the default value -->
</CREATE>
```



```

<CREATE ID="ATLANTA_IMAGE"
  JCLASS="com.ionicsoft.wmtmap.provider.image.SimpleProvider">
  <PARAM NAME="path" VALUE="home/Erdas/ApolloServer/data/erdas-
apollo/images/atlanta_landsat/40379914.ecw" />
  <PARAM NAME="METAURL" VALUE="{relative}/{name}.xml" />
</CREATE>

<CONFIGURATION>
  <METADATA TEMPLATE="{absolute}{id}/{name}.xml"
    DIR="/home/Erdas/ApolloServer/config/erdas-
apollo/metadata/map" />
</CONFIGURATION>

```

Metadata Configuration in the WFS Servlet

The metadata template is defined in the mapping file and is explicitly associated to each layer or feature type. The `<Metadata>` tag (under the `Info` tag) defines the template for the specific feature type. If the content of the tag is empty or equals "inherit", a default template will be used. If the tag is not defined, the feature type has no metadata URL. The mapping file may contain a `<Option>` tag whose `<DefaultMetadata>` child defines the default template appropriate for this provider. This enhances the standard mechanism (which defines a template for ALL providers) by defining a template per provider. The default template is computed as follows: if a default is provided through the `Option` tag, it is used (including empty, "inherit" and "nometadata" values; otherwise, the default provided through the factory file is used.



The `<Metadata>` tag holds a "type" attribute that lets you publish, in the WFS capabilities, the type of the metadata document. Allowed values are: TC211 (the default), 19115, 19139 and FGDC.

Example: Sample Mapping File for Metadata URL

```

<xsd:schema>
  ...
  <Option>
    <!-- define the provider default template-->

    <DefaultMetadata>http://www.metadata.org/Get?ID={name}</Default
Metadata>
  </Option>
  <Mapping >
    <SQL name="wfs:ESA_FIRE">
      <Primary name="ESAF_ID" type="xsd:integer" fid="true"/>
      <Element name="NDVI" nameSQL="ESAF_NDVI" />
      <Element name="STATION" nameSQL="ESAF_STATION" />
      <Geometry name="Geometry" nameSQL="GEOM"/>
    </SQL>
    <Info name="wfs:ESA_FIRE">
      <SRS>EPSG:4326 EPSG:4327</SRS>
      <BoundingBox SRS="EPSG:4326" minx="-180." miny="-90."
maxx="180." maxy="90." />

```

```

<Metadata/>
<!-- use default one-->
</Info>
<Info name="wfs:ESA_FIRE2">
  <SRS>EPSG:4326 EPSG:4327</SRS>
  <BoundingBox SRS="EPSG:4326" minx="-180." miny="-90."
    maxx="180." maxy="90." />
  <Metadata
type="19139">http://{host}/MetadataServlet?REQUEST=GetFeature&a
mp;TYPENAME=METADATA&amp;ID={name}</Metadata>
  </Info>
</Mapping>
...
</xsd:schema>

```

Legend URL

The OGC WMS capabilities documents can contain URLs to small bitmaps that represent Legend snippets. This is intended to permit the construction of a legend in any kind of application.

Similar to the "MetadataURL" system, ERDAS permits further configure of the services to publish LegendURL tags in the WMS capabilities documents. But an additional parameter, named {style}, is also available to build icons whose name contains the style name.



The servlet will check if the files are available in the expected directory with the expected name. If absent, the tag is not set in the capabilities document, except if the "AlwaysGenerateLegend" tag is set in the mapping file.

The Map Generation Transformer

Introduction

The Map Generation (MapGen) transformer is an additional configuration in the WFS mapping file used for changing the default server response to the GetMap request. This configuration can speed up processing at certain scales, extract unnecessary data columns and apply conditional filtering, grouping and ordering clauses to the original requests.



Please ensure that the Feature Mapping process detailed in [GML Application Schema and Mapping to Databases](#) and [Feature Mapping Tags](#) is fully understood before using the MapGen transformer.

Using MapGen

This section describes examples where Map Generation Transformation can be used to ensure smart and fast output. The use cases described here are only a few examples and one may find other practical applications of MapGen. The main aim of MapGen is to improve the map generation performance by not referencing unnecessary information from the data source.

Properties Selection

When mapping GI data, sometimes only the feature's geometric property, or location information is of interest.

Imagine a simple world map displaying only the boundaries of countries is to be created. However, there is a set of linear data for country boundaries that also includes several other properties such as population and economic statistics.

A GetMap that queries only the geometry will greatly improve performance. However, a GetFeatureInfo query needs to be performed on a country and all the attribute values must be retrieved. Without a particular configuration, the GetMap request will extract all property values and geometry. If there are too many properties to retrieve, the GetMap request will return with poor performance. Using MapGen, select the specific properties to be employed for map generation.

In the example below, the CITY feature has many properties (ADMIN_LEVEL, POPULATION, ECOSTAT, BOUNDARY). For each MapGen block, the <Field> tags mention only the properties that will be extracted in GetMap requests.

Scale Dependent Filtering

The goal is to display cities in the world map. At the world scale, all cities cannot be displayed as this will result in thousands of points overlapping thus rendering the map unreadable. Moreover, executing this GetMap request may return poor performance due to the large quantity of features.

Therefore, to logically portray cities, extract only the major cities at world level and then display more and more smaller cities depending on the zoom level. This can also be done with MapGen by using feature properties filters for each specified scale range.

In the example below, the CITY feature has several MapGen blocks, one per scale range. A <Where> tag expresses a filtering condition for each block so that only the cities with the adequate administration levels are extracted in each scale range.

Scale Dependent Geometries

Imagine that the cities to display actually have two geometries: a point to denote the center of the city and a polygon that denotes the city's extent. Using MapGen, either the polygon or point can be displayed depending on the scale of the map.

In the example below, the CITY feature has one published geometry, named "Boundary" but the underlying table has a second geometric column, named "CENTER". In the first MapGen block (which denotes the highest scale), the BOUNDARY column has been replaced with the CENTER column.

Example of MapGen for the CITY Feature Type

```
[...]
<Mapping>
  <SQL name="wfs:CITY">
    <Primary name="ID" nameSQL="ID" />
    [...]
    <Element name="ADMIN_LEVEL" nameSQL="LEVEL" />
    <Element name="POPULATION" nameSQL="POP" />
    <Element name="ECOSTAT" nameSQL="STAT" />
    <Element name="Boundary" nameSQL="BOUNDARY" />
    <!-- another geometric column exists, named CENTER, for the
    city center point -->

    <MapGen scaleMin="1000000">
      <Field name="Boundary">CENTER</Field/>
      <Where>ADMIN_LEVEL>4</Where>
    </MapGen>

    <MapGen scaleMax="1000000" scaleMin="100000">
      <Field name="Boundary"/>
      <Where>ADMIN_LEVEL=4</Where>
    </MapGen>

    <MapGen scaleMax="100000">
      <Field name="Boundary"/>
      <Where>ADMIN_LEVEL<4</Where>
    </MapGen>

  </SQL>
</Mapping>
[...]
```

Scale Dependent Table

Several data tables can be set up, each having the same structure but a different set of data and each to be invoked at a different scale range. The configuration is explained in section 5.10 below.

MapGen Tags and Attributes

The following DTD portion defines the MapGen elements and attributes. For more information about the mapping tags, refer to the appendix on feature type mappings.

```
<!ELEMENT MapGen (Field?, NoIdGeneration?, NoQueryGeneration?,
Where?, Last?)
<!ATTLIST MapGen
scaleMin NMTOKEN #IMPLIED
scaleMax NMTOKEN #IMPLIED >
<!ELEMENT Field (#PCDATA) >
<!ATTLIST Field name CDATA #REQUIRED>
<!ELEMENT NoIdGeneration EMPTY >
<!ELEMENT NoQueryGeneration EMPTY >
<!ELEMENT Where (#PCDATA) >
<!ELEMENT Last (#PCDATA) >
```

The <MapGen> Tag

MapGen is defined in the WFS feature mapping file by one or more <MapGen> tags that are embedded in the <SQL> mapping tag for each feature type.

```
<Mapping>
...
<SQL name="wfs:myFeature">
  <MapGen>
    ...
  </MapGen>
</SQL>
...
</Mapping>
```

The existing mapping configuration tags are still needed to provide default behavior and other types of requests such as GetCapabilities, GetFeatureInfo, DescribeFeatureType, and GetFeature. Remember that the MapGen tag only applies to GetMap requests.

Feature Properties (Re)definition

Each <MapGen>...</MapGen> block must mention the properties that will be used to portray features, except if the <NoQueryGeneration> element is used. This is done with a set of <Field> tags:

```
<Field name="name1">name2</Field>
```

"Name1" is the name of the Feature's property and "name2" is the name of the field to actually use in the underlying data source. "name2" only has to be set if the "name1" property has to be replaced with something else.

The properties can also be redefined with an SQL clause. For example, if the property has a numerical value, apply a formula and use it in conjunction with a "group by" operator.

```
<Field name="DATE">MAX(ESAF_DATE)</Field>
```



If the <Field> tag is not provided, properties (including geometry) will NOT be extracted thus making the data impossible to render.

A <NoIdGeneration> tag might be used so that no check to compute the feature identifier is made. Refer to the following example that has been configured for optimal GetMap response time.

```
[...]
<Mapping>
  <SQL name="wfs:POPULATION">
    <Primary name="ID" nameSQL="ID" />
    <Element name="YEAR_" nameSQL="YEAR_" />
    <Element name="COUNTRY" nameSQL="COUNTRY" />
    <Element name="NAME1" nameSQL="NAME1" />
    <Element name="NAME2" nameSQL="NAME2" />
    <Element name="URBAN_RURAL" nameSQL="URBAN_RURAL" />
    <Element name="POPULATION1" nameSQL="POPULATION1" />
    <Element name="MALE1" nameSQL="MALE1" />
    <Element name="FEMALE1" nameSQL="FEMALE1" />
    <Element name="HOUSEHOLDS1" nameSQL="HOUSEHOLDS1" />
    <Element name="LEVEL2_NAME" nameSQL="LEVEL2_NAME" />
    <Element name="LEVEL3_NAME" nameSQL="LEVEL3_NAME" />
    <Element name="LEVEL4_NAME" nameSQL="LEVEL4_NAME" />
    [...]
  <Element name="GEOM" nameSQL="GEOM" />

  <MapGen>
    <Field name="GEOM"/>
    <NoIdGeneration/>
  </MapGen>

</SQL>

<Info name="wfs:POPULATION">
  <SRS>EPSG:4326</SRS>
  <BoundingBox SRS="EPSG:4326" minx="-180." miny="-90."
maxx="180." maxy="90." />
</Info>

</Mapping>
[...]
```

scaleMin and scaleMax

In order to emulate the last two use cases presented, the scale interval attributes in the MapGen tag need to be set. Setting this interval by mentioning one or more MapGen sections at one per scale interval allows varying the display and filtering in accordance to the scale value of the request.

```
<MapGen scaleMin="1000" scaleMax="2000">
[... ]
</MapGen>
```

In this example, the mapping configuration is such that MapGen will be used between scales 1:1000 and 1:2000. For all other scales, classical mapping that maps all the columns in an <Element> tag is used.



The scale taken into consideration is the "standard scale" as defined in several OGC specifications. This scale is computed with the assumption that the size of a pixel on the final rendering device is 0.28mm x 0.28mm. The value of the standard scale used when generating a map can be viewed by appending the parameter `NEEDSTAT=TRUE` to the `GetMap` query.

Filter - The "Where" Tag

An optional <Where> ... </Where> block can be added to previous WHERE clauses in a query sent to the data source. This is particularly useful to define a filter on one or more fields as described in the example below.

The syntax of the WHERE clause is simply:

```
<Where>clause</Where>
```

where the clause is a string which must be valid for the data source. Beware that the quote character could need encoding as "'" (e.g.:

```
<Where>DSG IN
(&#39;PPL&#39;, &#39;PPLC&#39;, &#39;PPLA&#39;)</Where>).
```

The following example combines scale ranges and filters.

```
[... ]
<Mapping>
  <SQL name="wfs:POPULATION">
    <Primary name="ID" nameSQL="ID" />
    [...]
  <Element name="GEOM" nameSQL="GEOM" />

  <MapGen scaleMin="50000000">
```

```

    <Field name="POPULATION1"/>
    <Field name="GEOM"/>
    <Where>POPULATION1>100000</Where>
  </MapGen>

  <MapGen scaleMax="50000000" scaleMin="10000000">
    <Field name="POPULATION1"/>
    <Field name="GEOM"/>
    <Where>POPULATION1>10000</Where>
  </MapGen>

  <MapGen scaleMax="10000000">
    <Field name="POPULATION1"/>
    <Field name="GEOM"/>
    <Where>POPULATION1>1000</Where>
  </MapGen>

</SQL>

<Info name="wfs:POPULATION">
  <SRS>EPSG:4326</SRS>
  <BoundingBox SRS="EPSG:4326" minx="-180." miny="-90."
maxx="180." maxy="90." />
</Info>
<Mapping>
[...]
```

In the example, cities with more than 100 000 inhabitants have been extracted from the data source and will be portrayed at or above a scale of 1:50 000 000.

The "Last" Tag

This optional tag permits a grouping or sorting clause to be appended onto an SQL statement.

```
<Last>clause</Last>
```

Below is an example that uses the <Last> tag to redefine the property and scale range. The features in the example are points and the underlying database is Oracle Spatial.

Example of MapGen with a <Last> Tag

```

<Mapping>
  <SQL name="wfs:ESA_FIRE">

    <Primary name="ESAF_ID" type="xsd:integer" />
    <Element name="DATE" nameSQL="ESAF_DATE" />
    <Element name="HOUR" nameSQL="ESAF_HOUR" />
    <Element name="NDVI" nameSQL="ESAF_NDVI" />
    <Element name="STATION" nameSQL="ESAF_STATION" />
    <Element name="NUMERO" nameSQL="ESAF_ID" />
    <Element name="LAT" nameSQL="ESAF_LAT" />
    <Element name="LONG" nameSQL="ESAF_LONG" />

  </SQL>
</Mapping>
```



```

<Element name="Geometry" nameSQL="GEOM" />

<Table nameSQL="ESA_FIRE" alias="EF" />

<MapGen scaleMin="100000001">
  <Field name="DATE">MAX(ESAF_DATE)</Field>
  <Field name="HOUR">MAX(ESAF_HOUR)</Field>
  <Field name="STATION">MAX(ESAF_STATION)</Field>
  <Field name="NUMERO">count(*)</Field>
  <Field
name="Geometry">MDSYS.SDO_GEOMETRY(2001,NULL,MDSYS.SDO_POINT_TY
PE(
round(EF.GEOM.SDO_POINT.X,0),round(EF.GEOM.SDO_POINT.Y,0),NULL)
,NULL,NULL)</Field>
  <Last>group by round(EF.GEOM.SDO_POINT.Y, 0),
round(EF.GEOM.SDO_POINT.X, 0)</Last>
  <NoIdGeneration/>
</MapGen>

</SQL>

<Info name="wfs:ESA_FIRE">
  <SRS>EPSG:4326</SRS>
  <BoundingBox SRS="EPSG:4326" minx="-180." miny="-90."
maxx="180." maxy="90." />
  <Dimension name="TIME" property="DATE"?</Dimension>
</Info>
</Mapping>

```

To clarify this example:

- The MapGen mapping is applied above the scale of 1:100 000 000.
- The <Last> tag implements the following SQL statement: GROUP BY ROUND(EF.GEOM.SDO_POINT.Y, 0), ROUND(EF.GEOM.SDO_POINT.X, 0). This means that the coordinates are rounded to an integer and then a GROUP BY is done. Use of this tag completely changes the nature of the feature collection meaning that if one square degree contains several features, only one feature is outputted.
- The <NoIdGeneration> tag must be included since the "GROUP BY" tag loses the primary key of the features when it builds a new one.
- The properties of the features have now been redefined. Since a new feature will result from the use of the "GROUP BY" tag, SQL operations such as MAX, AVERAGE, COUNT(*) can be performed. Also, new geometries can be created through the Oracle syntax:

```

MDSYS.SDO_GEOMETRY(2001,NULL,MDSYS.SDO_POINT_TYPE(
round(EF.GEOM.SDO_POINT.X,0),round(EF.GEOM.SDO_POINT.Y,0),NULL)
,NULL,NULL) .

```

Warning: MapGen and Portrayal Rules

Each time a GetMap query is received by the ERDAS APOLLO WFS servlet (and if the requested map scale is in the bounds defined in one of the MapGen tag), the MapGen configuration is used to build the data extraction request. The features extracted are then portrayed by application of portrayal styles and rules. Please ensure that the MapGen configuration outputs the properties needed by the portrayal as the styles and rules applied to the features properties that have been extracted.

Scale Dependent Table

Set up several data tables each having the same structure but a different set of data and each to be invoked at a different scale range. The configuration consists of defining <TableScale> tags in the WFS mapping file, under the <Table> element. Each TableScale will mention a scale range (scaleMin and scaleMax attributes), the table name (nameSQL attribute), and possibly the user and alias attributes with a purpose similar to the corresponding elements of the <Table> element.

For GetMap and GetFeatureInfo requests from which the Scale value can be deduced, the appropriate table will automatically be chosen.

For GetFeature requests, the server currently does not analyze the Filter to determine the scale. Add a parameter:

- In HTTP-Get, add the STDSCALE parameter to the request;
- In HTTP-POST, add the stdScale attribute to the <GetFeature> element.

Sample Mapping code with Table scales:

```
<Table nameSQL="COUNTRY" >
  <TableScale scaleMin="0" scaleMax="50000" nameSQL="COUNTRY_50K"
  />
  <TableScale scaleMin="50000" scaleMax="1000000"
  nameSQL="COUNTRY_1M" />
</Table>
```



The "scale" mentioned in this section refer to the "Standard Scale" as defined by OpenGIS in the WMS 1.1.1 specification.

Data filtering

A WFS allows a set of data filtering possibilities through the use of the OGC Filter Encoding specification which is implemented in ERDAS APOLLO servlets.

However, additional filtering on the map services may be necessary. This section explains the various ways of building such filters and how to use filters in the GetMap requests.

Solution 1: Use ERDAS APOLLO Server specific parameter "FILTER" that offers the complete power of the OGC Filter Encoding specification in the GetMap requests. This only applies to WMS services on top of vector data, i.e., ERDAS APOLLO WFS servlet.

Solution 2: Configure the WFS so that one or more "Dimension" tags appear in the WMS capabilities complying to OGC WMS 1.1.1 specification. Then the Dimension name can be added to the GetMap requests for additional filtering.

Advanced Security

Several authentication mechanisms can be set up to request the client application to authenticate when querying a J2EE servlet engine or application server. Generally, the application servers allow three authentication mechanisms, BASIC, DIGEST and PKI, as well as the set up of a secure channel. In this chapter, the sample code is based on the BASIC mechanism, thus enabling authentication and authorization. In order to ensure integrity, non-repudiation and confidentiality, use SSL and PKI.

Depending on the expected security granularity, several modes are possible:

- A declarative security based on the J2EE mechanism gives coarse-grained security
- A basic ERDAS APOLLO security at provider level
- A fine-grained security can be done using ERDAS APOLLO Server specific components (like the JAAS interfaces of Java2)
- Some security at the data source level, like that provided by Oracle or a proxied WMS.

Each of those authentication modes are described in the following subsections of this chapter.



The browser-based clients rely on a login popup to allow user authentication. If the request is sent by an application (API, Desktop editor, etc.) instead of a web browser, the popup won't display. This problem has been solved in the ERDAS client GUIs by allowing a prefix to be added to the URL of the user name and

*password to produce a URL like
http://user:password@localhost:8080/erdas-
apollo/map/ATLANTA_IMGIDX . The GUI removes the 'user name
and password' prefix before calling the service, but uses it to
negotiate with the service.*

Coarse-Grained Security

The J2EE-based declarative security is derived from the set up of an authentication "Realm" containing the definition of users and groups. Several options are possible, depending on the type of application server used:

- Description in an XML file
- JDBC connection to a database
- LDAP directory



The role names that are associated with users should be written in lowercase. Mixed case or uppercase can cause association of the role with a user to fail.

Declarative Security in Apache Tomcat

For Apache Tomcat, the configuration of a Realm using LDAP is done in a configuration file named conf/server.xml . The configuration may look like:

```
<Realm className="org.apache.catalina.realm.JNDIRealm"  
debug="99"  
    connectionName="cn=Manager,dc=mycompany,dc=com"  
    connectionPassword="secret"  
    connectionURL="ldap://ldapserver:389"  
    userPassword="userPassword"  
    userPattern="uid={0},ou=people,dc=erdas,dc=com"  
    roleBase="ou=groups,dc=mycompany,dc=com"  
    roleName="cn"  
    roleSearch="(uniqueMember={0}) "  
>
```

In that case, information about the users and their groups is centralized in an LDAP directory. Each user can belong to one or more group. Rights are assigned to groups or to users in the web.xml file through the concept of "security-role". A "role-name" is either a user or a group.

The web.xml configuration file of the web application (like erdas-apollo, ...) holds the access rights relating to this web application. The following example defines a BASIC authentication through LDAP in Apache Tomcat. One role existing in the LDAP server is activated: esp_administrator. A set of URL-patterns filter the access to each service. Please refer to Tomcat documentation (<http://jakarta.apache.org/tomcat/index.html>) or the Java Servlet specification for more details on allowed URL patterns.

```
<!-- Administration servlet -->
  <security-constraint>
    <web-resource-collection>
      <web-resource-name>Administration servlet</web-
resource-name>
      <description>Server Administration</description>
      <url-pattern>/admin/*</url-pattern>
    </web-resource-collection>
    <auth-constraint>
      <role-name>esp_administrator</role-name>
    </auth-constraint>
  </security-constraint>

  <!-- Roles and realm definition -->
  <security-role>
    <role-name>esp_administrator</role-name>
  </security-role>
  <login-config>
    <auth-method>BASIC</auth-method>
    <realm-name>Apollo Realm</realm-name>
  </login-config>
```

Note: In the "login-config" tag, set "realm-name" to "default" in order to use the settings defined in TOMCAT_HOME/conf/tomcat-users.xml instead of those from an LDAP source.

PKI and SSL Configuration

PKI is another security configuration at the servlet engine level. Note that configuring PKI security may affect the behavior of the ERDAS servlets.

At the current time, ERDAS servlets behind such a "coarse-grained" setup in the servlet engine do support parameters based on SSL or TSL and only accept one class of access. A firewall configuration to allow the SSL port is also needed.



At this time, fine-grained PKI security is not supported. This means that if the servlet engine is set up in this manner, the ERDAS servlets may not respond properly to all requests.

The SSL configuration of the application server can be authenticated in the following manners:

- Client Authentication
- Client Access Policy
- Server Authentication
- Encryption of Messages



Do not mix secure and non-secure data in the same site. Several browsers do not allow this type of behavior.

Basic ERDAS APOLLO Security

ERDAS APOLLO provides a basic authentication mechanism available at the provider level, to limit the access to the service to authenticated users. At run time, the user must be authenticated before being allowed to retrieve data. The configuration is done in the `providers.fac` using the "security" parameter. Its value can either be:

- a string, composed of a couple of blank-separated names: "name password"
- a string, composed of an encrypted couple of blank-separated names, for Digest mode authentication. This encrypted string is obtained by the tool `com.ionicssoft.security.web.AuthenticatorFactory`, packaged in `tools/ows`.
- "provider" to redirect the authentication to the underlying data server, if it supports it.

The following configuration shows a sample provider enhanced with basic provider-level security. The security value is obtained by running the encryption tool:

```
cd /home/Erdas/APOLLO/tools/ows

. ../setclasspath.sh

java -cp $IA_CLASSPATH
com.ionicssoft.security.web.AuthenticatorFactory -user scott -
password tiger -realm "Apollo Realm"

a620102876920d9c85c297ed9a9bed3c

<CREATE ID="BOSTON_ORA"
  JCLASS="com.ionicssoft.wfs.provider.oracle.OracleProvider">
```

```

<PARAM NAME="name" VALUE="BOSTON_ORA"/>
<PARAM NAME="title" VALUE="Boston on Oracle"/>
<PARAM NAME="connect"
VALUE="oracle://myhost/user+myuser/password+mypassword/SID+mysi
d"/>
<PARAM NAME="types" VALUE="boston_ora.xsd" />
<PARAM NAME="mapping" VALUE="boston_ora.xml" />
<PARAM NAME="security" VALUE="a620102876920d9c85c297ed9a9bed3c"
/>
</CREATE>

```

Fine-Grained Security

Most applications will set up coarse-grained security, but some will also require constraints more related to the GIS world. ERDAS APOLLO Server supports that functionality through resolvers, credentials, and filters. The configuration is a three-step job:

- First, define the way authentication has to be handled by registering the proper resolver and subject filler in the resolver.xml file.
- Assign a set of credentials to each user declared in the web.xml file. This assignment is done in the credential.xml file.
- Finally, inform the engine of what set of filters are to be used.



The distribution includes a set of sample configuration files in the erdas-apollo web app under WEB-INF\classes\com\ionicsoft\security\auth\resource.

To activate them for ALL providers, remove the ".wcssample" suffix for WCS-related configurations or the ".wfssample" for WFS-related files.

To activate them for a single provider, copy them to another place (for example, near the providers.fac file) and set the SECURITYFILTER and SECURITYRESOLVER parameters to the path of those files.

Setting Up the Subject Resolver

The J2EE application servers provide the authentication mechanism. To enable fine grained security, more information is needed other than just a user name and a set of roles. Fine grained credentials are required. It is the responsibility of the resolvers to ensure that all the information is available before processing the requested action.

One major function of the resolver is to populate the Subject with its credentials. The next section describes how to configure XML credentials. Other types of credentials declarations are possible (LDAP server, Oracle Label Security, SOAP header, Oracle WebGate, ...) but need specific developments which are out of the scope of this guide. The basic file to configure is the `com/erdas/security/auth/resource/resolver.xml` file, or an alternate similar file if the "securityresolver" parameter is defined at the provider level.

Example:resolver.xml Configuration File using XML Credentials

```
<?xml version="1.0" encoding="UTF-8" ?>
<configuration>
  <register JCLASS="com.ionicsoft.security.auth.TestResolver" />
  <register
JCLASS="com.ionicsoft.security.auth.XMLSubjectFiller">
  <param NAME="credentialsurl" VALUE="./credential.xml"/>
  </register>
</configuration>
```

In most cases, the `com.ionicsoft.security.auth.TestResolver` must be used as the first resolver. ERDAS APOLLO Solution Toolkit users can develop other resolvers to use custom security sources.

The `<register>` tag can have a `<param>` sub-element with a NAME and VALUE attribute. If the Subject Filler is a simple text file, the JCLASS is "com.ionicsoft.security.auth.SimpleTextfileSubjectFiller" and the `<param>` sub-element locates the file with NAME="file" and VALUE="<path to the file>". For an XML Subject Filter, the JCLASS is "com.ionicsoft.security.auth.XMLSubjectFiller", the param NAME is "credentialsurl" and the VALUE is an URL, which default value is "./credential.xml".

Note: ERDAS APOLLO Solution Toolkit users can develop other subject fillers to use custom credential sources, such as Oracle Label Security.

Adding Credentials to The Users

As soon as the user is authenticated, add some credentials. In the coarse-grain configuration part, one credential has already been defined to give the right to access the service. This right is defined in the security-constraint tag of the WEB-INF/web.xml file. If restricting access to the service, tune that value to restrict access to a given set of users.

But behind this simple, coarse grained, access control, the user may require some more specific rights. This is done in an XML resource called `com/erdas/security/auth/resource/credential.xml`, as in the example below. Note that the location of this resource can be changed through the "credentialurl" parameter in the `resolver.xml` file.

As of ERDAS APOLLO and ERDAS APOLLO Data Manager, many types of credentials are defined and supported. The credentials are configured through their Java class name (or an equivalent short name) and can be activated through a filter Java class name (or an equivalent short name). The table below lists most of these credentials and the corresponding class and short names. The "LoginCredentialMap" credential is described in the next section.

Table 66: Geographic Credentials

Credential Type	Description	Credential Class	Filter Class
Layer Access Credential	to assign a set of WMS layers and styles to a user	<code>com.ionicsoft.security.credential.LayerAccessCredential</code> (short name: Layer, in a "type" attribute)	<code>com.ionicsoft.security.auth.LayerAccessFilter</code> (short name: LayerAccess, in a "type" attribute)
Feature Type Access Credential	to assign a set of WFS feature types and operations to a user	<code>com.ionicsoft.security.credential.FeatureTypeAccessCredential</code> (short name: FeatureType)	Filter on feature type access: <code>com.ionicsoft.security.auth.FeatureTypeAccessFilter</code> (short name: FeatureTypeAccess, in a "type" attribute); Filter on operation: <code>com.ionicsoft.security.auth.OperationAccessFilter</code> (short name: OperationAccess, in a "type" attribute)
Coverage Type Access Credential	to assign a set of WCS coverage types to a user	<code>com.ionicsoft.security.credential.CoverageAccessCredential</code> (short name: Coverage)	<code>com.ionicsoft.security.auth.impl.CoverageAccessFilter</code> (short name: CoverageAccess, in a "type" attribute)

Table 66: Geographic Credentials (Continued)

Spatial Area Access Credential	to assign a set of geographic areas (bounding boxes) to a user	com.ionicsoft.security.credential.SpatialAreaAccessCredential (short name: SpatialArea)	For a WMS request: com.ionicsoft.security.auth.SpatialAreaAccessFilter (short name: SpatialAreaAccess, in a "type" attribute); For a WCS request: com.ionicsoft.security.auth.CoverageSpatialAreaAccessFilter (short name: CoverageSpatialAccess, in a "type" attribute)
Scale Range Access Credential	to assign a set of scale ranges to a user	com.ionicsoft.security.credential.ScaleRangeAccessCredential (short name: ScaleRange)	For a WMS request: com.ionicsoft.security.auth.ScaleAccessFilter (short name: ScaleAccess, in a "type" attribute); For a WCS request: com.ionicsoft.security.auth.impl.CoverageScaleAccessFilter (short name: CoverageScaleAccess, in a "type" attribute)
Size Access Credential	to limit the maximum image size allowed (attributes WIDTH and HEIGHT of a GetMap request)	com.ionicsoft.security.credential.SizeAccessCredential (short name: ImageSize)	Only applies to a WMS GetMap request: com.ionicsoft.security.auth.impl.SizeAccessFilter (short name: SizeAccess, in a "type" attribute)
BooleanCredential	to accept or reject all messages	none	To accept all messages: com.ionicsoft.security.auth.TrueFilter (short name: AlwaysTrue, in a "type" attribute); To reject all messages: com.ionicsoft.security.auth.FalseFilter (short name: AlwaysFalse, in a "type" attribute)

Example: credential.xml Configuration File

Inside a credential set, the set of credentials are checked against the active filters (see next section). If at least one succeeds, access is granted. For example, if a given role is associated with several Spatial Area Access Credentials for several zones, the Spatial Area Access filter will succeed if at least one of the zones contains the box of the query.

Layers and Styles: If the credential is given without mentioning a set of styles, use the tag `<PARAM NAME="layers">` with a comma-separated list of layer names. If the filtering has to apply at the layer and style level, each layer name should appear in a `<PARAMBLOCK>` tag, and the styles are given in a `<PARAM NAME="styles">` tag inside the `PARAMBLOCK`.

Feature Types: the credential must contain a `PARAM` tag with `NAME="featuretypes"`, and the value is a comma-separated list of feature type names. A second `PARAM` may be added, to restrict the set of operations allowed on the given feature types. The `NAME` is "operations", and the `VALUE` is a comma-separated list of operation names, or "" for all. The supported operation names are: query, insert, update, delete, lock.

Coverages: the credential must contain a `PARAM` tag with `NAME="coverages"`, and the value is a comma-separated list of coverage names.

Credential IDs: They are mandatory but currently unused. Make them unique inside a `credential.xml` file for future use.

```
<?xml version="1.0" encoding="UTF-8" ?>
<credentials>
  <credentialset subject="dm">
    <credential ID="LayerAccessCredential"
JCLASS="com.ionicsoft.security.credential.LayerAccessCredential"
">
      <PARAMBLOCK NAME="hydro">
        <PARAM NAME="styles" VALUE="default,simple"/>
      </PARAMBLOCK>
      <PARAMBLOCK NAME="highways">
        <PARAM NAME="styles" VALUE="default"/>
      </PARAMBLOCK>
    </credential>
    <credential ID="FeatureTypeAccessCredential"
JCLASS="com.ionicsoft.security.credential.FeatureTypeAccessCredential"
">
      <PARAM NAME="featuretypes" VALUE="protectedareas"/>
      <PARAM NAME="operations" VALUE="query"/>
    </credential>
    <credential ID="SpatialAreaAccessCredential"
JCLASS="com.ionicsoft.security.credential.SpatialAreaAccessCredential"
">
```

```

        <PARAM NAME="box" VALUE="SRID=26986;POLYGON((228945.0
889853.0, 233537.0 889853.0.0,
        233537.0 894397.0, 228945.0 894397.0, 228945.0
889853.0.0))"/>
    </credential>
    <credential ID="ScaleRangeAccessCredential"
JCLASS="com.ionicsoft.security.credential.ScaleRangeAccessCrede
ntial">
        <PARAM NAME="minScale" VALUE="15000"/>
        <PARAM NAME="maxScale" VALUE="20000"/>
    </credential>
    <credential id="SizeAccessCredential"
JCLASS="com.ionicsoft.security.credential.SizeAccessCredential"
>
        <PARAM NAME="maxWidth" VALUE="1000"/>
        <PARAM NAME="maxHeight" VALUE="1000"/>
    </credential>
</credentialset>
<credentialset subject="admin">
    <credential ID="LayerAccessCredential"
JCLASS="com.ionicsoft.security.credential.LayerAccessCredential
">
        <PARAM NAME="layers"
VALUE="roads,hydro,highways,land_use,protectedareas"/>
    </credential>
    <credential ID="FeatureTypeAccessCredential"
JCLASS="com.ionicsoft.security.credential.FeatureTypeAccessCred
ential">
        <PARAM NAME="featuretypes"
VALUE="roads,hydro,highways,land_use,protectedareas"/>
        <PARAM NAME="operations" VALUE="*"/>
    </credential>
</credentialset>
<credentialset subject="ld">
    <credential ID="CoverageAccessCredential"
JCLASS="com.ionicsoft.security.credential.CoverageAccessCredent
ial">
        <PARAM NAME="coverages"
VALUE="MOD09GHK.EastCoast_1_Grid_L2g_2d" />
    </credential>
    <credential ID="SpatialAreaAccessCredential"
JCLASS="com.ionicsoft.security.credential.SpatialAreaAccessCred
ential">
        <PARAM NAME="box" VALUE="SRID=4326;POLYGON((-74.197 39.6632,-
74.197 44.0858,
        -69.3613 44.0858,-69.3613 39.6632,-74.197 39.6632))"/>
    </credential>
</credentialset>
</credentials>

```

Adding Filters to The Services

The last step is to configure the service to filter on advanced security constraints. ERDAS APOLLO currently allows many different fine grained security checks, some corresponding to a given access credential defined above, and some applying unconditionally:

- For the WMS, filter on the layer name and style name using the Layer Access Filter
- For the WMS, filter on the spatial extent of the query using the Spatial Area Access Filter
- For the WMS, filter on the scale of the query using the Scale Access Filter
- For the WMS, restrict the size of the image that a GetMap can produce, using the Size Access Filter
- For the WFS, filter on the feature type name using the Feature Type Access Filter.
- For the WFS, filter on the feature type name using the Feature Type Access Filter.

For the WFS, filter on the operation name using the Operation Access Filter.

- For the WCS, filter on the coverage type name using the Coverage Access Filter.
- For the WCS, filter on the spatial extent of the query using the Coverage Spatial Area Access Filter.
- For the WCS, filter on the scale of the query using the Coverage Scale Access Filter.
- For any type of request, apply a "allow all" filter using the True Filter.
- For any type of request, apply a "reject all" filter using the False Filter.

These fine-grained security checks are configured in the file `com/erdas/security/auth/resource/filter.xml`, or an alternative file if the "securityfilter" parameter is set at the provider level. Each entry in this file configures a security check.

The following example shows a filter.xml file filtering on all types of filters. To disable one or more of those filters, remove it from that file.

Example:filter.xml Configuration File

```
<?xml version="1.0" encoding="UTF-8" ?>
<configuration>
  <role name="dm">
    <and>
```

```

    <filter ID="ScaleFilter"
JCLASS="com.ionicsoft.security.auth.ScaleAccessFilter"/>
    <filter ID="SpatialFilter"
JCLASS="com.ionicsoft.security.auth.SpatialAreaAccessFilter"/>
    <filter ID="LayerFilter"
JCLASS="com.ionicsoft.security.auth.LayerAccessFilter"/>
    <filter ID="FeatureTypeFilter"
JCLASS="com.ionicsoft.security.auth.FeatureTypeAccessFilter"/>
    <filter ID="SizeAccessFilter"
JCLASS="com.ionicsoft.security.auth.impl.SizeAccessFilter"/>
  </and>
</role>
<role name="admin">
  <or>
    <filter ID="AdminLayerFilter"
JCLASS="com.ionicsoft.security.auth.LayerAccessFilter"/>
    <filter ID="AdminFeatureTypeFilter"
JCLASS="com.ionicsoft.security.auth.FeatureTypeAccessFilter"/>
  </or>
</role>
<role name="ld">
  <and>
    <filter ID="WCSSpatialFilter"
JCLASS="com.ionicsoft.security.auth.impl.CoverageSpatialAreaAccessFilter"/>
    <filter ID="WMSSpatialFilter"
JCLASS="com.ionicsoft.security.auth.SpatialAreaAccessFilter"/>
    <filter ID="CoverageFilter"
JCLASS="com.ionicsoft.security.auth.impl.CoverageAccessFilter"/>
  >
    <filter ID="LayerFilter"
JCLASS="com.ionicsoft.security.auth.LayerAccessFilter"/>
  </and>
</role>
<role name="trusted">
  <filter ID="FullAccessFilter"
JCLASS="com.ionicsoft.security.auth.TrueFilter"/>
</role>
<!-- role with no name applies to all other roles -->
<role>
  <filter ID="NoAccessFilter"
JCLASS="com.ionicsoft.security.auth.FalseFilter"/>
</role>
</configuration>

```

The <role> element is optional in the hierarchy. If not found, the filters apply to all roles.

The <and>, <or> and <not> tags can be applied to filters and a hierarchy of those logical operators can be used.

When applicable, a <filter> tag can have a <param> sub-element with a NAME and a VALUE parameter.

Future Work

Today it is only possible to store credentials and filters in XML files. It is projected to extend this to storage into other credential sources, such as LDAP directories.

However, the credential classes can also be registered in the Application Server and then configured and used without the need for an XML file. It implies writing a custom SubjectFiller to read the credentials and pass them to the servlet.

The set of credentials and filters will extend depending on customer needs.

Security at the Data Source Level

ERDAS APOLLO providers are designed to allow publishing of data without having to design a brand new data set. Some configuration is required in order to connect to an existing data server. When security concerns arise, security enforced at the data server level is optimum. This security should filter up and cross over to the publication layer with minimal configuration.

The following situations are often encountered:

- The data source is a strong database system, like Oracle, and it supports security. In order to let the authentication information cross the provider, set the "security" parameter to the value "provider". As soon as the provider type itself implements the method to translate the authentication information (user, password, etc.) into the database proprietary language and the method to extract the authentication capabilities, there is nothing to do as everything is transparent.
- For such a strong database system, like Oracle, a more advanced security can be configured to map the login credential for the service with the login to the database. This is achieved by configuring a "LoginCredentialMap" entry in the credential.xml file, like described in the table and the sample XML code below.
- The data source is an OGC-compliant WMS or WFS. The provider is a WMS (or WFS) Proxy Provider and the security is automatically proxied with no configuration needed. However, if the WMS (or WFS) needs to be secure, the "security" parameter permits this. If it is to be opened without the user authenticating, the "user" and "password" parameters will let the proxy WMS (or WFS) authenticate against the proxied WMS (or WFS). More detail in [Provider Types](#).

- For such an OGC-compliant WMS or WFS, the security for the proxy service can be mapped to the security to the proxied service. The "LoginCredentialMap" configuration described below applies, as it lets configure the user/password at the proxied service level based on the actually encoded user/password for each service URL and type.

Table 67: Login Credential Map

Credential Type	Parameters	Description
Login Credential Map	This credential type defines login credential	Credential Class: com.ionicsoft.security.credential.LoginCredentialMap (short name is "Login", in a "type" attribute)
	defaultuser	the user used for the default login credential (optional; if not set, the initial connection user is used)
	defaultpassword	the password associated to the default user (optional)
		the rest of the parameters can be grouped into a PARAMBLOCK so that they can be repeated
	user	the user name
	password	the user password
	url	the url for which the user/password is provided. Can be any kind of url as an HTTP one or a simple name. If the login relates to a connection to a database, the url value must correspond to the "credentialname" parameter for the provider.
	service	If the login relates to a connection to a database, the service value must be "database". If the URL is for an OGC service, this parameter will have one of the supported OGC service types ("WMS", "WFS", "WCS", ...)(optional parameter)
	action	If the database connection switch can be achieved through a proxy session, this parameter can be used to activate it. The only supported value is "proxysession".


```

<?xml version="1.0" encoding="UTF-8" ?>
<credentials>
  <credentialset subject="dm">
    <credential ID="LoginCredentialMap"
JCLASS="com.ionicsoft.security.credential.LoginCredentialMap">
      <PARAM NAME="defaultuser" VALUE="scott"/>
      <PARAM NAME="defaultpassword" VALUE="tiger"/>
      <PARAMBLOCK>
        <PARAM NAME="user" VALUE="boston"/>
        <PARAM NAME="password" VALUE="boston"/>
        <PARAM NAME="url" VALUE="BOSTON_ORA"/>
        <PARAM NAME="service" VALUE="database"/>
      </PARAMBLOCK>
      <PARAMBLOCK>
        <PARAM NAME="user" VALUE="usa"/>
        <PARAM NAME="password" VALUE="usa"/>
        <PARAM NAME="url" VALUE="USA_ORA"/>
        <PARAM NAME="service" VALUE="database"/>
      </PARAMBLOCK>
    </credential>
  </credentialset>
  <credentialset subject="bs">
    <credential type="Login">
      <PARAMBLOCK>
        <PARAM NAME="user" VALUE="guest" />
        <PARAM NAME="password" VALUE="anonymous" />
        <PARAM NAME="service" VALUE="WFS" />
        <PARAM NAME="url"
VALUE="http://remotehost/servlet/wfs/MyService" />
      </PARAMBLOCK>
    </credential>
  </credentialset>
</credentials>

```

Login Credential Map Example

This section provides a step-by-step sequence of operations in order to reach a configuration enabling fine-grain security with two types of credentials: FeatureType filtering, a Oracle Database connection mapping. It is running under Tomcat 5 with a ERDAS APOLLO 9.3 release. Each step benefits a small explanation and a piece of configuration code. The example is intended to be as simple as possible to allow a rapidly working example.

The environment is a BOSTON_ORA provider on top of a BOSTON Oracle schema. The BOSTON user is granted access to the schema, whereas the ATLANTA one is not.

1. In the TOMCAT_HOME/conf/tomcat-users.xml, define a couple of users and roles. Let's name them MASS (for Massachusetts) and GEO (for Georgia). The updated file could look like this:

```

<tomcat-users>
  <role rolename="tomcat"/>
  <role rolename="role1"/>
  <role rolename="manager"/>

```

```

        <role rolename="admin"/>
        <role rolename="mass"/>
        <role rolename="geo"/>
        <user username="tomcat" password="tomcat"
roles="tomcat"/>
        <user username="role1" password="tomcat" roles="role1"/>
        <user username="both" password="tomcat"
roles="tomcat,role1"/>
        <user username="admin" password="admin"
roles="admin,manager"/>
        <user username="mass" password="mass" roles="mass"/>
        <user username="geo" password="geo" roles="geo"/>
    </tomcat-users>

```

2. In the web.xml file of your web application, typically under TOMCAT_HOME/erdas-apollo/WEB-INF/web.xml, add three blocks of information at the end of the file, after the <welcome-file-list> block. You have to add a <security-constraint> block, a <login-config> block and a <security-role> block. The end of the file could look like this:

```

...
<welcome-file-list>
  <welcome-file>index.html</welcome-file>
</welcome-file-list>

<security-constraint>
  <web-resource-collection>
    <web-resource-name>Vector</web-resource-name>
    <url-pattern>/vector/*</url-pattern>
  </web-resource-collection>
  <auth-constraint>
    <role-name>mass</role-name>
    <role-name>geo</role-name>
  </auth-constraint>
</security-constraint>

<login-config>
  <auth-method>BASIC</auth-method>
  <realm-name>default</realm-name>
</login-config>

<security-role>
  <description>for Boston</description>
  <role-name>mass</role-name>
</security-role>
<security-role>
  <role-name>geo</role-name>
</security-role>

```

The <security-constraint> block associates a set of roles to a pattern of services. The <login-config> tells Tomcat where and how to authenticate the users who are logged in. Here BASIC authentication is applied, and the default realm, i.e. the tomcat-users.xml file, is the place where the roles are defined. The <security-role> blocks declare the active roles.

3. Activation of fine-grain security: it is done by renaming the three *.xml.sample files located in WEB-INF/classes/com/ionicsoft/security/auth to remove the ".sample" suffix. The newly named files have the default names sought by the servlet. Note that those files can also be relocated, by setting the securityfilter and securityresolver parameters for your provider.
4. The factory content of the resolver.xml file does not need any change, as it addresses the credential.xml file.
5. In the credential.xml, add a couple of credential sets, one for "mass" and one for "geo". For "mass", a FeatureType access credential is defined, and on "geo" a LoginMapCredential is defined to force a connection with the Oracle user "Atlanta". The credential.xml configuration looks like this:

```
<credentialset subject="mass">
  <credential ID="FeatureTypeAccessCredential"
  JCLASS="com.ionicsoft.security.credential.FeatureTypeAccessCred
  ential">
    <PARAM NAME="featuretypes"
  VALUE="protectedareas,place_names,hydro"/>
  </credential>
</credentialset>

<credentialset subject="geo">
  <credential ID="LoginCredentialMap" type="Login">
    <PARAMBLOCK>
      <PARAM NAME="user" VALUE="atlanta"/>
      <PARAM NAME="password" VALUE="atlanta"/>
      <PARAM NAME="url" VALUE="BOSTON_CREDENTIAL"/>
      <PARAM NAME="service" VALUE="database"/>
    </PARAMBLOCK>
  </credential>
</credentialset>
```

Note 1: Beware that only one credential set per role is defined in the credential.xml. If there are more than one, each will be checked until one succeeds.

Note 2: As soon as the attempt to connect to the database through a different user succeeds, the default schema is no more the original one. The new user will fail to address the tables if their schema is not given. To avoid this, add a "user" attribute to the "Table" elements in the mapping file, so that each request to one of those tables will prefix them with the schema name. The change can look like this:

```
...
<SQL name="wfs:roads">
  <Table nameSQL="ROADS" user="BOSTON_ORA"/>
  <Primary name="MRD_ID" type="xsd:float" fid="true"/>
  <Element name="wfs:FNODE_" nameSQL="FNODE_" />
  <Element name="wfs:TNODE_" nameSQL="TNODE_" />
  <Element name="wfs:LPOLY_" nameSQL="LPOLY_" />
...
```

Note 3: The LoginCredentialMap configuration makes the assumption that no particular configuration was achieved on Oracle side. In some cases, Oracle allows to define proxies and in those cases the connection switch can be achieved according to this proxy setting.

6. The providers.fac entry for this provider does not need to be modified in order to activate fine-grain security: the existence of a resolver.xml is sufficient for that security to be checked. However, the LoginCredentialMap credential has to be linked with the provider. It is achieved through the "url" tag in the credential.xml file, which should match the "credentialname" in the providers.fac. The updated provider configuration could look like this:

```

      <CREATE ID="BOSTON_ORA"
JCLASS="com.ionicsoft.wfs.provider.oracle.OracleProvider">
      <PARAM NAME="OWSINFOURL" VALUE="./wfs_md.xml"/>
      <PARAM NAME="TITLE" VALUE="Boston on Oracle"/>
      <PARAM NAME="TYPES" VALUE="boston_ora.xsd"/>
      <PARAM NAME="MAPPING" VALUE="boston_ora.xml"/>
      <PARAM NAME="CONNECT"
VALUE="oracle://myhost:1521/sid+orcl/user+boston/password+boston/defaultRowPrefetch+10"/>
      <PARAM NAME="CREDENTIALNAME" VALUE="BOSTON_CREDENTIAL"/>
      <!-- uncomment to relocate the files in the same directory
as the providers.fac
      <PARAM NAME="SECURITYRESOLVER" VALUE="myresolver.xml"/>
      <PARAM NAME="SECURITYFILTER" VALUE="myfilter.xml"/>
      -->
      </CREATE>

```

7. Finally, the last file to adapt is filter.xml, to activate the FeatureType credential on "mass" and to activate the checks on "geo". Its update can look like this:

```

        <role name="mass">
            <filter ID="FeatureTypeFilter"
JCLASS="com.ionicsoft.security.auth.FeatureTypeAccessFilter"/>
        </role>

        <role name="geo">
        </role>

```

8. The first test to run is to execute from your browser a GetFeature on the place_names table and log as mass/mass. The request can look like this:

```

http://localhost:8080/erdas-
apollo/vector/BOSTON_ORA?VERSION=1.0.0&REQUEST=GetFeature&SERVI
CE=WFS&TYPENAME=place_names&MAXFEATURES=20

```

Upon success, a GML collection displays. If it fails, the displayed log message and the Tomcat log file (under TOMCAT_HOME/logs) gives more detail on the reason of the failure.

9. After closing and restarting your browser, or alternatively after clearing the HTTP authentication cache on the client side, the same request can be executed while logging in as geo/geo. If the Oracle Atlanta user is not granted access to the Boston schema, a security denial message is responded.

Oracle Proxy Session Example

Mapping an authenticated user with an Oracle user used to establish the connection can be achieved with the LoginCredentialMap configuration described above. When the provider is instantiated, the connection is established with the connection string (including user and password) defined in the providers.fac . As soon as a user sends a request to the service for authentication, the LoginCredentialMap definition found in the credential.xml file is used to achieve the mapping between this user and the Oracle user name to use to change the connection.

At that moment, the service is opening a new connection to the Oracle database using this new Oracle user.

Another use case is to keep the initial connection and just change the session so that the user connecting the database is just switched. It lets the user take advantage of the proxy session mechanism supported by Oracle 10g. The step-by-step configuration of such a use case is described in this section, but if you are already familiar with the Oracle proxy session mechanism and with ERDAS fine-grain security, you just need to add an "action" parameter to the LoginCredentialMap block to tell the service that the change in the Oracle connection is through that proxy-session mechanism.

Warning: Due to limitations to older versions of Oracle database, an Oracle 10g Release 2 is the minimum needed for the proxy session use case to run. With older versions, the proxy user will not be authorized to connect to the database.

1. A couple of Oracle users need to be defined, one with minimal rights for the initial service-to-database connection, and one with more privileges but being restricted to connect to the database through the other user's grants. If those users are named `bostonmini` and `bostonadmin`, the sequence of SQL statements can be:

```
create user bostonmini identified by bostonmini;
grant connect to bostonmini;
create user bostonadmin identified by bostonadmin;
grant create session to bostonadmin;
grant select, insert, update, delete on
boston_ora.place_names to bostonadmin;
alter user bostonadmin grant connect through bostonmini;
```

2. In the `providers.fac`, the connection string should use the "bostonmini" user so that the connection can be established to the database with no more than the ability to create that connection. Additionally, a "credentialname" parameter must be set to achieve the link with the credential. The value can be "BOSTON_CRED". The configuration in the `providers.fac` could look like this:

```
<CREATE ID="BOSTON_ORA"
JCLASS="com.ionicsoft.wfs.provider.oracle.OracleProvider">
  <PARAM NAME="OWSINFOURL" VALUE="./wfs_md.xml"/>
  <PARAM NAME="TITLE" VALUE="Boston on Oracle"/>
  <PARAM NAME="TYPES" VALUE="boston_ora.xsd"/>
  <PARAM NAME="MAPPING" VALUE="boston_ora.xml"/>
  <!-- The initial user is replaced with the proxying one
  <PARAM NAME="CONNECT"
VALUE="oracle://myhost:1521/sid+orcl/user+boston_ora/password+b
oston_ora"/>
  -->
  <PARAM NAME="CONNECT"
VALUE="oracle://myhost:1521/sid+orcl/user+bostonmini/password+b
ostonmini"/>
  <PARAM NAME="CREDENTIALNAME" VALUE="BOSTON_CRED"/>
</CREATE>
```

3. In the `credential.xml`, the `LoginCredentialMap` mentions the "bostonadmin" user, sets the "url" parameter to "BOSTON_CRED" to reference the provider, and sets the "action" parameter to "proxysession" so that the service does a session switch. The code in the `credential.xml` could look like this:

```
...
<credential ID="LoginCredentialMap" type="Login">
  <PARAMBLOCK>
    <PARAM NAME="user" VALUE="bostonadmin"/>
```

```

        <!-- NO password is needed as the connection is achieved
through bostonmini
        <PARAM NAME="password" VALUE="bostonadmin"/>
        -->
        <PARAM NAME="url" VALUE="BOSTON_CRED"/>
        <PARAM NAME="service" VALUE="database"/>
        <PARAM NAME="action" VALUE="proxysession"/>
    </PARAMBLOCK>
    <credential>
    ...

```

Masking

Fine-grained security allows to accept or reject WMS queries depending on matching geospatial criteria. But it is only binary: allow or deny. And when the restriction is by Bounding Box, that limitation is sometimes too strong: some users would still like to get a map back, with the restricted box rubbed out.

Moreover, a BBOX is a rectangle. In some situations, a more flexible geometry like a polygon is expected.

This chapter describes how a masking mechanism can be applied to complement or to replace the fine-grain security by BBOX.

Setting up a Masker Service

Before masking can be activated for a vector or coverage service, the masking geometries and roles have to be configured. They can be either stored in a shapefile whose path is mentioned in the masked service configuration, or in a WFS of any kind (GML, Oracle, ...) as soon as the proper types are defined.

For masking in a shapefile, the masking configuration consists of a single line that will look like:

```

If the definition is in the providers.fac:
<PARAM NAME="featureServer"
VALUE="wfsf:shape:/home/Erdas/APOLLO/data/erdas-
apollo/shapes/boston;srs=26986"/>

```

```

or if it is in a mapping file:
<Server>wfsf:shape:/home/Erdas/APOLLO/data/erdas-
apollo/shapes/boston;srs=26986</Server>

```

Masking in a WFS is possible as soon as the masker WFS is properly configured. The configuration examples below are provided in the distribution, under C:/Erdas/APOLLO/data/erdas-apollo/security/masking.

The source of masking polygons has to be defined, configured, and each polygon should be associated to a role. For vector data masking, the source has to be a WFS exposing a feature type that contains a geometry property, a masked role property and optionally a masking mode property. The schema could look like this (GML 2 syntax):

```
<xsd:element name="Mask" substitutionGroup="gml:_Feature"
type="wfs:Mask"/>
<xsd:complexType name="Mask">
  <xsd:complexContent>
    <xsd:extension base="gml:AbstractFeatureType">
      <xsd:sequence>
        <xsd:element name="GEOMETRY" minOccurs="0"
type="gml:PolygonPropertyType"/>
        <xsd:element name="MaskingRole" minOccurs="0"
type="xsd:string"/>
        <xsd:element name="MaskingMode" minOccurs="0"
type="xsd:string"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
```

Based on this schema, the simplest way is to set up a WFS over a GML file. That GML file is easily produced by a GetFeature request on a WFS, if the polygons already exist. Post-processing of the GML document to update the set of properties and fill the MaskingRole and MaskingMode values. If the masking polygons do not exist yet, the fast path is to configure a WFS-T and use any feature creation tool to draw and save polygons. The mapping file of such a WFS-T over Oracle could look like this:

```
<xsd:schema
  xmlns:gml="http://www.opengis.net/gml"
  xmlns:wfs="http://www.erdas.com/wfs"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xlink="http://www.w3.org/1999/xlink"
  targetNamespace="http://www.erdas.com/wfs"
  elementFormDefault="qualified">
  <wfs:Mapping>
    <SQL name="wfs:Mask">
      <Table nameSQL="MASK"/>
      <Primary name="MASK_ID" type="xsd:integer" fid="generated" />
      <Element name="wfs:MaskingRole" nameSQL="MASKINGROLE"/>
      <Element name="wfs:MaskingMode" nameSQL="MASKINGMODE"/>
      <Element name="wfs:GEOMETRY" nameSQL="GEOMETRY"/>
    </SQL>
  </wfs:Mapping>
  <Info name="wfs:Mask">
    <Title>Boston Mask</Title>
    <Abstract>Boston Mask information</Abstract>
    <Operations>*</Operations>
    <SRS>EPSG:26986</SRS>
```



```

    <BoundingBox SRS="EPSG:26986" minx="227317.3811199999"
miny="889948.2662399999" maxx="238669.28896000012"
maxy="901300.1740800001"/>
    <!--
    <Metadata/>
    <Legend/>
    -->
</Info>
<Export generation="exportOnly">
  <Add name="wfs:Mask"/>
</Export>
</xsd:schema>

```

To create the table and indexes, the following statements can be used:

```

CREATE TABLE MASK
(
MASK_ID INTEGER          NOT NULL,
GEOMETRY MDSYS.SDO_GEOMETRY,
MASKINGROLE VARCHAR2(30),
MASKINGMODE VARCHAR2(30)
);

INSERT INTO USER_SDO_GEOM_METADATA VALUES ('MASK', 'GEOMETRY',
MDSYS.SDO_DIM_ARRAY(MDSYS.SDO_DIM_ELEMENT('X', 232000.,
238000., 0.005),
MDSYS.SDO_DIM_ELEMENT('Y', 889000.0, 895000.0, 0.0005)),41051);

CREATE INDEX I_MASK_GEOM ON MASK(GEOMETRY) INDEXTYPE IS
MDSYS.SPATIAL_INDEX;

```

Masking of Vector Data

Masking of vector data can be configured in a few simple steps.

1. The service (WMS over WFS) to be masked has to be secure. The Coarse-Grain Security section above fully applies.
2. The source of masking polygons has to be configured, as described in the previous section, in a shapefile or in a WFS.
3. The service to benefit from the mask has to be updated to reference the masking service. This is done in its mapping file, by the addition of a <Masking> tag. It will look like this:

```

<Masking>
  <Server>BOSTON_ORA_MASK</Server>
  <!-- it could also be the URL of a WFS prefixed with wfs:
  <Server>wfs:http://myhost:8080/erdas-
apollo_s/vector/BOSTON_MASKER</Server>
  or a Shapefile path prefixed with wfsf:shape: and
  optionally suffixed with the SRS if not 4326

```

```

        <Server>wfsf:shape:/home/Erdas/ApolloServer/data/erdas-
apollo/shapes/boston;srs=26986</Server>
        -->
        <Type>Mask</Type>
        <!-- The default is the first feature type exposed by the
WFS -->
        <Geometry indexed="true">GEOMETRY</Geometry>
        <!-- default geometry name is "geometry" -->
        <Role>MaskingRole</Role>
        <Mode default="TRANSPARENT">MaskingMode</Mode>
        <!-- default Mode property name is "mode".
        If set to "unused", then the mode property is ignored and
the default value is always used -->
        <!-- default can be one of "TRANSPARENT" or "OPPOSITE"
(also existing: BACKGROUND, OPPOSITE_BACKGROUND) -->
        <Color>TRANSPARENT</Color>
        <!-- it can also be a real color, such as "red" or
"0xFFFFFFFF" -->
        </Masking>

```

4. The last step is to tell the masked WFS that security information has to be passed to the underlying provider. This is done by setting the SECURITYRESOLVER parameter to "container". The entry in the providers.fac is:

```
<PARAM NAME="SECURITYRESOLVER" VALUE="container" />
```



Even though the masked service exposes both WMS and WFS interfaces, only the WMS requests (GetMap, GetFeatureInfo) requests will benefit from the masking configuration. No WFS-type request (GetFeature, Transaction) will filter the data based on the masking configuration.

Masking of Coverage Data

Masking of coverage data defined in a WCS is configured in a way similar to vector data, except that it is done in the WCS providers.fac instead of in a mapping file. The masking parameters are slightly different, but have the same semantic.

1. The service (WMS over WCS) to be masked has to be secure. The Coarse-Grain Security section above fully applies.
2. The source of masking polygons has to be configured, like described in the first section above, in a Shapefile or in a WFS.
3. The service to benefit from the mask has to be updated to reference the masking service. This is done in the providers.fac, by filling the "Masking Info" compound property. The resulting tags will look like this:

```
<PARAMBLOCK NAME="masking">
```

```

        <PARAM NAME="featureServer" VALUE="BOSTON_ORA_MASK"/>
        <!--
        the id alone implies that the provider is defined in the
        same providers.fac file as the WCS.
        This is allowed! But that service will not be accessible
        via its URL, only from the masked WCS.
        it could also be the URL of a WFS prefixed with wfs:
        "wfs:http://myhost/erdas-apollo-
demo_s/vector/BOSTON_MASKER"
        or a Shapefile path prefixed with wfsf:shape: and
        optionally suffixed with the SRS if not 4326
        "wfsf:shape:/home/Erdas/ApolloServer/data/erdas-
apollo/shapes/boston;srs=26986"
        -->
        <PARAM NAME="featureType" VALUE="Mask"/>
        <!-- The default is the first feature type exposed by
the WFS -->
        <PARAM NAME="featureProperty" VALUE="GEOMETRY"/>
        <!-- default geometry name is "geometry" -->
        <PARAM NAME="roleProperty" VALUE="MaskingRole"/>
        <!-- If not specified no role is used to filter the
geometry -->
        <PARAM NAME="modeProperty" VALUE="MaskingMode"/>
        <!-- If not specified, the defaultMode value is used.
        <PARAM NAME="defaultMode" VALUE="TRANSPARENT"/>
        <!-- default can be one of "TRANSPARENT", "OPPOSITE",
"BLURRING" (also existing: BACKGROUND, OPPOSITE_BACKGROUND) -->
        <PARAM NAME="color" VALUE="TRANSPARENT"/>
        <!-- it can also be a real color, such as "red" or
"0XFFFFFF".
        It can be a fixed value or TRANSPARENT or BACKGROUND.
        Note that BACKGROUND has currently the same effect as
TRANSPARENT.
        -->
        </PARAMBLOCK>

```

4. The last step is to tell the masked WFS that security information has to be passed to the underlying provider. This is done by setting the SECURITYRESOLVER parameter to "container". The entry in the providers.fac is:

```
<PARAM NAME="SECURITYRESOLVER" VALUE="container" />
```

Table 68: The masking parameters

FEATURESERVER	Defines the url of the remote server or the id in the same factory file. Mandatory.
FEATURETYPE	Defines the feature type containing the mask information. Optional. If not defined, the first feature type exported by the server is used.

Table 68: The masking parameters (Continued)

FEATUREPROPERTY	The name of the property containing the mask geometry. Optional. If not defined, the first geometric property is used
ROLEPROPERTY	The name of the property containing the role associated to the mask. Optional. If not defined, no role is used. If defined, the retrieved masks are those associated to the role of the current request.
MODEPROPERTY	The name of the property containing the type of mask to apply. Optional. If defined, the actual type of mask will be computed by the value of the property and the DEFAULTMODE parameter. If not defined, the DEFAULTMODE parameter will be also ignored and the type of mask will be TRANSPARENT or BLURRING (if the RESOLUTION parameter value is greater than 0).
DEFAULTMODE	Defines the default type of mask. Possible values are defined in Table 69:Types of mask . Optional. The default is TRANSPARENT.
COLOR	Defines the masking color, It can have the value "TRANSPARENT", "BACKGROUND", a color name, "rgb(x,y,z)" or an hex string. "TRANSPARENT" means use a transparent color, "BACKGROUND" means use the background color of the request if available. Optional. Default value is "TRANSPARENT".
COMPOSITE	Optional. If defined (the only useful value is "alpha"), it tells the mask process to use alpha composition when masking areas. In order to be taken into account, it implies that the masking color (COLOR) has an alpha channel and GetMap requests are made using TRANSPARENCY=TRUE and with image format supporting transparency (e.g. image/png).

Table 68: The masking parameters (Continued)

RESOLUTION	Defines the size of the pixel in meter below that blurring masking applies. If you set 10m, a image with more resolution having pixel size of 5m will contained blurred area.
------------	---

Table 69: Types of mask

BLURRING	It will blurred the geometry mask if the RESOLUTION is greater than 0, otherwise it behaves like a transparent mask.
TRANSPARENT	It will fill the geometry mask by the masking color.
OPPOSITE	It will fill the exterior of the geometry mask by the masking color.
TRANSPARENT_BACKGROUND	Mostly like TRANSPARENT but uses the request background color by default.
OPPOSITE_BACKGROUND	Mostly like OPPOSITE but uses the request background color by default.

The type of mask is determined in the following way:

If the parameter MODEPROPERTY is defined, the type of mask is given by the value of the property. If this value is null or unknown, the value of the DEFAULTMODE parameter is taken. If this parameter is not defined, TRANSPARENT is taken as type of mask.

5. If the parameter MODEPROPERTY is not defined, the type of mask is BLURRING if RESOLUTION is greater than 0, otherwise the type of mask is TRANSPARENT.



Since the coverage framework responds to WCS and WMS interfaces, masking will apply to both. But for the WCS interface (i.e. GetCoverage request), the notion of color does not really exist. In that case, defining a COLOR can lead to unexpected results. COLOR should not be applied to GetCoverage requests, except for coverage which are basically RGB images. There is no such problem with the WMS interface.

SRS Configuration Parameters

Structure

Each coordinate transform system that follows the specifications of OGC is defined in a series of XML files. The entry file is called sref.xml and is located under com.ionicsoft.sref.impl.resource. Each file contains a series of tags that need to be populated in order to build the coordinate systems database.

Inside this file, the user can change the option value and/or add other systems definition.

The main tags are defined in the table below and described in the following sections.

Table 70: SRS ConfigurationTags

Tag	Attributes	Description
SREF		The root element of the XML file
Main descendant of SREF		
STORAGE		
	TYPE	The type of storage to use. Allowed values are FILE or MEMORY
OPTION	Define one option	
	NAME	The option name
	VALUE	The option value
INCLUDE	include a file	
	NAME	The file name to include (usually a relative path)
	OPTIONAL	if yes the file is an optional one and no error is generated if it is missing
INCLUDEESRI	Include the ESRI mapping file (see structure info below)	
	NAME	The file name to include
	OPTIONAL	If yes the file is an optional one and no error is generated if it is missing
NAMESPACE		Defines some additional namespaces of ids

Table 70: SRS ConfigurationTags (Continued)

ASSOCIATE		Associates some other ids to an existing id
Main definition tags (these tags are also direct descendant of SREF)		
UNIT	Defines a unit	
MERIDIAN	Defines a meridian	
SPHEROID	Defines a spheroid	
DATUM	Defines a datum	
GEOCS	Defines a geographic system	
PROJCS	Defines a projected system	
COORDINATESYSTEM	Defines a coordinate system	Used to define the axis unit of a coordinate system. This allow to support XXXX ids.

All tags (except the root) can be mixed and freely used. The only constraint is the object definition rules cannot be broken (see below). However; it is advisable to have a main document containing only 'include' and option instructions and documents containing only object definitions.

STORAGE

The storage tag should be the first child of the SREF root tag. It defines the type of storage to use. Currently two types are available:

1. FILE: Uses a btree file as database support
2. MEMORY: Uses a memory map. This is intended for small SRS database.

OPTION

The option tag defines options (hints) about different conversions to the SRS or Data Manager.

The following options are defined:

1. useostn97: specify if the ostn97 conversion is used to go from OSGB1936 to WGS84. This conversion uses 1,000,000 bands that are stored into memory.
2. parseesriid: specify if the INCLUDEESRI instruction should really parse the esri ids file.
3. unknownDatumAsWGS84: if set (default is yes) the unknown datum are considered as equal to WGS84 during datum transformation. If not set an error will be produced during datum transformation.

NOTE: the option names are case insensitive and their usual values are 'yes' or 'no' (also case insensitive).

INCLUDE

The 'include' and 'includeesri' tags allow the inclusion of other files. They allow splitting the configuration into several files. The 'include' tag acts as if the content of the file was inline. The format of the esri file is a simple text file where each line is in the form id=name. It defines associations between EPSG id and ArcSde system name.

The current structure of the sref.xml is the following

Example:Sample sref.xml File

```
<?xml version="1.0" encoding="utf-8" ?>
<SREF>
  <!-- the storage type may be MEMORY or FILE -->
  <STORAGE TYPE="MEMORY"/>
  <!-- include the user option -->
  <INCLUDE NAME="option.xml" OPTIONAL="YES" />
  <!-- include the user option -->
  <INCLUDE NAME="useroption.xml" OPTIONAL="YES" />
  <!--include the standard definition -->
  <INCLUDE NAME="factorysref.xml" />
  <!-- include the area definition -->
  <INCLUDE NAME="arearef.xml" OPTIONAL="YES" />
  <!-- include the erdas user definition -->
  <INCLUDE NAME="ionicsref.xml" OPTIONAL="YES" />
  <!-- include the user definition-->
  <INCLUDE NAME="usersref.xml" OPTIONAL="YES" />
  <!-- include the esri id file-->
  <INCLUDEESRI NAME="esri.txt" OPTIONAL="YES" />
  <!-- include the ogc id file-->
  <INCLUDEESRI NAME="ogc.txt" OPTIONAL="YES" />
</SREF>
```

All those files are provided in the distribution except the usersref.xml. It is intended to contain user options and definitions.

Object Definition

All objects defining a coordinate system must be defined. This includes:

- Units of measure
- The meridian
- The spheroid
- The datum
- The geographic system
- The projected system

Object Sharing

- The coordinate system

The general rules are:

1. Objects must be defined before being used.
2. IDs must be unique among all definitions
3. Objects are reused and shared based on their IDs

So the following,

```
<GEOCS ID="4312" NAME="MGI" >
<UNIT ID="9108" />
</GEOCS>
```

defines a new geographic system that reuses the unit definition 9108.

If the following is used,

```
<GEOCS ID="4312" NAME="MGI" >
<UNIT ID="9108" NAME="Toto" VALUE="0.5" />
</GEOCS>
```

it will lead to the (re)definition of unit 9108. This is fine if the unit does not already exist but otherwise will redefine it for all instances.

This is used to redefine a local definition. It is useful when redefining a datum to use another Bursa-Wolf transformation variant.

This is done by using the attribute UNSHARED="YES" to the definition.

```
<GEOCS ID="4312" NAME="MGI">
<UNIT ID="9108" />
<MERIDIAN ID="8901" />
<DATUM UNSHARED="YES" ID="6312" NAME="MGI" SHIFTX="577.326"
SHIFTY="90.129" SHIFTZ="463.919" ROTX="5.137" ROTY="1.474"
ROTZ="5.297" SCALEFACTOR="2.4232">
<SPHEROID ID="7004" />
</DATUM>
</GEOCS>
```

In this example, the datum will be a local copy inside the geographic system

Unit Definition

A unit is defined as

```
<UNIT ID="9001" NAME="meter" VALUE="1.0"/>
```

Items	Description
ID	The unit id
NAME	The unit name
VALUE	The unit factor relative to the reference unit. The reference unit is the meter for metric units and the radian for angular units.

Spheroid Definition

A spheroid (geoid) is defined as

```
<SPHEROID ID="7001" NAME="Airy 1830"  
FLATTENING="299.3249646" SEMIMAJORAXIS="6377563.396"/>
```

Items	Description
ID	The spheroid id
NAME	The spheroid name
FLATTENING	The 1/f value
SEMIMAJORAXIS	The length of the semi-major axis in meters

Meridian Definition

```
<MERIDIAN ID="8913" NAME="Oslo" VALUE="10.722916666666666"/>
```

or

```
<EXTENDEDMERIDIAN ID="50000" NAME="HonkKong"  
VALUE="0.0024444444444444444" DELTA="-  
0.0015277777777777777"/>
```

Items	Description
ID	The meridian id
NAME	The meridian name
VALUE	The offset from Greenwich longitude in degrees

DELTA	The offset from Greenwich latitude in degrees (only used in EXTENDED MERIDIAN)
--------------	--

Datum Definition

A datum is defined as:

```
<DATUM ID="6124" NAME="Rikets_koordinatsystem_1990"
SHIFTX="414.1055246174"
SHIFTY="41.3265500042"
SHIFTZ="603.0582474221"
ROTX="0.8551163377"
ROTY="-2.1413174055"
ROTZ="7.0227298286"
SCALEFACTOR="0"
METHOD="CFR" >
<SPHEROID ID="7004" />
</DATUM>
```

Items	Description
ID	The datum id
NAME	The datum name
SPHEROID	The spheroid definition
METHOD	The method used to transform this datum to the WGS84 datum (Values are PV,CFR, GO or SHIFT) Default is PV which stands for Position Method Vector, Coordinate Frame Rotation, Geographic Offset and Shift.
	CFR/PV additional parameters (The values comes from the EPSG database)
SHIFTX	The X shift in meters (default is 0)
SHIFTY	The Y shift in meters (default is 0)
SHIFTZ	The Z shift in meters (default is 0)
ROTX	The X rotation (default is 0)
ROTY	The Y rotation (default is 0)
ROTZ	The Z rotation (default is 0)
SCALEFACTOR	The scale factor
	GO additional parameters
ROTX	The X offset (default is 0)
ROTY	The Y offset (default is 0)

Geographic System Definition

```
<GEOCS ID="4125" NAME="Samboja">
<UNIT ID="9108" />
<MERIDIAN ID="8901"/>
<DATUM ID="6125" />
</GEOCS>
```

Items	Description
ID	The geographic system id
NAME	The geographic system name
UNIT	The angular unit definition
MERIDIAN	The meridian definition
DATUM	The datum definition

Projected System Definition

```
<PROJCS ID="206" NAME="St Lucia 1955 / British West Indies Grid">
<UNIT ID="9001" />
<GEOCS ID="4606" />
<PROJECTION NAME="Transverse Mercator">
<PARAMETER NAME="central_meridian" VALUE="-62.0"/>
<PARAMETER NAME="false_easting" VALUE="400000.0"/>
<PARAMETER NAME="false_northing" VALUE="0.0"/>
<PARAMETER NAME="latitude_of_origin" VALUE="0.0"/>
<PARAMETER NAME="scale_factor" VALUE="0.9995"/>
</PROJECTION>
</PROJCS>
```

Items	Description
ID	The projected system id
NAME	The projected system name
UNIT	The projected unit definition
GEOCS	The geographic system definition
PROJECTION	The projection definition
VALIDITY	The area of validity of the projected system.

NOTE: The validity information is defined by four attributes: X1,Y1,X2,Y2. Currently it is automatically defined for some projections as Transverse Mercator. The Values must be set in the associated geographic system.

Other IDs

The projected and geographic system include other associated IDs by having their tags added to the system's definition.

Currently the tags OGC, ESRI, DMS and ALIAS are defined. Their ID attribute is associated to the system as another ID candidate which can only be used to get the system reference through the SRS manager. Generally, the ESRI ID is defined in another file named esri.txt and associated with the EPSG ID through the INCLUDEESRI tag in the main sref.xml file. Nevertheless it is possible to associate other IDs to each defined objects, using the namespace mechanism (see below).

Ex:

```
<PROJCS ID="206" NAME="St Lucia 1955 / British West Indies Grid">
<DMS ID="My DMS ID" />
</PROJCS>
```

Other id domains can be added later using the ASSOCIATE element

```
<ASSOCIATE ID="206">
<DMS ID="My DMS ID" />
<FIPS ID="2900" />
</ASSOCIATE>
```

The ALIAS id is implicitly associated to the "name:" protocol and allows IDs to be retrieved by their name of indirectly if they end an anchor.

Namespaces

Users can define their own namespaces of id. For example, to add the FIPS and NASA namespaces, add the block:

```
<NAMESPACE>
<DEFINE ID="FIPS" />
<DEFINE ID="NASA" />
</NAMESPACE>
```

Those declarations should occur after the STORAGE definition and obviously before those ids are used.

Projection Definition

```
<PROJECTION NAME="Transverse Mercator">
<PARAMETER NAME="central_meridian" VALUE="-62.0"/>
<PARAMETER NAME="false_easting" VALUE="400000.0"/>
<PARAMETER NAME="false_northing" VALUE="0.0"/>
<PARAMETER NAME="latitude_of_origin" VALUE="0.0"/>
<PARAMETER NAME="scale_factor" VALUE="0.9995"/>
</PROJECTION>
```

NOTE: The latitude and longitude in the projection parameters are given in the underlying geographic system unit, e.g., longitudes are relative to the prime meridian of the geographic system. If this is not the case, use the REF attribute.

Items	Description
-------	-------------

NAME	The projection name
PARAMETER	A set of projection parameters
Each parameter has the following attributes	
NAME	The parameter name
VALUE	The parameter value, alternatively a parameter can be given as deg min sec using 3 attributes instead of one
DEG	
MIN	
SEC	
REF	If present, it must have the value Greenwich to specify that the unit is relative to Greenwich instead of the prime meridian of the geographic system.
Projection	Parameters
Albers Conical	
	central_meridian
	latitude_of_origin
	standard_parallel_1
	standard_parallel_2
	false_easting
	false_northing
EquiCylindrical	
	central_meridian
	standard_parallel_1
	false_easting
	false_northing
Lambert Azimuthal Equal Area	
	central_meridian
	latitude_of_center
	azimuth
	false_easting
	false_northing

Lambert Conformal Conic	same as Albers Conical
Lambert Conformal Conic 2sp	same as Albers Conical
Lambert Conformal Conic 2sp Belgium	same as Albers Conical
Lambert Conformal Conic 1sp	same as stereographic
Mercator	same as Polar stereographic
Mollweide	
	central_meridian
	false_easting
	false_northing
Oblique Stereographic	
	central_meridian
	latitude_of_origin
	scale_factor
	false_easting
	false_northing
Orthographic	
	central_meridian
	latitude_of_origin
	false_easting
	false_northing
Polyconic	same as Polar Stereographic
Polar Stereographic	
	central_meridian
	latitude_of_center
	false_easting
	false_northing
Transverse Mercator	same as Oblique Stereographic
Oblique Mercator	
	latitude_of_center
	longitude_of_center

	rectified_grid_angle
	azimuth
	scale_factor
	false_easting
	false_northing
Hotine Oblique Mercator	same as ObliqueMercator
Swiss Projection	same as Oblique Mercator
Sinusoidal	same as Mollweide
Spatial Oblique Mercator	
	false_easting
	false_northing
	ascending_orbit
	inclination_orbit
	satellite_ratio
	path_flag
	satellite_period
Spatial Oblique Mercator B	
	false_easting
	false_northing
	satellite_number
	path_number

Coordinate System Definition

This is required to support the new ids builds as DatumIdCoordinateSystemId like 63266405.

```
<COORDINATESYSTEM ID="6413" NAME="Ellipsoidal 3D CS. Axes:
latitude, longitude, ellipsoidal height. Orientations: north,
east, up. UoM: deg, deg, m." >
<XYUNIT ID="9102" />
<ZUNIT ID="9001" />
</COORDINATESYSTEM>
```

Items	Description
ID	The coordinate system id
NAME	The coordinate system name

XYUNIT	The unit of the XY axis
ZUNIT	The unit of the Z axis (if any)

Structure of the ESRI Mapping File

This file is a text file containing one line per mapping. Each line has the following structure:

```
EpsgId=Esri Name [=IsForced]
```

Where

- EpsgId is the EPSG id
- Esri Name is the name used in the ESRI SDE Server

IsForced is an optional boolean whose usage is required to force mapping of EPSG id greater than 32767

E.g.

```
26757=NAD_1927_StatePlane_Delaware_FIPS_0700
26758=NAD_1927_StatePlane_Florida_East_FIPS_0901
104113=GCS_Majuro=true
```

If IsForced is not set, all mappings for ids greater than 32767 are ignored.

The current mapping file should not be modified by users, as it contains the mapping for EPSG-defined ids. To define a mapping for a user coordinate system, the preferred way is to add a mapping inside the definition as:

```
<PROJCS ID="37200" NAME="Belge 1972 / Belge Lambert 72">
  <ESRI ID="PCS_Lambert_Conformal_Conic" />
  <UNIT ID="9001" />
  <GEOCS ID="4313" />
  <PROJECTION NAME="Lambert Conformal Conic 2sp">
    <PARAMETER NAME="central_meridian" DEG="4" MIN="22"
    SEC="4.71"/>
    <PARAMETER NAME="false_easting" VALUE="149950"/>
    <PARAMETER NAME="false_northing" VALUE="5400149"/>
    <PARAMETER NAME="latitude_of_origin" VALUE="90.0"/>
    <PARAMETER NAME="standard_parallel_1"
    VALUE="49.83333333333333"/>
    <PARAMETER NAME="standard_parallel_2"
    VALUE="51.16666666666666"/>
  </PROJECTION>
</PROJCS>
```

Installing an Optional Spatial Transformation

You can install additional spatial transformations in order to extend the set of standard available spatial transformations. For example, you can choose to install the transformation from ED50 to etrf89 over Spain.

All the optional spatial transformations are available as jar files (such as the `cots-srs-ntv2.jar` file) in the `tools/optional` directory inside the ERDAS APOLLO Server installation directory.

You must install the files in the `webapps/apollo-catalog/webapp/WEB-INF/lib` directory, the `webapps/erdas-apollo/webapp/WEB-INF/lib` directory, the `webapps/erdas-apollo/webapp/WEB-INF/lib` directory, and the `webapps/apollo-client/webapp/WEB-INF/lib` directory. Then, invoke the **ant** `<servlet_engine>` command in the `install` directory in order to rebuild the webapps (See [Rebuilding the Webapps](#)). You must redeploy the webapps for your changes to take effect.

You must also install the files in the ERDAS APOLLO Indexing System Manager and ERDAS APOLLO Style Editor. Copy the files into the `tools/lib/java` and `tools/styleeditor/lib` directories that are located in the ERDAS APOLLO Server installation directory.

ERDAS IMAGINE Projection System Configuration

WCS GIO Coverage decoders uses ERDAS IMAGINE raster engine to decode metadata and raster data. The ERDAS IMAGINE Projection System comes with an extensive library of built-in projections, spheroids, and datums and is also referred to as EPRJ. This section contains the following:

- [Spheroids and Datums](#)
- [ERDAS IMAGINE Projection Configuration Files](#)
- [Understanding Datums, Spheroids, and Projections](#)

Spheroids and Datums

Parametric Datums

A horizontal datum is a mathematical model of the Earth's surface that is used to calculate the coordinate components (for example, latitude and longitude) of a point on the surface of the Earth. This surface is defined by a spheroid and the position and orientation relationship of the spheroid to a reference mathematical model of the Earth. The georeferenced coordinates are unique only if qualified by a datum. If you go across two different datums during georeferencing without considering the coordinate shift between them, the potential error can be up to hundreds of meters.

There are three types of parametric datums supported in the IMAGINE Projection System:

- The first type of datum is defined by seven parameters referred to the reference ellipsoid WGS 84. The seven parameters are xy- z translations, omega-phi-kappa rotations, and scale variations. The Standard Molodensky transformation and sevenparameter transformation are used for parametric datum shifts.



The seven-parameter datum corresponds to the EPSG Coordinate Frame Transformation.

- The second type of datum is defined by NADCON grids in which the coordinate shifts among datum NAD 27, NAD 83, and HARN are calculated by bilinear interpolation.

- The third type of datum is defined by MREs based on DMA documents. You can identify these datums by the labels ending with MRE on the Spheroid tab.

Most parametric datums applied to global areas are basically spheroids themselves without any position shift or rotations relative to WGS 84. They are assumed to have the same centers as that of WGS 84. The main reason to use the spheroid name as a global datum name is to make a smooth transition from older to newer projection versions. Avoid using them when other appropriate local datums are available.



It is not recommended that you use any datum if you are not sure what it is. Using the wrong datum can introduce significant geometric errors (up to a few hundred meters) when performing datum shift calculation.

For more information about the parametric datum shift, refer to the DMA TR 8350.2 document. For NADCON, please check with the National Geodetic Survey.

Surface Datums

Just as a parametric datum can be used to calculate the coordinate components (for example, latitude and longitude) of a point on the surface of the Earth, the surface datum is a reference surface to which heights and elevations are referred.

There are four types of surface datums:

- Constant surface datums provide a constant shift value to apply to the entire area.

Raster (or grid) surface datums use information stored in raster data files to define shift surfaces.

Multiregression surface datums use an MRE to define the shift surface.

Vector (or point) surface datums are not currently supported in the IMAGINE Projection System.



For more information about the parameters for each of these datum transformations, see [Grid Datum Example](#).

ERDAS IMAGINE Projection Configuration Files

ERDAS IMAGINE uses the following projection configuration files to define spheroids and datums:

- mapprojections.dat
- epsg.plb
- spheroid.tab
- sptable.tab
- units.dat

All of these files except for the epsg.plb file are located in this directory:

C:\Erdas\ApolloServer/tools/lib/gio/etc

mapprojections.dat

There are three projections shown in the example below. Each projection has an external or internal projection type and related parameters that are recognized by EPRJ.

```
"Kertau / R.S.O. Malaya (ch) (24571)" {  
  EXTERNAL "eprj_rso" "Modified Everest" "Kertau 1948" 0  
  0:4.0000000000000000E+004 1:0.0000000000000000E+000  
  2:1.0000000000000000E+000 "benoit_chain"  
}  
"Kertau 1968 (4245)" {  
  INTERNAL 0 "Modified Everest" "Kertau 1948" 0  
  "dd"  
}  
"Kertau 1968 / Singapore Grid (24500)" {  
  INTERNAL 51 "Modified Everest" "Kertau 1948" 0  
  2:1.0000000000000000E+000 4:1.8125768268587654E+000  
  5:2.2473673935663251E-002 6:3.0000000000000000E+004  
  7:3.0000000000000000E+004 "meters"  
}
```

epsg.plb

All of the *.plb files are located in the C:\Erdas\ApolloServer/tools/lib/gio/etc/projections directory. They are the standard projection categories that EPRJ uses. Each category contains a number of projections.

The epsg.plb file is one of the many *.plb files in this folder. For more information, see Projection Entry File Details[TBD].

spheroid.tab

For most of the projections defined by EPRJ, the supported spheroids and datums come from the spheroid.tab file. This file contains a table of spheroids and datums that EPRJ uses.

```

"Modified Everest" {
  11 6377304.063 6356103.039
  "Modified Everest" 0 0 0 0 0 0
  "Kertau 1948" -11 851 5 0 0 0
  "Kertau 1948 (MRE) Geoid" SURFACE
  {
    REGRESSION -90 0 90 360 0.418879020 -
1.256637060 0.418879020 -42.725660040
    HEIGHT =
      {
        0 0 -3.833
        0 1 -2.701
        1 0 17.861
        0 2 0.386
        2 0 0.980
        3 0 -0.793
        3 1 -2.390
        4 2 -1.455
      }
    } DESCRIPTION = "Kertau Datum 1948 (West Malaysia and
Singapore) (MRE) Geoid, from DMA TR 8350.2"
  }

```

sptable.tab

In the example below, 'EAST' is the zone. In other cases, the zone is expressed as a number or alpha-numerically.

```

"GEORGIA" {
  "EAST" 1 NAD27 3651 -1001
  0.6378206400000000e+07 0.6768657997291094e-02 -
0.8201000000000000e+08
  0.9999000000000000e+00 0.0000000000000000e+00
0.0000000000000000e+00
  0.3000000000000000e+08 0.1524003048006096e+06
0.0000000000000000e+00
}
"GEORGIA" {
  "WEST" 1 NAD27 3676 -1002
  0.6378206400000000e+07 0.6768657997291094e-02 -
0.8401000000000000e+08
  0.9999000000000000e+00 0.0000000000000000e+00
0.0000000000000000e+00
  0.3000000000000000e+08 0.1524003048006096e+06
0.0000000000000000e+00
}

```

units.dat

Units.dat is an IMAGINE Projection System file that is used to convert distance, angle, time, mass, and so on. It is a simple ASCII file that contains definitions for various types of unit categories such as distance, angle, or time. Within each category, there is a definition that consists of a unit name followed by the conversion to the standard unit.

Here is an example of the entry for distance:

```

distance
{
  meters 1.0 ;
  meter 1.0 ;
  feet 0.3048006099012192 ;
  .
  .
  .
}

```

Understanding Datums, Spheroids, and Projections

The datums, spheroids, and projections are found in the C:\Erdas\ApolloServer\tools\lib\gio\etc\spheroid.tab file, which is used by the ERDAS IMAGINE Projection System.

The following listing shows the entry from the spheroid.tab file for the Australian National spheroid:

```

"Australian National" {
15 6378160.0 6356774.719
"Australian National" 0 0 0 0 0 0 0
"Anna 1 Astro 1965" -491 -22 435 0 0 0 0
"Australian Geodetic 1966" -133 -48 148 0 0 0 0
"Australian Geodetic 1984" -134 -48 149 0 0 0 0
"Australian Geodetic 1966 (MRE) Geoid" SURFACE
{
REGRESSION -90 0 90 360 0.052359880
1.413716760 0.052359880 -7.016223920
HEIGHT =
{
0 0 -4.258
0 1 2.740
0 2 70.479
1 1 -34.946
2 0 12.676
1 2 24.680
0 4 -348.586
2 2 -14.488
3 1 122.302
0 5 37.035
4 1 -17.307
5 0 -6.704
0 6 645.556
3 3 -80.304
5 1 -108.866
0 7 -19.475
3 4 -271.301
0 8 -430.089
8 0 -5.561
2 8 97.772
4 6 449.462
8 2 -69.354
4 7 -692.991
9 3 265.504

```

```

5 9 1311.842
}
} DESCRIPTION = "Australian Geodetic 1966 (MRE) Geoid,
from DMA TR 8350.2"
"Australian Geodetic 1984 (MRE) Geoid" SURFACE
{
REGRESSION -90 0 90 360 0.052359880
1.413716760 0.052359880 -7.016223920
HEIGHT =
{
0 0 -4.155
0 1 3.939
1 0 14.693
0 2 65.144
1 1 -18.651
2 0 10.460
3 0 -120.684
0 4 -267.125
3 1 39.025
1 4 -74.039
4 1 -26.446
5 0 301.719
0 6 445.176
5 1 -33.732
6 0 16.358
3 4 384.399
7 0 -312.773
0 8 -257.076
3 5 -122.384
6 2 -117.856
8 0 -18.365
5 4 -502.439
8 1 9.820
9 0 110.502
1 9 54.807
7 5 915.970
8 7 -3661.876
9 9 3579.951
}
} DESCRIPTION = "Australian Geodetic 1984 (MRE) Geoid,
from DMA TR 8350.2"
}

```

"Australian National" is the name of the spheroid. The next line defines its sequence number in the spheroid.tab file, the semi-major axis, and the semi-minor axis (in meters). The general syntax is:

```

"Spheroid Name" {
Sequence_Number Semi-Major_Axis Semi-Minor_Axis
"Spheroid Name" 0 0 0 0 0 0 0
Datums...
}

```


Seven-Parameter Ellipsoidal Transformation

Following the spheroid definition are datums associated with each spheroid. There are two types of datums that can be used by the IMAGINE Projection System: the ellipsoidal datum and the surface datum. The ellipsoidal datums are recorded in terms of the seven parameters required to calculate a shift to the WGS84 datum.



The seven-parameter ellipsoidal transformation corresponds to The EPSG coordinate frame transformation.

Any datums added to the spheroid.tab file must also be recorded as datum shift parameters to WGS84. The parameters are recorded in one of two ways, as shown in the two examples that follow.

Parametric Datum Example

The example below shows a parametric datum:

```
"Datum Name" [PARAMETRIC] dx dy dz rw rj rk ds  
[DESCRIPTION = string]
```

Where:

dx , dy and dz are the x,y,z translations to WGS84, in meters, rw , rj , and rk are the omega, phi, kappa rotations to WGS84, in radians and scientific notation, and, ds is the scale change to WGS84 in scientific notation. The keyword `ELLIPSOIDAL` is optional. The text `DESCRIPTION` of the datum is also optional

Most parametric datums applied to global areas are basically spheroids themselves without any position shifts or rotations relative to WGS 84. They are assumed to have the same centers as that of WGS 84. The use of the "global datum" syntax below is no longer required by the IMAGINE Projection System.

Grid Datum Example

The example below shows a grid datum:

```
"Datum Name" GRID gridfilename [DESCRIPTION =  
Description of the datum]
```

Where:

The keyword `GRID` is mandatory.

The `gridfilename` is the raster file used to define the datum. The raster files used for the transformations must reside in the `C:\Erdas\ApolloServer\tools\lib\gio\etc` folder. No paths can be used to point to these files.

The text `DESCRIPTION` of the datum is optional.

Spheroid Example

The example below shows a spheroid:

```
"Spheroid Name" 0 0 0 0 0 0 0
```

The main reason to use the spheroid name as a global datum name is to make a smooth transition from older to newer projection versions. Avoid using them when other appropriate local datums are available.

You can add new datums to an existing spheroid by editing the `spheroid.tab` file with any text editor and adding a new line for each datum in the section for that spheroid. You can also add a new spheroid with associated datums by adding a block of text to the end of the file using the following syntax:

```
"Spheroid Name" {  
Sequence_Number Semi-Major_Axis Semi-Minor_Axis  
"Datum Name 1" dx dy dz rw rj rk ds  
"Datum Name 2" dx dy dz rw rj rk ds  
"Datum Name 3" dx dy dz rw rj rk ds  
"Datum Name n" dx dy dz rw rj rk ds  
}
```

Surface Datum Types

In addition to the ellipsoidal datum, the ERDAS IMAGINE Projection System supports surface datums. There are four types of surface datums:

- Constant
- Raster (or Grid)
- Vector (or Point)
- Multiregression

Of these, the ERDAS IMAGINE Projection System currently supports the constant, raster, and multiregression datums.

Each one of the three supported datums has its own parameters.

CONSTANT datums provide a constant shift value to apply to the entire area.

```
"Datum Name" SURFACE [BASEDATUM="DatumName"]
{
CONSTANT minlat minlon maxlat maxlon latshift lonshift
htshift
}
DESCRIPTION = "transformation description"
```

Where:

SURFACE indicates the type of surface datum

BASEDATUM indicates the datum that is being used as the base for the shift surface. If no **BASEDATUM** is declared, the **BASEDATUM** defaults to the datum with the same name as the spheroid. Any other basedatum must be declared and predefined based upon the same base spheroid. The **BASEDATUM** can be either an ellipsoidal or surface datum.

CONSTANT indicates that a constant shift is performed on the **BASEDATUM**.

minlat provides the minimum latitude value of the bounding box in decimal degrees.

minlon provides the minimum longitude value of the bounding box in decimal degrees.

maxlat provides the maximum latitude value of the bounding box in decimal degrees.

maxlon provides the maximum longitude value of the bounding box in decimal degrees.

latshift the amount of shift in the latitude direction in decimal seconds.

lonshift the amount of shift in the longitude direction in decimal seconds.

htshift the amount of shift in the elevation in meters.

DESCRIPTION provides a description about the transformation, including the basedatum and the source for the shift constants.

RASTER surface datums use information stored in raster data files to provide the transformation information.

The raster files used for the transformations must reside in the C:\Erdas\ApolloServer\tools\lib\gio\etc older. No paths can be used to point to these files.

```
"Datum Name" SURFACE [BASEDATUM="DatumName"]
{
  RASTER RESAMPLE="Resample_Method"
  noDataValue
  LATITUDE = "LatFile"
  LONGITUDE = "LonFile"
  HEIGHT = "HtFile"
}
DESCRIPTION = "transformation description"
```

Where:

`SURFACE` indicates the type of surface datum.

`BASEDATUM` indicates the datum that is being used as the base for the shift surface. If no `BASEDATUM` is declared, the `BASEDATUM` defaults to the datum with the same name as the spheroid. Any other basedatum must be declared and predefined based upon the same base spheroid. The `BASEDATUM` can be either an ellipsoidal or a surface datum.

`RASTER` indicates that a raster shift surface based on the `BASEDATUM` is defined.

`RESAMPLE` defines the resampling method used on the raster files. The `BILINEAR` resampling method is the default and is optional. The `BICUBIC SPLINE` resampling method must be declared.

`noDataValue` defines the value to substitute for null values in the shift surface files.

`LATITUDE` indicates the raster file to use as the latitude shift surface. This file is optional.

`LONGITUDE` indicates the raster file to use as the longitude shift surface. This file is optional.

`HEIGHT` indicates the raster file to use as the elevation shift surface. This file is optional.



While LAT, LON, and HEIGHT are all optional parameters, at least one of these three must be specified to define a valid surface datum.

DESCRIPTION provides a description about the surface datum, including a more descriptive datum name and the source for the shift files.

MULTIREGRESSION surface datums use an MRE to represent the datum shift surface.

```
"Datum Name" SURFACE [BASEDATUM="DatumName"]
{
REGRESSION minlat minlon maxlat maxlon
A B C D
LATITUDE = {Vexp1 Uexp1 Coeff1
Vexp2 Uexp2 Coeff2
Vexpn Uexpn Coeffn
}
LONGITUDE = {Vexp1 Uexp1 Coeff1
Vexp2 Uexp2 Coeff2
Vexpn Uexpn Coeffn
}
Height = {Vexp1 Uexp1 Coeff1
Vexp2 Uexp2 Coeff2
Vexpn Uexpn Coeffn
}
} DESCRIPTION = "transformation description"
```

Where:

SURFACE indicates the type of surface datum.

BASEDATUM indicates the datum that is being used as the base for the shift surface. If no **BASEDATUM** is declared, the **BASEDATUM** defaults to the datum with the same name as the spheroid. Any other basedatum must be declared and predefined based upon the same base spheroid. The **BASEDATUM** can be either an Ellipsoidal or a surface datum.

REGRESSION indicates that a multiregression transformation will be performed on the **BASEDATUM**.

minlat provides the minimum latitude value of the bounding box in decimal degrees.

minlon provides the minimum longitude value of the bounding box in decimal degrees.

maxlat provides the maximum latitude value of the bounding box in decimal degrees.

maxlon provides the maximum longitude value of the bounding box in decimal degrees.

A, B are the coefficients for the longitude shift. If **v** is the latitude,

$$V = A * v + B$$

C, D are the coefficients for the latitude shift. If u is the latitude,

$$U = C * u + D$$

LATITUDE lists the exponents and coefficients in decimal seconds for the MRE. Each row gives the V exponent, the U exponent, and the coefficient for one term in the MRE. The sum of all of these terms is the result of the MRE. This surface is optional.

LONGITUDE lists the exponents and coefficients in decimal seconds for the MRE. Each row gives the V exponent, the U exponent, and the coefficient for one term in the MRE. The sum of all of these terms is the result of the MRE. This surface is optional.

Height lists the exponents and coefficients in meters for the MRE. Each row gives the V exponent, the U exponent, and the coefficient for one term in the MRE. The sum of all of these terms is the result of the MRE. This surface is optional.



While LAT, LON, and HEIGHT are all optional parameters, at least one of these three must be specified to successfully perform a shift surface transformation.

DESCRIPTION provides a description about the surface datum, including a more descriptive datum name and the source for the shift files.

NOTE: For detailed information on how the ERDAS IMAGINE Projection System works, please see the ERDAS IMAGINE documentation.

[1] A georeferenced image has a position, a location that is either stored in the file or in a world file associated with the image. Please refer to the Image Data Model chapter for more information on this subject.

