
EOxServer

EOxServer Documentation

Release 0.4.1dev

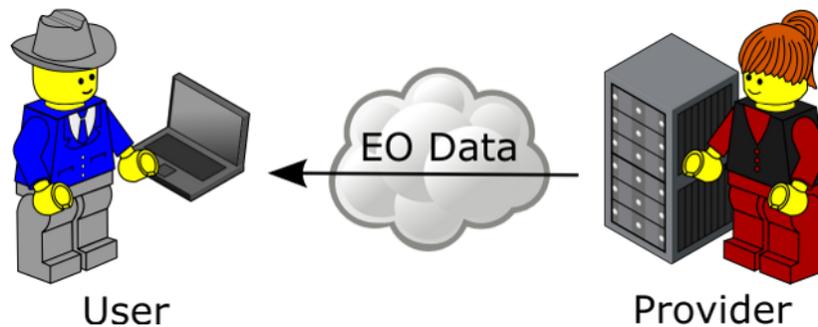
Stephan Meissl **Stephan Krause**
Fabian Schindler **Gerhard Triebnig**
Milan Novacek **Arndt Bonitz** **Martin Paces**
Joachim Ungar **Marko Locher** **Christian Schiller**

Jul 25, 2017

Contents

1	Users' Guide	1
2	Developers' Guide	121
3	Requests for Comments	147
4	Release Notes	225
5	API Reference	229
6	License	263
7	Credits	265

This section is intended for users of the EOxServer software stack. Users range from administrators installing and configuring the software stack and operators registering the available *EO Data* on the *Provider* side to end users consuming the registered *EO Data* on the *User* side.



Developers needing to know all the nitty-gritty about EOxServer implementation and APIs please refer to the *Developers' Guide* (page 121).

EOxServer Basics

Table of Contents

- *EOxServer Basics* (page 1)
 - *Introduction* (page 2)
 - * *What is EOxServer?* (page 2)
 - * *What are the main features of EOxServer?* (page 2)
 - * *Where can I get it?* (page 2)
 - * *Where can I get support?* (page 3)
 - * *EOxServer Documentation* (page 3)

- * *Demonstration Services* (page 3)
- *Data Model* (page 3)
- *Service Model* (page 4)
 - * *Web Coverage Service* (page 4)
 - * *Web Map Service* (page 4)
 - * *Web Processing Service* (page 5)

Introduction

What is EOxServer?

EOxServer is an open source software for registering, processing, and publishing Earth Observation (EO) data via different Web Services. EOxServer is written in Python and relies on widely-used libraries for geospatial data manipulation.

The core concept of the EOxServer data model is the one of a coverage. In this context, a coverage is a mapping from a domain set (a geographic region of the Earth described by its coordinates) to a range set. For original EO data, the range set usually consists of measurements of some physical quantity (e.g. radiation for optical instruments).

The EOxServer service model is designed to deliver (representations of) EO data using open standard web service interfaces as specified by the [Open Geospatial Consortium](#)⁴ (OGC).

What are the main features of EOxServer?

- Repository for Earth Observation data
- OGC Web Services
- Administration Tools
- Web Client
- Identity Management System

Where can I get it?

You can get the EOxServer source from

- the [EOxServer Download page](#)⁵
- the [Python Package Index \(PyPi\)](#)⁶
- the [EOxServer Git repository](#)⁷

Additionally the following binary packages are provided:

- Enterprise Linux RPMs from [EOX' YUM repository](#)⁸

The recommended way to install EOxServer on your system is to use the Python installer utility [pip](#)⁹.

Please refer to the [Installation](#) (page 15) document for further information on installing the software.

⁴ <http://www.opengeospatial.org>

⁵ <http://eoxserver.org/wiki/Download>

⁶ <http://pypi.python.org/pypi/EOxServer/>

⁷ <https://github.com/EOxServer/eoxserver>

⁸ <http://packages.eox.at>

⁹ <http://www.pip-installer.org/en/latest/index.html>

Where can I get support?

If you have questions or problems, you can get support at the official EOxServer Users' mailing list users@eoxserver.org. See *Mailing Lists* (page 36) for instructions how to subscribe.

Documentation is available on this site and as a part of the EOxServer source.

EOxServer Documentation

The EOxServer documentation consists of the

- *Users' Guide* (page 1) (which this document is part of)
- *Developers' Guide* (page 121) (where you can find implementation details)
- *Requests for Comments* (page 147) (where you can find high-level design documentation)

Furthermore, you can consult the inline documentation in the source code e.g. in the [Source Browser](#)¹⁰.

Demonstration Services

There is a demonstration service available on the EOxServer site. You can reach it under http://eoxserver.org/demo_stable/ows. For some sample calls to different OGC Web Services, see *Demonstration* (page 37).

Data Model

The EOxServer data model describes which data can be handled by the software and how this is done. This section gives you a short overview about the basic components of the data model.

The term coverage is introduced by the OGC Abstract Specification. There, coverages are defined as a mapping between a domain set that can be referenced to some region of the Earth to a range set which describes the possible values of the coverage. This is, of course, a very abstract definition. It comprises everything that has historically been called "raster data" (and then some, but that is out of scope of EOxServer at the moment).

The data EOxServer originally was designed for is satellite imagery. So the domain set is the extent of the area that was scanned by the respective sensor, and the range set contains its measurements, e.g. the radiation of a spectrum of wavelengths (optical data).

In the language of the OGC Abstract Specification ortho-rectified data corresponds to "rectified grid coverages", whereas data in the original geometry corresponds to "referenceable grid coverages".

The EOxServer coverage model relies heavily on the data model of the Web Coverage Service 2.0 Earth Observation Application Profile which is about to be approved by OGC. This profile introduces different categories of Earth Observation data:

- Rectified or Referenceable Datasets roughly correspond to satellite scenes, either ortho-rectified or in the original geometry
- Rectified Stitched Mosaics are collections of Rectified Datasets that can be combined to form a single coverage
- Dataset Series are more general collections of Datasets; they are just containers for coverages, but not coverages themselves

Datasets, Stitched Mosaics and Dataset Series are accompanied by Earth Observation metadata. At the moment, EOxServer supports a limited subset of metadata items, such as the identifier of the Earth Observation product, the acquisition time and the acquisition footprint.

¹⁰ <https://github.com/EOxServer/eoxserver>

Service Model

Earth Observation data are published by EOxServer using different OGC Web Services. The OGC specifies open standard interfaces for the exchange of geospatial data that shall ensure interoperability and universal access to geodata.

Web Coverage Service

The OGC [Web Coverage Service](http://www.opengeospatial.org/standards/wcs)¹¹ (WCS) is designed to deliver original coverage data. EOxServer implements three versions of the WCS specification:

- version 1.0.0
- version 1.1.0
- version 2.0.1 including the Earth Observation Application Profile (EO-WCS)

Each of these versions supports three operations:

- GetCapabilities - returns an XML document describing the available coverages (and Dataset Series)
- DescribeCoverage - returns an XML document describing a specific coverage and its metadata
- GetCoverage - returns (a subset of) the coverage data

The WCS 2.0 EO-AP (EO-WCS) adds an additional operation:

- DescribeEOCoverageSet - returns an XML document describing (a subset of) the datasets contained in a Rectified Stitched Mosaic or Dataset Series

For detailed lists of supported parameters for each of the operations see [EO-WCS Request Parameters](#) (page 45) .

In addition, EOxServer supports the WCS 1.1 Transaction operation which provides an interface to ingest coverages and metadata into an existing server.

Web Map Service

The OGC [Web Map Service](http://www.opengeospatial.org/standards/wms)¹² (WMS) is intended to provide portrayals of geospatial data (maps). In EOxServer, WMS is used for viewing purposes. The service provides RGB or grayscale representations of Earth Observation data. In some cases, the Earth Observation data will be RGB imagery itself, but in most cases the bands of the images correspond to other parts of the wavelength spectrum or other measurements altogether.

EOxServer implements WMS versions 1.0, 1.1 and 1.3 as well as parts of the Earth Observation Application Profile for WMS 1.3. The basic operations are:

- GetCapabilities - returns an XML document describing the available layers
- GetMap - returns a map

For certain WMS 1.3 layers, there is also a third operation available

- GetFeatureInfo - returns information about geospatial features (in our case: datasets) at a certain position on the map

Every coverage (Rectified Dataset, Referenceable Dataset or Rectified Stitched Mosaic) is mapped to a WMS layer. Furthermore, Dataset Series are mapped to WMS layers as well. In WMS 1.3 a “bands” layer is appended for each coverage that allows to select and view a subset of the coverage bands only. Furthermore, queryable “outlines” layers are added for Rectified Stitched Mosaics and Dataset Series which show the footprints of the Datasets they contain.

¹¹ <http://www.opengeospatial.org/standards/wcs>

¹² <http://www.opengeospatial.org/standards/wms>

Web Processing Service

The OGC [Web Processing Service](http://www.opengeospatial.org/standards/wps)¹³ (WPS) is intended to make processing resources for geospatial data available online. EOxServer features an implementation of this standard as well.

The WPS server provides three operations:

- GetCapabilities - returns an XML document describing the available processes
- DescribeProcess - returns an XML document describing a specific process
- Execute - allows to invoke a process

Global Use Case

Table of Contents

- *Global Use Case* (page 5)
 - *The General Provider View* (page 6)
 - * *Environment & Software Configuration* (page 8)
 - * *Data Registration* (page 8)
 - *The General User View* (page 10)
 - * *Web Browser* (page 13)
 - * *GIS Tool* (page 13)

This section describes the global Use Case of EOxServer including concrete usage scenarios as examples.

Figure: “*Parties involved in the EOxServer Global Use Case* (page 5)” introduces the involved parties in this global Use Case.

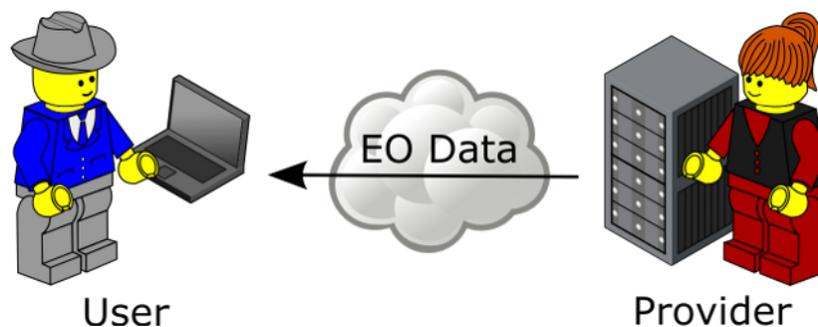


Fig. 1.1: *Parties involved in the EOxServer Global Use Case*

On the one side there is a provider of Earth Observation (EO) data. The provider has a possibly huge, in terms of storage size, archive of EO data and wants to provide this data to users. Of course the data provision has to follow certain constraints and requirements like technical, managerial, or security frame conditions but in general the provider wants to reach as many users as possible with minimal efforts.

On the other side there is a user of EO data. The user has the need of certain EO data as input to some processing which varies from simple viewing to complex data analysis and generation of derived data. The user wants to obtain the needed EO data as easily as possible which includes finding the right data from the right provider at the right time at the right location and retrieving it in the right representation e.g. format.

¹³ <http://www.opengeospatial.org/standards/wps>

Already from this simple constellation the need for standardized interfaces is evident. Thus EOxServer implements the open publicly available interface standards defined by the Open Geospatial Consortium (OGC). In particular EOxServer contains an implementation of the Web Coverage Service (WCS) including its Earth Observation Application Profile (EO-WCS) and the Web Map Service (WMS) again including its EO extension (EO-WMS).

These interface standards have been chosen to support the new paradigm of “zooming to the data”. This means looking at previews of the data rather than searching in a catalogue in order to find the right data. WMS together with its EO extension is used for the previews whereas WCS with its EO extensions is used to download the previously viewed data and metadata. Of course a provider is free to operate a catalogue in parallel including references to the EO-WMS and EO-WCS.

The EOxServer software stack is a collection of Open Source Software designed to enhance a wide range of legacy systems of EO data archives with controlled Web-based access (“online data access”) with minimal efforts for the provider. The user not only significantly benefits from the provider’s enhanced online data access but also from the client functionalities included in the EOxServer software stack.

In particular the EOxServer software stack provides the following features:

- easy to install
- simple yet powerful web interface for data registration for the provider
- standardized way to access geographic data i.e. via EO-WMS and EO-WCS
- download of subsets of data
- on the fly re-projection, re-sampling, and format conversion
- visual preview of data
- integrated usage of EO-WMS and EO-WCS to view and download the same data
- intelligent automated handling of EO collections and mosaics
- homogeneous way to access different data, metadata, and packaging formats
- homogeneous access to different storage systems i.e. file system, ftp, and rasdaman

These features result in the general benefit for the provider to be more attractive to the user.

The following sub-sections provide details from the provider and user point of view highlighting the possible usage of the EOxServer software stack.

The General *Provider* View

The provider operates an archive of EO data with different ways of actually accessing the data. For simplicity let’s assume the data archived in this legacy system can be accessed in two ways. First there is the local access directly to the file system via operating system capabilities. Second there is an online access by exposing certain directories via FTP.

The EOxServer software stack acts as a middle-ware layer in front of the legacy archive system that expands the offered functionality and thus widens the potential user or customer base. The additional functionality compared to plain FTP access includes:

- interoperable online access via a standard interface defined by an accepted international industry consortium
- domain and range sub-setting of coverages allowing to download only the needed parts of a coverage and thus saving bandwidth
- spatial-temporal search within the offered coverages
- on-the-fly mosaicking
- on-the-fly re-projection
- delivery in multiple encoding formats i.e. on-the-fly format conversion
- on-the-fly scaling and re-sampling

- preview via EO-WMS
- embedding of metadata (EO-O&M) adjusted to the actual delivered coverage

Figure: “*Provider View* (page 7)” provides an overview of the provider environment showing the provider’s legacy system and the extending EOxServer software stack.

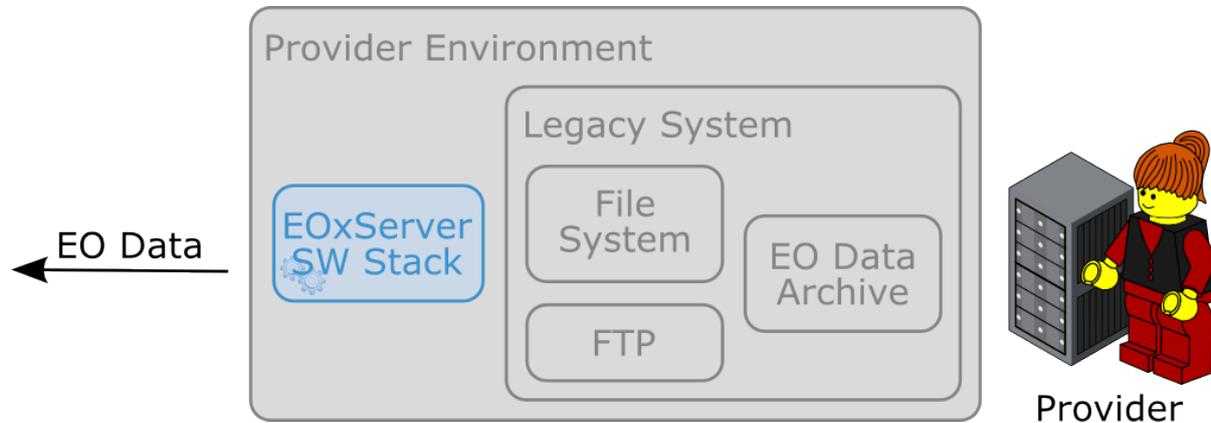


Fig. 1.2: *Provider View*

The recommended way for the installation of the EOxServer software stack is to use a host which has direct read access to the data via the file system using operating system capabilities. If this file system is physically located on the same hardware host or if it is mounted from some remote storage e.g. via NFS or Samba doesn't matter in terms of functionality. However, in terms of performance the actual configuration has some impact as big data might have to be transferred over the network with different bandwidths.

The other option is to use the read access via FTP which is a practical configuration in terms of functionality. However, in terms of performance this isn't the recommended configuration because of the need to always transfer whole files even if only a subset is needed. Various caching strategies will significantly improve this configuration, though.

After the installation of all software components needed for the EOxServer software stack there are two main activities left for the provider:

- Configure the environment (e.g. register service endpoint(s) in a web server) and EOxServer (e.g. enable or disable components like services)
- Register data

Figure: “*Activities to Enhance the Provider’s Environment* (page 7)” shows these activities needed to enhance the provider’s environment with online data access to the EO data archive legacy system.

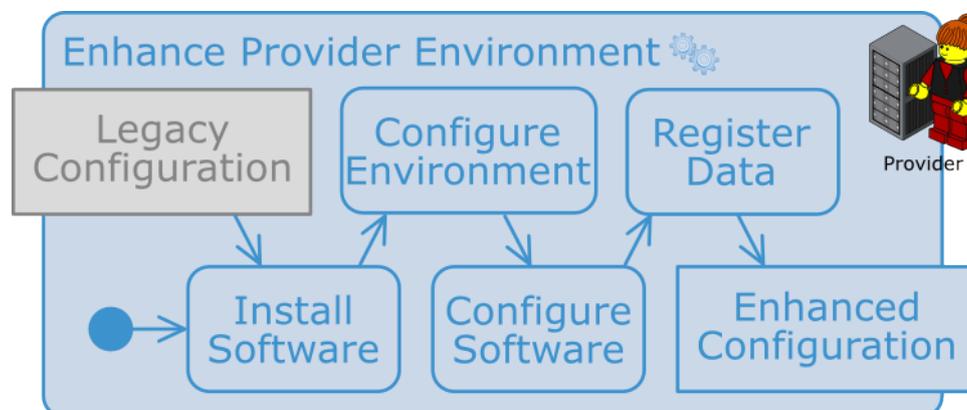


Fig. 1.3: *Activities to Enhance the Provider’s Environment*

Environment & Software Configuration

The EOxServer software stack consists of the EOxServer, the Identity Management, and the Applications Interface software components.

The Identity Management layer is an optional layer on top of EOxServer. Thus and because its configuration is extensively discussed in section *Identity Management System* (page 74) we skip it here.

The Applications Interface software components are discussed in detail in section *The General User View* (page 10) below.

As EOxServer is based on Python, MapServer, GDAL/OGR, and Django these software components need to be installed first. The base configuration of EOxServer consists of the generation of an EOxServer instance and registering it in a web server.

The EOxServer instance generation includes the configuration of various parameters like database name, type, and connection info, instance id, paths to logfiles, temporary directories, etc. as well as the initialization of its database. There are two options for the database management system (DBMS). The first is SQLite together with SpatialLite which is a single file DBMS and thus best suited for testing purposes. The second is PostgreSQL together with PostGIS which is a full fledged DBMS with numerous management functionalities and thus best suited for operational environments.

The database itself holds the configuration of components and resources (e.g. is WCS 1.0.0 enabled) as well as the coverage metadata ingested during registration (see section *Data Registration* (page 8)).

EOxServer can be operated with any web server that supports the [Python WSGI standards](#)¹⁴. For testing and implementation purposes the Django framework directly provides a simple web server. However, in operational environments the recommended deployment of EOxServer is to use the well-known [Apache web server](#)¹⁵ together with [mod_wsgi](#)¹⁶. In most cases it will be the easiest, fastest, and most stable deployment choice.

At this point the provider's administrator or operator can actually run the software stack and configure the remainder via EOxServer's admin app. This app is accessed via a standard web browser and, when using Django's internal web server, available at the URL: "<http://localhost:8000/admin>". Use the user credentials that have been set in the database initialization step.

Figure: "*Admin app - Start* (page 9)" shows the admin app after successful login. On the left side the four modules "Auth", "Backends", "Core", and "Coverages" are shown. "Auth" is the internal Django user management module which is at the moment only used for the admin app itself. "Backends" and "Coverages" are the modules for data registration which is described in section *Data Registration* (page 8) below.

The "Core" module is used to enable or disable EOxServer components like services. The provider can decide which services and even which versions of which services EOxServer shall expose. A possible configuration is to expose WCS 2.0 and WMS 1.3.0 which are the latest versions but not any older version. In the default database initialization all services are enabled.

Data Registration

The data registration is done via the functionalities provided by the "Backends" and "Coverages" modules of the admin app. Figure: "*Admin app - Start* (page 9)" shows for which data types, or models in Django terminology, instances can be added or changed in these modules. These data types correspond to tables in the database. Only a subset of the full data model (see Figure: "*EOxServer Data Model for Coverage Resources* (page 123)") is shown in the admin app because some are filled automatically upon saving and some are included in the available ones like TileIndex in Stitched Mosaics.

The Dataset Series provides a convenient way to register a complete dataset series or collection at once. Figure: "*Admin app - Add/Change Dataset Series* (page 10)" shows the admin app when changing a Dataset Series instance. The operator has to provide an "EO ID" and an "EO Metadata Entry". All other parameters are optional as can be seen by the usage of normal instead of bold face text. However, in order to actually register coverages either one or multiple "Data sources", consisting of a "Location" e.g. a data directory and a "Search pattern", have to be

¹⁴ <https://docs.djangoproject.com/en/1.4/howto/deployment/>

¹⁵ <http://httpd.apache.org>

¹⁶ <http://code.google.com/p/modwsgi/>

EOxServer Admin Client Welcome, **admin**. [Change password](#) / [Log out](#)

Site administration

Auth	
Groups	+ Add ✎ Change
Users	+ Add ✎ Change

Backends	
Cache files	+ Add ✎ Change
Ftp storages	+ Add ✎ Change
Local paths	+ Add ✎ Change
Locations	✎ Change
Rasdaman locations	+ Add ✎ Change
Rasdaman storages	+ Add ✎ Change
Remote paths	+ Add ✎ Change

Core	
Components	✎ Change

Coverages	
Bands	+ Add ✎ Change
Data packages	✎ Change
Data sources	+ Add ✎ Change
Dataset Series	+ Add ✎ Change
EO Metadata Entries	+ Add ✎ Change
Extents	+ Add ✎ Change
Layer Metadata	+ Add ✎ Change
Lineage Entries	+ Add ✎ Change
Local data packages	+ Add ✎ Change
NII Values	+ Add ✎ Change
Range Types	+ Add ✎ Change
Rasdaman data packages	+ Add ✎ Change
Rectified Datasets	+ Add ✎ Change
Remote data packages	+ Add ✎ Change
Single File Coverages	+ Add ✎ Change
Stitched Mosaics	+ Add ✎ Change
Tile Indices	+ Add ✎ Change

Recent Actions	
My Actions	None available

Fig. 1.4: Admin app - Start

added. Alternatively, the administrator can decide to register single coverages and link them to the Dataset Series via the “Advanced coverage handling” module (see Figure: “*Admin app - Add/Change Dataset Series Advanced* (page 12)”).

Fig. 1.5: *Admin app - Add/Change Dataset Series*

Figure: “*Admin app - Add/Change EO Metadata* (page 11)” shows the screen for adding or changing an EO metadata entry. The operator has to provide the “Begin of acquisition”, “End of acquisition”, and “Footprint” of the overall Dataset Series in the same way as for any EO Coverage. Calendar, clock, and map widgets are provided to ease the provision of these parameters. Optionally a full EO O&M metadata record can be supplied.

Saving a Dataset Series triggers a synchronization process. This process scans the Locations, e.g. directories and included sub-directories, of all configured Data Sources for files that follow the configured search pattern e.g. “*.tif”. All files found are evaluated using GDAL and for any valid and readable raster file a Dataset instance is generated in the database holding all metadata including EO metadata for the raster file. Of course the raster file itself remains unchanged in the file system.

Let’s look in more detail at the synchronization process and assume a plain GeoTIFF file with name “demo.tif” was found. The synchronization process extracts the necessary geographic metadata i.e. the domainSet or extent consisting of CRS, size, and bounding box directly from the GeoTIFF file. Where does the metadata come from? In order to retrieve the EO metadata at the moment the process looks for a file called “demo.xml” accompanying the GeoTIFF file. In future this may be expanded to automatically retrieve the metadata from catalogues like the ones the EOLI-SA connects to but for the moment the files have to be generated before the registration. The content of this file can either be a complete EO-O&M metadata record or a simple native metadata record containing only the mandatory parameters which are: “EOID”, “Begin of acquisition”, “End of acquisition”, and “Footprint”. If no “demo.xml” is found the process uses default values which are: file name without extension, current date and time, and full bounding box of raster file. Of course, the synchronization process can be re-run at any time e.g. from a daily, hourly, etc. cronjob.

This configuration is sufficient to bring online a complete EO data archive accessible via the file system.

A comparable synchronization process is available for FTP and rasdaman back-ends as well as for Stitched Mosaics. However, mostly these processes require more complex synchronization steps. For example, via the FTP back-end it is better to not inspect the raster files itself which would mean to completely transfer them but to retrieve the geographic information together with the EO metadata. Please refer to the remainder of this *Users’ Guide* (page 1) for detailed information and usage instructions.

The General User View

The user needs certain EO data as input to some processing. This processing ranges from simply viewing certain parameters of EO data to complex data analysis and generation of derived data. The user has an environment with the software installed needed for the processing. For simplicity let’s assume the user has two different software

EOxServer Admin Client Welcome, **admin**. [Change password](#) / [Log out](#)

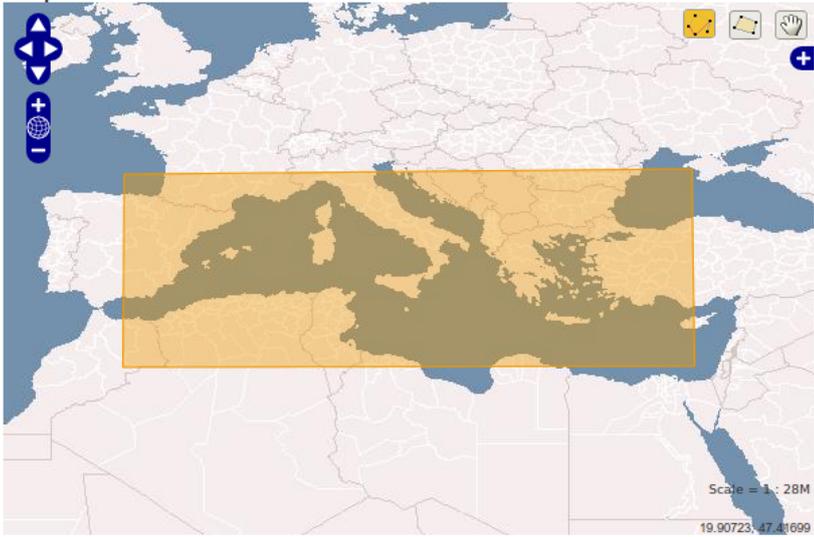
[Home](#) > [Coverages](#) > [EO Metadata Entries](#) > BeginTime: 2006-08-16 00:00:00

Change EO Metadata Entry History

Begin of acquisition: Date: Today | Time: Now |

End of acquisition: Date: Today | Time: Now |

Footprint:



Delete all Features

EO O&M:

Fig. 1.6: Admin app - Add/Change EO Metadata

EOxServer Admin Client
Welcome, **admin**. [Change password](#) / [Log out](#)

[Home](#) > [Coverages](#) > [Dataset Series](#) > MER_FRS_1P_reduced

Change Dataset Series History

Demo DatasetSeries description.

EO ID:

EO Metadata Entry: +

Data sources: +

Hold down "Control", or "Command" on a Mac, to select more than one.

Advanced coverage handling (Hide)

Hold down "Control", or "Command" on a Mac, to select more than one.

Stitched Mosaic(s): +

Available Stitched Mosaic(s)

Choose all

Chosen Stitched Mosaic(s)

Select your choice(s) and click +

mosaic_MER_FRS_1P_RGB_reduced

Clear all

Hold down "Control", or "Command" on a Mac, to select more than one.

Rectified Dataset(s): +

Available Rectified Dataset(s)

mosaic_MER_FRS_1PNPDE20060830_100949_000001972050
 mosaic_MER_FRS_1PNPDE20060822_092058_000001972050

Choose all

Chosen Rectified Dataset(s)

Select your choice(s) and click +

MER_FRS_1PNPDE20060830_100949_000001972050_00423
 MER_FRS_1PNPDE20060816_090929_000001972050_00222
 MER_FRS_1PNPDE20060822_092058_000001972050_00308
 mosaic_MER_FRS_1PNPDE20060816_090929_000001972050

Clear all

Hold down "Control", or "Command" on a Mac, to select more than one.

Referenceable Dataset(s): +

Available Referenceable Dataset(s)

Choose all

Chosen Referenceable Dataset(s)

Select your choice(s) and click +

Clear all

✖ Delete
Save and continue editing
Save and add another
Save

Fig. 1.7: Admin app - Add/Change Dataset Series Advanced

tools installed to process the data. First there is a standard web browser which manages the HTTP protocol and is capable of viewing HTML web pages. Second there is a GIS software which shall be QGIS in our example.

Figure: “*User View* (page 13)” shows the user environment and its installed software.

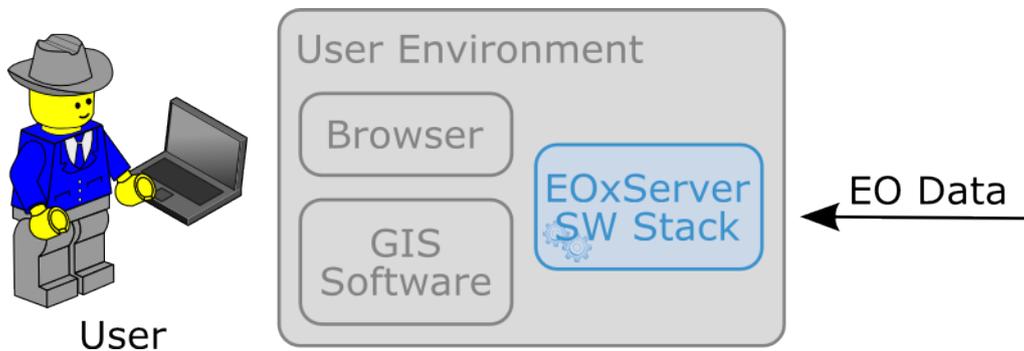


Fig. 1.8: *User View*

First of all the user needs to find an EO data provider who has data that fit the user’s purpose and who offers the data via a mechanism the user can handle. Luckily the user happens to know a provider who is running the EOxServer software stack on an EO data archive holding the required data. Thus the user can decide between several ways how to retrieve the data. Some involve client side components of the EOxServer software stack but because of the strict adherence to open standards various other ways are possible in parallel. However, we’ll focus below on two ways involving EOxServer software components.

Web Browser

In the first case the provider offers a dedicated app using EOxServer’s Web API. This app consists of HTML and Javascript files and is served via a web server from the provider’s environment. In our case the app provides access to one dataset series holding some MERIS scenes over Europe.

Figure: “*Browser app featuring EOxServer’s Web API* (page 14)” shows a screen shot of this app. The app implements the paradigm of “zooming to the data” i.e. the user directly looks at previews of the data served via EO-WMS rather than having to search in a catalogue first. After zooming to and therewith setting the Area of Interest (AoI) and setting the Time of Interest (ToI) the user following the download button is presented with the metadata of the included datasets retrieved from the offered EO-WCS. The metadata includes grid, bands, CRS, nil values, etc. of the datasets but also formats, CRSs, and interpolation methods the dataset can be retrieved in. Based on this information the user decides which datasets to download and specifies parameters of the download like spatial sub-setting, band sub-setting, CRS, size/resolution, interpolation method, format, and format specific parameters like compression. The app guides the user to specify all these parameters and downloads only the really needed data to the user’s environment. The EO-WCS protocol is used by the app transparently to the user i.e. most of the complexity of the EO-WCS protocol is hidden.

This app shows the benefit of the integrated usage of EO-WMS and EO-WCS for the online data access to the EO data archive.

The Webclient Interface (page 71) section of the documentation provides more details about the Web API.

GIS Tool

Note, that the Python Client API is not yet implemented and only available as concept.

In the second case the user wants to use the full-fledged GIS software tool QGIS and thus decides to use the handy EO-WCS plug-in provided by the provider. This plug-in makes extensive use of EOxServer’s Python Client API.

Figure: “*QGIS EO-WCS Plug-in featuring EOxServer’s Python Client API* (page 15)” shows a screen shot how the usage of the EO-WCS plug-in for QGIS might look like. The user first has to connect to the provider’s EO-WCS endpoint. Once connected the plug-in retrieves the metadata about the available dataset series and shows them as a



Fig. 1.9: Browser app featuring EOxServer's Web API

list to the user together with the tools to specify AoI and ToI. Metadata of datasets and stitched mosaics might also be retrieved in this step if the provider configured some to be directly visible in the capabilities of the EO-WCS.

The selected dataset series are transparently searched within the set spatio-temporal bounding box and available datasets and stitched mosaics presented to the user. After exploring and setting the download parameters like in the first case the EO-WCS plug-in downloads again only the required data sub-sets. In addition to the previous case the EO-WCS plug-in applies various strategies to limit the data download. For example if a dataset is added to the current list of layers only the currently viewed area needs to be filled with data at the resolution of the screen. In addition the data can be sub-setted to one or three bands that are shown i.e. there's no need to download numerous float32 bands just to preview the data.

With using the EOxServer software stack on the provider side the plug-in includes the possibility to exploit the integrated usage of EO-WMS and EO-WCS. This exploitation includes the displaying of previews in the two steps described above. Another feature is, that the possibly nicer looking images are retrieved from the performance optimized EO-WMS to fill the current view.

Once the user starts some sophisticated processing the plug-in retrieves the required sub-sets of the original data. Again strategies to limit the data download are applied.

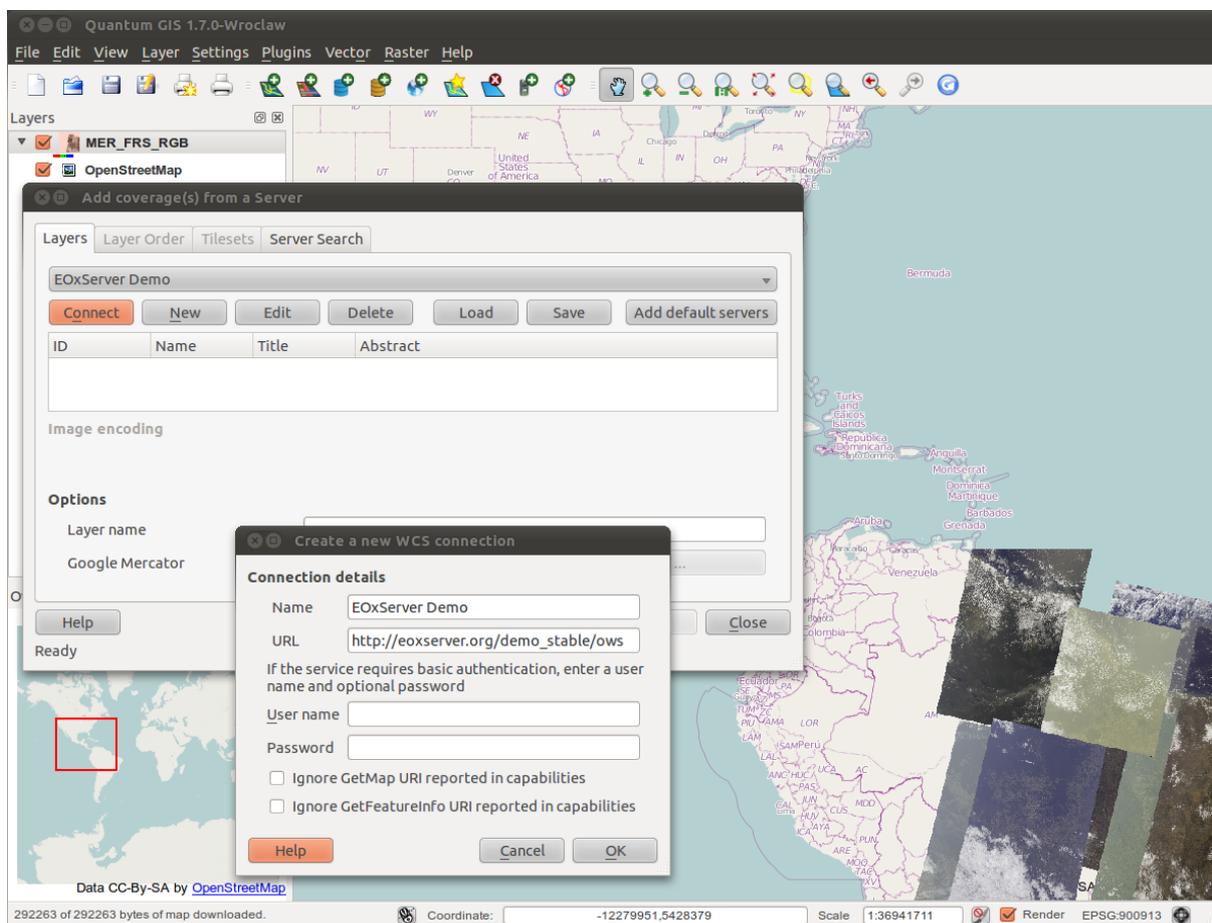


Fig. 1.10: QGIS EO-WCS Plug-in featuring EOxServer's Python Client API

Installation

Table of Contents

- *Installation* (page 15)

- *Dependencies* (page 16)
- *Installing EOxServer* (page 17)
- *Upgrading EOxServer* (page 18)
- *Hardware Guidelines* (page 18)

To use EOxServer it must be installed first. Following this guide will give you a working software installation.

See also:

- *Installation on CentOS* (page 18) for specific installation on CentOS.
- *Service Instance Creation and Configuration* (page 21) to configure an instance of EOxServer after successful installation.
- *Recommendations for Operational Installation* (page 25) to configure an operational EOxServer installation.

Dependencies

EOxServer depends on some external software. Table: “*EOxServer Dependencies* (page 16)” below shows the minimal required software to run EOxServer.

Table 1.1: EOxServer Dependencies

Software	Required Version	Description
Python	>= 2.5, < 3.0 (>=2.6.5 for Django 1.5)	Scripting language
Django	>= 1.4 (1.5 for PostGIS 2.0 support)	Web development framework written in Python including the GeoDjango extension for geospatial database back-ends.
GDAL	>= 1.7.0 (1.8.0 for rasdaman support)	Geospatial Data Abstraction Library providing common interfaces for accessing various kinds of raster and vector data formats and including a Python binding which is used by EOxServer
GEOS	>= 3.0	GEOS (Geometry Engine - Open Source) is a C++ port of the Java Topology Suite (JTS).
libxml2	>= 2.7	Libxml2 is the XML C parser and toolkit developed for the Gnome project.
lxml	>= 2.2	The lxml XML toolkit is a Pythonic binding for the C libraries libxml2 and libxslt.
MapServer	>= 6.2 (works partly with 6.0)	Server software implementing various OGC Web Service interfaces including WCS and WMS. Includes a Python binding which is used by EOxServer.

The Python bindings of the GDAL, MapServer (MapScript) and libxml2 libraries are required as well.

EOxServer is written in Python¹⁷ and uses the Django¹⁸ framework which requires a Python version from 2.5 to 2.7. Due to backwards incompatibilities in Python 3.0, Django and thus EOxServer does not currently work with Python 3.0.

EOxServer makes heavy usage of the OSGeo¹⁹ projects GDAL²⁰ and MapServer²¹.

EOxServer also requires a database to store its internal data objects. Since it is built on Django, EOxServer is mostly database agnostic, which means you can choose from various database systems. Since EOxServer requires

¹⁷ <http://www.python.org/>

¹⁸ <https://www.djangoproject.com>

¹⁹ <http://osgeo.org>

²⁰ <http://www.gdal.org>

²¹ <http://mapserver.org>

the database to have geospatial enablement, the according extensions to that database have to be installed. We suggest you use one of the following:

- For testing environments or small amounts of data, the [SQLite](#)²² database provides a lightweight and easy-to-use system.
- However, if you'd like to work with a “large” database engine in an operational environment we recommend installing [PostgreSQL](#)²³.

For more and detailed information about database backends please refer to [Django database notes](#)²⁴ and [GeoDjango installation](#)²⁵.

Table 1.2: Database Dependencies

Backend	Required Version	Required extensions/software
SQLite	>= 3.6	spatialite (>= 2.3), pysqlite2 (>= 2.5), GEOS (>= 3.0), PROJ.4 (>= 4.4)
PostgreSQL	>= 8.1	PostGIS (>= 1.3), GEOS (>= 3.0), PROJ.4 (>= 4.4), psycopg2 (== 2.4.1)

Installing EOxServer

There are several easy options to install EOxServer:

- Install an official release of EOxServer, the best approach for users who want a stable version and aren't concerned about running a slightly older version of EOxServer. You can install EOxServer either from

- [PyPI - the Python Package Index](#)²⁶ using [pip](#)²⁷:

```
sudo pip install eoxserver
```

- or from the [EOxServer release page](#)²⁸ using pip:

```
sudo pip install https://github.com/EOxServer/eoxserver/releases/download/  
↪release-<version>/EOxServer-<version>.tar.gz
```

or manually:

```
wget https://github.com/EOxServer/eoxserver/releases/download/release-  
↪<version>/EOxServer-<version>.tar.gz .  
tar xvfz EOxServer-<version>.tar.gz  
cd EOxServer-<version>  
sudo python setup.py install
```

- or binaries provided by your operating system distribution e.g. [CentOS](#) (page 18).

- Install the latest development version, the best option for users who want the latest-and-greatest features and aren't afraid of running brand-new code. Make sure you have [git](#)²⁹ installed and install EOxServer's main development branch using pip:

```
sudo pip install git+https://github.com/EOxServer/eoxserver.git
```

or manually:

²² <http://sqlite.org/>

²³ <http://www.postgresql.org/>

²⁴ <https://docs.djangoproject.com/en/1.4/ref/databases/>

²⁵ <https://docs.djangoproject.com/en/1.4/ref/contrib/gis/install/>

²⁶ <http://pypi.python.org/pypi>

²⁷ <http://www.pip-installer.org/en/latest/index.html>

²⁸ <https://github.com/EOxServer/eoxserver/releases>

²⁹ <http://git-scm.com/>

```
mkdir eoxserver_git
git clone git@github.com:EOxServer/eoxserver.git eoxserver_git
cd eoxserver_git
sudo python setup.py install
```

If the directory EOxServer is installed to is not on the Python path, you will have to configure the deployed instances accordingly, see *Deployment* (page 23) below.

The successful installation of EOxServer can be tested using the *autotest instance* (page 129) which is described in more detail in the *Developers' Guide* (page 121).

Now that EOxServer is properly installed the next step is to *create and configure a service instance* (page 21).

Upgrading EOxServer

To upgrade an existing installation of EOxServer simply add the `-upgrade` switch to your pip command e.g.:

```
sudo pip install --upgrade eoxserver
```

or rerun the manual installation as explained above.

Please carefully follow the *migration/update procedure* (page 33) corresponding to your version numbers for any configured EOxServer instances in case of a major version upgrade.

Hardware Guidelines

EOxServer has been deployed on a variety of different computers and virtual machines with commonplace hardware configurations. The typical setup is:

- a dual-core or quad-core CPU
- 1 to 4 GB of RAM

The image processing operations required for certain OGC Web Service requests (subsetting, reprojection, re-sampling) may be quite expensive in terms of CPU load and memory consumption, so adding more RAM or an additional core (for VMs) may increase the performance of the service. Bear in mind however, that disk I/O speed is often a bottleneck.

Installation on CentOS

Table of Contents

- *Installation on CentOS* (page 18)
 - *Prerequisites* (page 19)
 - *Installation from RPM Packages* (page 19)
 - * *Preparation of RPM Repositories* (page 19)
 - * *Installing EOxServer* (page 19)
 - *Alternate installation method using pip* (page 20)
 - * *Required Software Packages* (page 20)
 - * *Installing EOxServer* (page 20)
 - *Special pysqlite considerations* (page 20)

This section describes specific installation procedure for EOxServer on CentOS³⁰ GNU/Linux based operating systems. In this example, a raw CentOS 6.4 minimal image is used.

This guide is assumed (but not tested) to be applicable also for equivalent versions of the prominent North American Enterprise Linux and its clones.

See also:

- *Installation* (page 15) generic installation procedure for GNU/Linux operating systems.
- *Service Instance Creation and Configuration* (page 21) to configure an instance of EOxServer after successful installation.
- *Recommendations for Operational Installation* (page 25) to configure an operational EOxServer installation.

Prerequisites

This example requires a running CentOS installation with superuser privileges available.

Installation from RPM Packages

Preparation of RPM Repositories

The default repositories of CentOS do not provide all software packages required for EOxServer, and some packages are only provided in out-dated versions. Thus several further repositories have to be added to the system's list.

The first one is the [ELGIS \(Enterprise Linux GIS\)](http://elgis.osgeo.org/)³¹ repository which can be added with the following *yum* command:

```
sudo rpm -Uvh http://elgis.osgeo.org/repos/6/elgis-release-6-6_0.noarch.rpm
```

The second repository to be added is [EPEL \(Extra Packages for Enterprise Linux\)](http://www.fedoraproject.org/wiki/Extra_packages_for_Enterprise)³² again via a simple *yum* command:

```
sudo yum install epel-release
```

Finally EOxServer is available from the yum repository at packages.eox.at³³. This repository offers current versions of packages like [MapServer](http://www.mapserver.org/)³⁴ as well as custom built ones with extra drivers enabled like [GDAL](http://gdal.org/)³⁵ and/or with patches applied like [libxml2](http://libxml2.org/)³⁶. It is not mandatory to use this repository as detailed below but it is highly recommended in order for all features of EOxServer to work correctly. The repository is again easily added via a single *yum* command:

```
sudo rpm -Uvh http://yum.packages.eox.at/el/eox-release-6-2.noarch.rpm
```

Installing EOxServer

Once the RPM repositories are configured EOxServer and all its dependencies are installed via a single command:

```
sudo yum install EOxServer
```

³⁰ <http://www.centos.org/>

³¹ http://wiki.osgeo.org/wiki/Enterprise_Linux_GIS

³² <http://fedoraproject.org/wiki/EPEL>

³³ <http://packages.eox.at>

³⁴ <http://mapserver.org/>

³⁵ <http://gdal.org/>

³⁶ <http://xmlsoft.org/>

To update EOxServer simply run the above command again or update the whole system with:

```
sudo yum update
```

Please carefully follow the *migration/update procedure* (page 33) corresponding to your version numbers for any configured EOxServer instances in case of a major version upgrade.

Further packages may be required if additional features (e.g: a full DBMS) are desired. The following command for example installs all packages needed when using SQLite:

```
sudo yum install sqlite libspatialite python-pysqlite python-pyspatialite
```

Alternatively the PostgreSQL DBMS can be installed as follows:

```
sudo yum install postgresql postgresql-server postgis python-psycopg2
```

To run EOxServer behind the Apache web server requires the installation of this web server:

```
sudo yum install httpd mod_wsgi
```

Now that EOxServer is properly installed the next step is to *create and configure a service instance* (page 21).

Alternate installation method using *pip*

Required Software Packages

The installation via *pip* builds EOxServer from its source. Thus there are some additional packages required which can be installed using:

```
sudo yum install gdal gdal-python gdal-devel mapserver mapserver-python \  
libxml2 libxml2-python python-lxml python-pip \  
python-devel gcc
```

Installing EOxServer

For the installation of Python packages *pip*³⁷ is used, which itself was installed in the previous step. It automatically resolves and installs all dependencies. So a simple:

```
sudo pip-python install eoxserver
```

suffices to install EOxServer itself.

To upgrade an existing installation of EOxServer simply add the `--upgrade` switch to your *pip* command:

```
sudo pip-python install --upgrade eoxserver
```

Please don't forget to follow the update procedure for any configured EOxServer instances in case of a major version upgrade.

Now that EOxServer is properly installed the next step is to *create and configure a service instance* (page 21).

Special *pysqlite* considerations

When used with *spatialite*³⁸ EOxServer also requires *pysqlite*³⁹ and *pyspatialite* which can be either installed as RPMs from packages.eox.at⁴⁰ (see *Installing EOxServer* (page 19) above) or from source.

³⁷ <http://www.pip-installer.org/>

³⁸ <http://www.gaia-gis.it/spatialite/>

³⁹ <http://code.google.com/p/pysqlite/>

⁴⁰ <http://packages.eox.at>

If installing from source please make sure to adjust the `SQLITE_OMIT_LOAD_EXTENSION` parameter in `setup.cfg` which is set by default but not allowed for EOxServer. The following provides a complete installation procedure:

```
sudo yum install libspatialite-devel geos-devel proj-devel
sudo pip-python install pyspatialite
wget https://pysqlite.googlecode.com/files/pysqlite-2.6.3.tar.gz
tar xzf pysqlite-2.6.3.tar.gz
cd pysqlite-2.6.3
sed -e '/^define=SQLITE_OMIT_LOAD_EXTENSION$/d' -i setup.cfg
sudo python setup.py install
```

If the installation is rerun the `--upgrade` respectively the `--force` flag have to be added to the `pip-python` and `python` commands in order to actually redo the installation:

```
sudo pip-python install --upgrade pyspatialite
sudo python setup.py install --force
```

Service Instance Creation and Configuration

Table of Contents

- [Service Instance Creation and Configuration](#) (page 21)
 - [Instance Creation](#) (page 21)
 - [Instance Configuration](#) (page 22)
 - [Database Setup](#) (page 23)
 - [Deployment](#) (page 23)
 - [Data Registration](#) (page 24)

Speaking of EOxServer we distinguish the common EOxServer installation (the installed code implementing the software functionality) and EOxServer instances. An instance is a collection of data and configuration files that enables the deployment of a specific service. A single server will typically contain a single software installation and one or more specific instances.

This section deals with the creation and configuration of EOxServer instances.

See also:

- [Installation](#) (page 15) generic installation procedure for GNU/Linux operating systems.
- [Installation on CentOS](#) (page 18) for specific installation on CentOS.
- [Recommendations for Operational Installation](#) (page 25) to configure an operational EOxServer installation.

Instance Creation

To create an instance, we recommend to use the `eoxserver-instance.py` script that comes with EOxServer:

```
Usage:          eoxserver-instance.py [options] INSTANCE_ID [Optional
destination directory]
```

Creates a new EOxServer instance with name `INSTANCE_ID` in the current or optionally given directory with all necessary files and folder structure. If the `--init-spatialite` flag is set, then an initial sqlite database will be created and initialized.

Options:

- h, --help** show this help message and exit
- init-spatialite** Flag to initialize the sqlite database.

Instance Configuration

Every EOxServer instance has various configuration files:

- `settings.py` - [template](#)⁴¹
- `urls.py` - [template](#)⁴²
- `conf/eoxserver.conf` - [template](#)⁴³

For each of them there is a template in the `eoxserver/instance_template` directory of the EOxServer distribution (referenced above) which is copied and adjusted by the `eoxserver-instance.py` script to the instance directory. If you create an EOxServer instance without the script you can copy those files and edit them yourself.

The file `settings.py` contains the Django configuration. Settings that need to be customized:

- `PROJECT_DIR`: Absolute path to the instance directory.
- `DATABASES`: The database connection details. For detailed information see [Database Setup](#) (page 23)
- `COMPONENTS`: The EOxServer components enabled for this instance. This is the main way how the active functionality of EOxServer is controlled, and also a way to extend the existing capabilities with extensions. Please refer to the [Plugins](#) (page 126) section to see how this is done. By default all available components are enabled.
- `LOGGING`: what and how logs are processed and stored. EOxServer provides a very basic configuration that stores logfiles in the instance directory, but they will probably not be suitable for every instance.

You can also customize further settings, for a complete reference please refer to the [Django settings overview](#)⁴⁴.

Please especially consider the setting of the `TIME_ZONE`⁴⁵ parameter and read the Notes provided in the `settings.py` file.

The file `conf/eoxserver.conf` contains EOxServer specific settings. Please refer to the configuration options section for details.

Once you have created an instance, you have to configure and synchronize the database. If you are using the `eoxserver-instance.py` script with the `--init-spatialite` flag, all you have to do is:

- Make sure EOxServer is on your `PYTHONPATH` environment variable
- run in your instance directory:

```
python manage.py syncdb
```

This script will also create an administration user if you want to. Note the username and password you provide. You'll need it to log in to the admin client.

You can always create a user at a later time by running `python manage.py createsuperuser`.

⁴¹ https://github.com/EOxServer/eoxserver/blob/0.4/eoxserver/instance_template/project_name/settings.py

⁴² https://github.com/EOxServer/eoxserver/blob/0.4/eoxserver/instance_template/project_name/urls.py

⁴³ https://github.com/EOxServer/eoxserver/blob/0.4/eoxserver/instance_template/project_name/conf/eoxserver.conf

⁴⁴ <https://docs.djangoproject.com/en/1.4/topics/settings/>

⁴⁵ https://docs.djangoproject.com/en/1.4/ref/settings/#std:setting-TIME_ZONE

Database Setup

This section is only needed if the `--init_spatialite` flag was not used during instance creation or a PostgreSQL/PostGIS database back-end shall be used. Before proceeding, please make sure that you have installed all required software for the database system of your choice.

Using a SQLite database, all you have to do is to copy the `TEMPLATE_config.sqlite` and place it somewhere in your instance directory. Now you have to edit the `DATABASES` of your `settings.py` file with the following lines:

```
DATABASES = {
    'default': {
        'ENGINE': 'django.contrib.gis.db.backends.spatialite',
        'NAME': '/path/to/config.sqlite',
    }
}
```

Using a PostgreSQL/PostGIS database back-end configuration for EOxServer is a little bit more complex. Setting up a PostgreSQL database requires also installing the PostGIS extensions (the following example is an installation based on a Debian system):

```
sudo su - postgres
POSTGIS_DB_NAME=eoxserver_db
POSTGIS_SQL_PATH=`pg_config --sharedir`/contrib/postgis-1.5
createdb $POSTGIS_DB_NAME
createlang plpgsql $POSTGIS_DB_NAME
psql -d $POSTGIS_DB_NAME -f $POSTGIS_SQL_PATH/postgis.sql
psql -d $POSTGIS_DB_NAME -f $POSTGIS_SQL_PATH/spatial_ref_sys.sql
psql -d $POSTGIS_DB_NAME -c "GRANT ALL ON geometry_columns TO PUBLIC;"
psql -d $POSTGIS_DB_NAME -c "GRANT ALL ON geography_columns TO PUBLIC;"
psql -d $POSTGIS_DB_NAME -c "GRANT ALL ON spatial_ref_sys TO PUBLIC;"
```

This creates the database and installs the PostGIS extensions within the database. Now a user with password can be set with the following line:

```
createuser -d -R -P -S eoxserver-admin
```

Depending on the configuration of the system used there may be the need to enable access for the user in the `pg_hba.conf`.

In the `settings.py` the following entry has to be added:

```
DATABASES = {
    'default': {
        'ENGINE': 'django.contrib.gis.db.backends.postgis',
        'NAME': 'eoxserver_db',
        'USER': 'eoxserver-admin',
        'PASSWORD': 'eoxserver',
        'HOST': 'localhost', # or the URL of your server hosting the DB
        'PORT': '',
    }
}
```

Please refer to [GeoDjango Database API](https://docs.djangoproject.com/en/1.4/ref/contrib/gis/db-api/)⁴⁶ for more instructions.

Deployment

EOxServer is deployed using the Python WSGI interface standard as any other [Django application](https://docs.djangoproject.com/en/1.4/howto/deployment/)⁴⁷. The WSGI endpoint accepts HTTP requests passed from the web server and processes them synchronously. Each request is

⁴⁶ <https://docs.djangoproject.com/en/1.4/ref/contrib/gis/db-api/>

⁴⁷ <https://docs.djangoproject.com/en/1.4/howto/deployment/>

executed independently.

In the [deployment git repository](#)⁴⁸ we collect snippets for various deployment scenarios.

In the following we present the way to deploy it using the [Apache2 Web Server](#)⁴⁹ and its [mod_wsgi](#)⁵⁰ extension module.

The deployment procedure consists of the following:

- Customize the Apache2 configuration file, e.g. `/etc/apache2/sites-enabled/000-default`, by adding:

```
Alias /<url> <absolute path to instance dir>/wsgi.py
<Directory "<absolute path to instance dir>">
    AllowOverride None
    Options +ExecCGI -MultiViews +SymLinksIfOwnerMatch
    AddHandler wsgi-script .py
    Order Allow,Deny
    Allow from all
</Directory>
```

- If using EOxServer < 0.3 customize `wsgi.py` in your EOxServer instance and add:

```
import sys

path = "<absolute path to instance dir>"
if path not in sys.path:
    sys.path.append(path)
```

- Restart the Web Server

As a general good idea the number of threads can be limited using the following additional Apache2 configuration. In case an old version of MapServer, i.e. < 6.2 or < 6.0.4, is used the number of threads **needs** to be limited to 1 to avoid some [thread safety issues](#)⁵¹:

```
WSGIDaemonProcess ows processes=10 threads=1
<Directory "<absolute path to instance dir>">
    ...
    WSGIProcessGroup ows
</Directory>
```

This setup will deploy your instance under the URL `<url>` and make it publicly accessible.

Finally all the static files need to be collected at the location configured by `STATIC_ROOT` in `settings.py` by using the following command from within your instance:

```
python manage.py collectstatic
```

Don't forget to update the static files by re-running above command if needed.

Data Registration

To insert data into an EOxServer instance there are several ways. One is the admin interface, which is explained in detail in the [Admin Client](#) (page 62) section.

Another convenient way to register datasets is the command line interface to EOxServer. As a Django application, the instance can be configured using the `manage.py`⁵² script.

⁴⁸ <https://github.com/EOxServer/deployment>

⁴⁹ <http://httpd.apache.org>

⁵⁰ <http://code.google.com/p/modwsgi/>

⁵¹ <https://github.com/mapserver/mapserver/issues/4369>

⁵² <https://docs.djangoproject.com/en/1.4/ref/django-admin/>

EOxServer provides a specific command to insert datasets into the instance, called `eoxs_dataset_register`. It is invoked from command line from your instance base folder:

```
python manage.py eoxs_dataset_register --data DATAFILES --range-type RANGETYPE
```

The mandatory parameter `--data` is a path to a file containing the raster data for the dataset to be inserted. If the file resides in a package (a ZIP or TAR archive) then the location must be preceded with the following: `<package-type>:<package-location>`. It is also possible to chain multiple packages, e.g a ZIP file in a ZIP file containing the actual raster data. In conjunction to packages, it is also possible to state the storage of the data files. By default it is assumed that the data is available locally, but other storages (such as FTP or HTTP backends) are also possible. If used, it must be declared as first item in the aforementioned in the chain.

For each `--data` item a `--semantic` can be stated. The semantic defines how this data item is being used. For example a semantic of `"bands[1:3]"` defines that the first three bands of the dataset is in the first data item.

The same rules also apply for files declared via the `--meta-data` directive. This basically creates a `--data` item with `"metadata"` semantic. Also, these files are preferred when trying to determine the mandatory meta-data of a dataset.

To specify the Range Type of the dataset, the `--range-type` parameter is mandatory to specify the name of a previously registered Range Type.

The following options are used to supply metadata values that are either not possible to retrieve automatically or are to overwrite values automatically collected:

- `--identifier`: the main identifier of the dataset
- **`--extent`: the (minx,miny,maxx,maxy) bounding box of the dataset** expressed in the units defined in `--srid` or `--projection`
- `--size`: the pixel size of the dataset (`size_x,size_y`)
- `--srid` or `--projection`: the native projection of the dataset
- `--footprint`: the footprint (multi-) polygon in WKT format
- `--begin-time` and `--end-time`: the datasets time span
- `--coverage-type`: the type of the dataset

By default, a dataset is not advertised in WMS/WCS GetCapabilities. In order to enable this, use the `--visible` flag.

When this dataset shall be inserted into a collection, use the `--collection` option with the collections identifier. This option can be set multiple times for different collections.

Recommendations for Operational Installation

Table of Contents

- *Recommendations for Operational Installation* (page 25)
 - *Introduction EOxServer* (page 26)
 - *Directory Structure* (page 26)
 - *User Management* (page 27)
 - * *Operating System Users* (page 27)
 - * *Database User* (page 27)
 - * *Django Sysadmin* (page 28)
 - * *Application User Management* (page 28)

- *EOxServer Configuration Step-by-step* (page 28)
 - * *Step 1 - Web Server Installation* (page 28)
 - * *Step 2 - Database Backend* (page 29)
 - * *Step 3 - Creating Users and Directories for Instance and Data* (page 30)
 - * *Step 4 - Instance Creation* (page 30)
 - * *Step 5 - Database Setup* (page 30)
 - * *Step 6 - Web Server Integration* (page 32)
 - * *Step 7 - Start Operating the Instance* (page 33)

This section provides a set of recommendations and a step-by-step guide for the installation and configuration of EOxServer as an operational system. This guide goes beyond the basic installation presented in previous sections.

Unless stated otherwise this guide considers installing on CentOS GNU/Linux operating systems although the guide is applicable for other distributions as well.

We assume that the reader of this guide *knows* what the presented commands are doing and he/she understands the possible consequences. This guide is intended to help the administrator to setup the EOxServer quickly by extracting the salient information but the administrator must be able to alter the procedure to fit the particular needs of the administered system. We bear no responsibility for any possible harms caused by mindless following of this guide by a non-qualified person.

See also:

- *Installation* (page 15) generic installation procedure for GNU/Linux operating systems.
- *Installation on CentOS* (page 18) for specific installation on CentOS.
- *Service Instance Creation and Configuration* (page 21) to configure an instance of EOxServer after successful installation.

Introduction EOxServer

When installing and configuring EOxServer a clear distinction should be made between the common EOxServer installation (the installed code implementing the software functionality) and EOxServer instances. An instance is a collection of data and configuration files that enables the deployment of a specific service. A single server will typically contain a single software installation and one or more specific instances.

While the EOxServer installation is straightforward and typically does not require much effort (see the *generic* (page 15) and *CentOS* (page 18) installation guides) the *configuration* (page 21) requires more attention of the administrator and a bit of planning as well.

Closely related to EOxServer is the (possibly large) served EO data. It should be borne in mind, that EOxServer as such is not a data management system, i.e., it can register the stored data but does neither control nor require any specific data storage locations itself. Where and how the data is stored is thus in the responsibility of the administrator.

EOxServer registers the EO data and keeps only the essential metadata (data and full metadata location, geographic extent, acquisition time, etc.) in a database.

Directory Structure

First, the administrator has to decide in which directory each instance should be located. Each of the EOxServer instances is represented by a dedicated directory.

For system wide installation we recommend to create a single specific directory to hold all instances in one location compliant with the [filesystem hierarchy standard](#)⁵³:

⁵³ <http://www.pathname.com/fhs/pub/fhs-2.3.html#SRVDATAFORSERVICESPROVIDEDBYSYSTEM>

```
/srv/eoxserver
```

Optionally, for user defined instances a folder in the user's home directory is acceptable as well:

```
~/eoxserver
```

Note: We **strongly discourage** to keep the instance configuration in system locations not suited for this purpose such as `/root` or `/tmp`!

A dedicated directory should also be considered for the served EO data, e.g.:

```
/srv/eodata
```

or:

```
~/eodata
```

User Management

The EOxServer administrator has to deal with four different user management subsystems:

- system user (operating system),
- database user (SQL server),
- django user (Django user management), and
- application user (e.g., Single Sign On authentication).

Each of them is described hereafter.

Operating System Users

On a typical mutli-user operating system several users exist each of them owning some files and each of them is given some right to access other files and run executables.

In a typical EOxServer setup, the installed executables are owned by the `root` user and when executed they are granted the rights of the invoking process owner. When executed as a WGSi application, the running EOxServer executables run with the same ID as the web server (for Apache server this is typically the `apache` or `www-data` system user). This need to be considered when specifying access rights for the files which are expected to be changed or read by a running application.

The database back-end has usually its own dedicated system user (for PostgreSQL this is typically `postgres`).

Coming back, for EOxServer instances' configuration we recommend both instance and data to be owned by one or (preferably) two distinct system or ordinary users. These users can be existing (e.g., the `apache` user) or new dedicated users.

Note: We **strongly discourage** to keep the EOxService instances (i.e., configuration data) and the served EO data owned by the system administrator (`root`).

Database User

The Django framework (which EOxServer is build upon) requires access to a Database Management System (DBMS) which is typically protected by user-name/password based authentication. Specification of these DBMS credential is part of the service instance *configuration* (page 23).

The sole purpose of the DBMS credentials is to access the database.

It should be mentioned that user-name/password is not the only possible way how to secure the database access. The various authentication options for PostgreSQL are covered, e.g., [here](#)⁵⁴.

Django Sysadmin

The Django framework provides its own user management subsystem. EOxServer uses the Django user management system for granting access to the system administrator to the low level *Admin Web GUI*. (page 62). The Django user management is neither used to protect access to the provided Web Service interfaces nor to restrict access via the command line tools.

Application User Management

EOxServer is based on the assumption that the authentication and authorisation of an operational system would be performed by an external security system (such as the Shibboleth based *Single Sign On* (page 74) infrastructure). This access control would be transparent from EOxServer's point of view.

It is beyond the scope of this document to explain how to configure a Single Sign On (SSO) infrastructure but principally the configuration does not differ from securing plain apache web server.

EOxServer Configuration Step-by-step

The guidelines presented in this section assume a successful installation of EOxServer and of the essential dependencies performed either from the available RPM packages (see CentOS *Installation from RPM Packages* (page 19)) or via the Python Package Index (see *Alternate installation method using pip* (page 20)).

This guide assume that the `sudo`⁵⁵ command is installed and configured on the system.

In case of installation from RPM repositories it is necessary to install the required repositories first:

```
sudo rpm -Uvh http://elgis.argeo.org/repos/6/elgis-release-6-6_0.noarch.rpm
sudo yum install epel-release
sudo rpm -Uvh http://yum.packages.eox.at/el/eox-release-6-2.noarch.rpm
```

and then install EOxServer's package:

```
sudo yum install EOxServer
```

Step 1 - Web Server Installation

EOxServer is a Django based web application and as such it needs a web server (the simple Django provided server is not an option for an operational system). Any instance of EOxServer receives HTTP requests via the WSGI interface. EOxServer is tested to work with the Apache⁵⁶ web server using the WSGI⁵⁷ module. The server can be installed using:

```
sudo yum install httpd mod_wsgi
```

EOxServer itself is not equipped by any authentication or authorisation mechanism. In order to secure the resources an external tool must be used to control access to the resources (e.g., the Shibboleth Apache module or the Shibboleth based *Single Sign On* (page 74)).

To start the apache server automatically at the boot-time run following command:

⁵⁴ <http://www.postgresql.org/docs/dev/static/auth-pg-hba-conf.html>

⁵⁵ http://www.centos.org/docs/4/4.5/Security_Guide/s3-wstation-privileges-limitroot-sudo.html

⁵⁶ <http://www.apache.org/>

⁵⁷ http://en.wikipedia.org/wiki/Web_Server_Gateway_Interface

```
sudo chkconfig httpd on
```

The status of the web server can be checked by:

```
sudo service httpd status
```

and if not running the service can be started as follows:

```
sudo service httpd start
```

It is likely the ports offered by the web service are blocked by the firewall. To allow access to port 80 used by the web service it should be mostly sufficient to call:

```
sudo iptables -I INPUT -m state --state NEW -m tcp -p tcp --dport 80 -j ACCEPT
```

Setting up access to any other port than 80 (such as port 443 used by HTTPS) is the same, just change the port number in the previous command.

To make these **iptables** firewall settings permanent (preserved throughout reboots) run:

```
sudo service iptables save
```

Step 2 - Database Backend

EOxServer requires a Database Management System (DBMS) for the storage of its internal data. For an operational system a local or remote installation of PostgreSQL⁵⁸ with PostGIS⁵⁹ extension is recommended over the simple file-based SQLite backend. To install the DBMS run following command:

```
sudo yum install postgresql postgresql-server postgis python-psycopg2
```

PostgreSQL comes with reasonable default settings which are often sufficient. For details on more advanced configuration options (like changing the default database location) see, e.g., PostgreSQL's wiki⁶⁰

On some Linux distributions like recent RHEL and its clones such as CentOS, the PostgreSQL database must be initialized manually by:

```
sudo service postgresql initdb
```

To start the service automatically at boot time run:

```
sudo chkconfig postgresql on
```

You can check if the PostgreSQL database is running or not via:

```
sudo service postgresql status
```

If not start the PostgreSQL server:

```
sudo service postgresql start
```

Once the PostgreSQL daemon is running we have to setup a database template including the required PostGIS extension:

```
sudo -u postgres createdb template_postgis
sudo -u postgres createlang plpgsql template_postgis
PG_SHARE=/usr/share/pgsql
sudo -u postgres psql -q -d template_postgis -f $PG_SHARE/contrib/postgis.sql
```

⁵⁸ <http://www.postgresql.org/>

⁵⁹ <http://postgis.net/>

⁶⁰ http://wiki.postgresql.org/wiki/Main_Page

```
sudo -u postgres psql -q -d template_postgis -f $PG_SHARE/contrib/spatial_ref_sys.
↪sql
psql -d postgres psql -q -d template_postgis -c "GRANT ALL ON geometry_columns TO_
↪PUBLIC;"
psql -d postgres psql -q -d template_postgis -c "GRANT ALL ON geography_columns TO_
↪PUBLIC;"
psql -d postgres psql -q -d template_postgis -c "GRANT ALL ON spatial_ref_sys TO_
↪PUBLIC;"
```

Please note that the `PG_SHARE` directory can vary for each Linux distribution or custom PostgreSQL installation. For CentOS `/usr/share/pgsql` happens to be the default location. The proper path can be found, e.g., by:

```
locate contrib/postgis.sql
```

Step 3 - Creating Users and Directories for Instance and Data

To create the users and directories for the EOxServer instances and the served EO Data run the following commands:

```
sudo useradd -r -m -g apache -d /srv/eoxserver -c "EOxServer's administrator" _
↪eoxserver
sudo useradd -r -m -g apache -d /srv/eodata -c "EO data provider" eodata
```

For meaning of the used options see documentation of `useradd`⁶¹ command.

Since we are going to access the files through the Apache web server, for convenience, we set the default group to `apache`. In addition, to make the directories readable by other users run the following commands:

```
sudo chmod o+=rx /srv/eoxserver
sudo chmod o+=rx /srv/eodata
```

Step 4 - Instance Creation

Now it's time to setup a sample instance of EOxServer. Create a new instance e.g., named `instance00`, using the `eoxserver-instance.py` command:

```
sudo -u eoxserver mkdir /srv/eoxserver/instance00
sudo -u eoxserver eoxserver-instance.py instance00 /srv/eoxserver/instance00
```

Now our first bare instance exists and needs to be configured.

Step 5 - Database Setup

As the first to animate the instance it is necessary to setup a database. Assuming the PostgreSQL DBMS is up and running, we start by creating a database user (replace `<db_username>` by a user-name of your own choice):

```
sudo -u postgres createuser --no-createdb --no-superuser --no-createrole --
↪encrypted --password <db_username>
```

The user's password is requested interactively. Once we have the database user we can create the database for our instance:

```
sudo -u postgres createdb --owner <db_username> --template template_postgis --
↪encoding UTF-8 eoxs_instance00
```

⁶¹ <http://unixhelp.ed.ac.uk/CGI/man-cgi?useradd+8>

Where `eoxs_instance00` is the name of the new database. As there may be more EOxServer instances, each of them having its own database, it is a good practice to set a DB name containing the name of the instance.

In addition the PostgreSQL access policy must be set to allow access to the newly created database. To get access to the database, insert the following lines (replace `<db_username>` by your actual DB user-name):

```
local eoxs_instance00 <db_username> md5
```

to the file:

```
/var/lib/pgsql/data/pg_hba.conf
```

Note: This allows *local* database access only.

When inserting the line make sure you put this line **before** the default access policy:

```
local all all ident
```

In case of an SQL server running on a separate machine please see PostgreSQL documentation⁶².

The location of the `pg_hba.conf` file varies from one system to another. In case of troubles to locate this file try, e.g.:

```
sudo locate pg_hba.conf
```

Once we created and configured the database we need to update the EOxServer settings stored, in our case, in file:

```
/srv/eoxserver/instance00/instance00/settings.py
```

Make sure the database is configured in `settings.py` as follows:

```
DATABASES = {
    'default': {
        'ENGINE': 'django.contrib.gis.db.backends.postgis',
        'NAME': 'eoxs_instance00',
        'USER': '<db_username>',
        'PASSWORD': '<bd_password>',
        'HOST': '', # keep empty for local DBMS
        'PORT': '', # keep empty for local DBMS
    }
}
```

As in our previous examples replace `<db_username>` and `<bd_password>` by the proper database user's name and password.

Finally it is time to initialize the database of your first instance by running the following command:

```
sudo -u eoxserver python /srv/eoxserver/instance00/manage.py syncdb
```

The command interactively asks for the creation of the Django system administrator. It is safe to say no and create the administrator's account later by:

```
sudo -u eoxserver python /srv/eoxserver/instance00/manage.py createsuperuser
```

The `manage.py` is the command-line proxy for the management of EOxServer. To avoid repeated writing of this fairly long command make a shorter alias such as:

```
alias eoxsi00="sudo -u eoxserver python /srv/eoxserver/instance00/manage.py"
eoxsi00 createsuperuser
```

⁶² <http://www.postgresql.org/docs/devel/static/auth-pg-hba-conf.html>

Step 6 - Web Server Integration

The remaining task to be performed is to integrate the created EOxServer instance with the Apache web server. As it was already mentioned, the web server access the EOxServer instance through the WSGI interface. We assume that the web server is already configured to load the `mod_wsgi` module and thus it remains to configure the WSGI access point. The proposed configuration is to create the new configuration file `/etc/httpd/conf.d/default_site.conf` with the following content:

```
<VirtualHost *:80>
  # EOxServer instance: instance00
  Alias /instance00 "/srv/eoxserver/instance00/instance00/wsgi.py"
  Alias /instance00_static "/srv/eoxserver/instance00/instance00/static"
  WSGIDaemonProcess ows processes=10 threads=1
  <Directory "/srv/eoxserver/instance00/instance00">
    Options +ExecCGI FollowSymLinks
    AddHandler wsgi-script .py
    WSGIProcessGroup ows
    AllowOverride None
    Order allow,deny
    allow from all
  </Directory>
</VirtualHost>
```

In case there is already a `VirtualHost` section present in `/etc/httpd/conf/httpd.conf` or in any other `*.conf` file included from the `/etc/httpd/conf.d/` directory we suggest to add the configuration lines given above to the appropriate virtual host section.

The `WSGIDaemonProcess` option forces execution of the Apache WSGI in daemon mode using multiple single-thread processes. While the number of daemon processes can be adjusted the number of threads *must* be always set to 1.

On systems such as CentOS, following option must be added to Apache configuration (preferably in `/etc/httpd/conf.d/wsgi.conf`) to allow communication between the Apache server and WSGI daemon (the reason is explained, e.g., [here](#)⁶³):

```
WSGISocketPrefix run/wsgi
```

Don't forget to adjust the URL configuration in `/srv/eoxserver/instance00/instance00/conf/eoxserver.conf`:

```
[services.owscommon]
http_service_url=http://<you-server-address>/instance00/ows
```

The location and base URL of the static files are specified in the EOxServer instance's `setting.py` file by the `STATIC_ROOT` and `STATIC_URL` options:

```
...
STATIC_ROOT = '/srv/eoxserver/instance00/instance00/static/'
...
STATIC_URL = '/instance00_static/'
...
```

These options are set automatically by the instance creation script.

The static files needed by the EOxServer's web GUI need to be initialized (*collected*) using the following command:

```
alias eoxsi00="sudo -u eoxserver python /srv/eoxserver/instance00/manage.py"
eoxsi00 collectstatic -l
```

To allow the apache user to write to the instance log-file make sure the user is permitted to do so:

⁶³ <http://code.google.com/p/modwsgi/wiki/ConfigurationIssues>

```
sudo chmod g+w /srv/eoxserver/instance00/instance00/logs/eoxserver.log
```

And now the last thing to do remains to restart the Apache server by:

```
sudo service httpd restart
```

You can check that your EOxServer instance runs properly by inserting the following URL to your browser:

```
http://<you-server-address>/instance00
```

Step 7 - Start Operating the Instance

Now we have a running instance of EOxServer. For different operations such as data registration see *EOxServer Operators' Guide* (page 56).

Migration

Table of Contents

- *Migration* (page 33)
 - *Migration from 0.3 to 0.4* (page 33)
 - *Migration from 0.2 to 0.3* (page 33)
 - * *Disclaimer* (page 34)
 - * *Preparatory steps* (page 34)
 - * *Software upgrade* (page 34)
 - *Django & GDAL* (page 34)
 - *EOxServer* (page 34)
 - * *Instance migration* (page 35)
 - * *New configuration options* (page 35)

Migrating or upgrading an existing EOxServer instance may require to perform several tasks depending on the version numbers. In general upgrading versions with changes in the third digit of the version number only e.g. from 0.2.3 to 0.2.4 doesn't need any special considerations. For all other upgrades please carefully read the relevant sections below.

Migration from 0.3 to 0.4

Unfortunately there are no migrations from version 0.3 to version 0.4 due to a major overhaul of the database schema and configuration. We recommend that you upgrade the EOxServer software, create a new instance and register all the data again.

Migration from 0.2 to 0.3

From version 0.2 to version 0.3 a lot of development effort has been put into EOxServer. Many new features have been implemented and a couple of bugs are now eradicated.

However, if you already have an instance running EOxServer 0.2, this requires a couple of changes to that instance and enables you to configure some new optional configurations aswell.

Disclaimer

Before trying to upgrade EOxServer please make sure to backup your database. This step depends on the actual DBMS you are using for your instance.

Note: If you do not have a lot of datasets registered, or can easily reproduce the current status of your instance, a complete newly created instance may be more failsafe than trying to migrate your instance.

Warning: Because of changes in the database schema, the migration of referenceable datasets does **not** work. Please re-register them once the instance is migrated/re-created.

Preparatory steps

Before you upgrade your software, you will need to perform a database dump. The dump is required to migrate your registered objects to the new database. It is performed with the following call:

```
python manage.py dumpdata core backends coverages --indent=4 > dump.json
```

Unfortunately in some versions `spatialite` produces some output aswell, which has to be removed from the top of the created `dump.json` file.

Software upgrade

Now you are ready to actually perform the software upgrade.

Django & GDAL

The most notable changes concern our technology base: Django & GDAL. EOxServer now relies on features of Django 1.4, so if you still have Django 1.3 or lower installed, please upgrade to (at least) that version. This step, however, depends on how you installed Django in the first place. With `pip` it should be easy as `pie/py`:

```
pip install Django --upgrade
```

If EOxServer is installed via `pip`, the upgrade of Django should be done automatically.

Similar to Django, EOxServer now requires at least version 1.7 of the GDAL library respectively its python bindings. GDAL is not explicitly stated in the EOxServer dependencies to allow custom builds and OS specific installations. So you are required to install the minimum required version on your own, via `pip`, `yum`, `apt`, `msi` or whatever mechanism you prefer.

Please refer to the *EOxServer Dependencies* (page 16) table for details on dependencies.

EOxServer

The upgrade of EOxServer is quite similar to *Installing EOxServer* (page 17). For `pip` you will need the `-U` (`--upgrade`) option:

```
pip install -U EOxServer==0.3
```

or

```
pip install -U "svn+http://eoxserver.org/svn/branches/0.3"
```

Instance migration

Now that you have installed your software, there is a small step to perform which requires manual handling to upgrade your instance to the new version of EOxServer.

Please open the `conf/eoxserver.conf` file within your instance directory and locate the modules setting of the `[core.registry]` setting. The list entry `eoxserver.resources.coverages.covmgrs` must be corrected to `eoxserver.resources.coverages.managers`.

Now it is time to re-create your database which is done in three steps: deletion of the old database, creation of a new one, and a synchronization. The deletion and creation of the database depend on the database backend used. For SQLite, for example, only the database file needs to be deleted.

The initialization of the database is done via:

```
python manage.py syncdb
```

The old contents of the database can be restored via:

```
python manage.py loaddata dump.json
```

New configuration options

Since version 0.2 a couple of new configuration options are available, most notably for defining output *formats* (page 111) and *CRSs* (page 110). Please have a look at the relevant sections to see how both are set up.

With Django 1.4, EOxServer allows a much more fine-grained logging mechanism defined in `settings.py`. Details can be obtained from the [Django documentation](#)⁶⁴. The following is an example of how the logging is set up by default in new EOxServer instances using version 0.3:

```
LOGGING = {
    'version': 1,
    'disable_existing_loggers': True,
    'filters': {
        'require_debug_false': {
            '()': 'django.utils.log.RequireDebugFalse'
        }
    },
    'formatters': {
        'simple': {
            'format': '%(levelname)s: %(message)s'
        },
        'verbose': {
            'format': '[%(asctime)s] [%(module)s] %(levelname)s: %(message)s'
        }
    },
    'handlers': {
        'eoxserver_file': {
            'level': 'DEBUG',
            'class': 'logging.FileHandler',
            'filename': join(PROJECT_DIR, 'logs', 'eoxserver.log'),
            'formatter': 'verbose',
            'filters': [],
        }
    },
    'loggers': {
        'eoxserver': {
            'handlers': ['eoxserver_file'],
            'level': 'DEBUG' if DEBUG else 'INFO',
            'propagate': False,
        }
    }
}
```

⁶⁴ <https://docs.djangoproject.com/en/dev/topics/logging/#configuring-logging>

```
    },  
    }  
}
```

Another important feature that was introduced in Django 1.4 is the implicit support of time-zones. This can be activated in `settings.py`:

```
USE_TZ = True
```

For a complete list of changes in Django see the official documentation ([1.4⁶⁵](#) and [1.5⁶⁶](#)).

Mailing Lists

Table of Contents

- *Mailing Lists* (page 36)
 - *Users Mailing List* (page 36)
 - *Dev Mailing List* (page 36)

Users Mailing List

The users mailing list is the primary means for EOxServer users and developers to exchange and discuss ideas and potential software improvements, and to ask questions.

Subscribe at <http://eoxserver.org/mailman/listinfo/users/>. You can later change your subscription information or unsubscribe from the list at this website too.

Here are some points to remember when posting to the list:

- Search the archive at <http://eoxserver.org/pipermail/users/> or <http://eoxserver.2316974.n4.nabble.com/EOxServer-Users-f4264995.html> for your answer first, people get tired of answering the same questions over and over.
- Before posting subscribe to the list by following the procedure described above.
- Post questions to the list by sending an email message to users@eoxserver.org.
- Provide version and configuration information for your EOxServer installation, like relevant snippets of your configuration files.
- Always post your responses back to the whole list, as opposed to just the person who replied to your question.
- Questions to the list are usually answered quickly and often by the developers themselves.

Dev Mailing List

A separate mailing list is available for EOxServer developers. It is meant to be used by individuals working on EOxServer source code and related libraries to discuss issues that would not be of interest to the entire users mailing list.

Subscribe at <http://eoxserver.org/mailman/listinfo/dev/>. You can later change your subscription information or unsubscribe from the list at this website too.

⁶⁵ <https://docs.djangoproject.com/en/dev/releases/1.4/>

⁶⁶ <https://docs.djangoproject.com/en/dev/releases/1.5/>

The archive is located at <http://eoxserver.org/pipermail/dev/> or <http://eoxserver.2316974.n4.nabble.com/EOxServer-Dev-f4265142.html>.

Demonstration

Table of Contents

- *Demonstration* (page 37)
 - *GetCapabilities* (page 37)
 - *DescribeCoverage* (page 38)
 - *DescribeEOCoverageSet* (page 39)
 - * *Dataset* (page 39)
 - * *StitchedMosaic* (page 39)
 - * *DatasetSeries* (page 40)
 - *GetCoverage* (page 41)
 - *GetCoverage POST/XML* (page 43)

The EOxServer demonstration is an instantiation of the *autotest instance* (page 129) and is based on the Envisat MERIS sample data available [here](#)⁶⁷.

The configuration includes one DatasetSeries and one StitchedMosaic both combining the three available datasets:

- DatasetSeries (EOId: MER_FRS_1P_reduced) containing the 3 MERIS sample datasets with all 15 radiance bands encoded as uint16 values
- StitchedMosaic (CoverageId: mosaic_MER_FRS_1P_reduced_RGB) containing the 3 MERIS sample datasets reduced to RGB 8-bit

Note, the data has been reduced from 300m resolution to 3000m.

The demonstration tries to show the usage of all available *EO-WCS request parameters* (page 45).

GetCapabilities

GetCapabilities⁶⁸:

```
http://eoxserver.org/demo_stable/ows?
service=wcs&
version=2.0.1&
request=GetCapabilities
```

Interesting parts of the response:

- Advertising EO-WCS:

```
<ows:Profile>http://www.opengis.net/spec/WCS_application-profile_earth-
->observation/1.0/conf/eowcs</ows:Profile>
```

- The additional EO-WCS operation:

⁶⁷ <http://earth.esa.int/object/index.cfm?fobjectid=4320>

⁶⁸ http://eoxserver.org/demo_stable/ows?service=wcs&version=2.0.1&request=GetCapabilities

```

<ows:Operation name="DescribeEOCoverageSet">
  <ows:DCP>
    <ows:HTTP>
      <ows:Get xlink:href="http://eoxserver.org/demo_stable/ows?"
↳xlink:type="simple"/>
      <ows:Post xlink:href="http://eoxserver.org/demo_stable/ows?"
↳xlink:type="simple">
        <ows:Constraint name="PostEncoding">
          <ows:AllowedValues>
            <ows:Value>XML</ows:Value>
          </ows:AllowedValues>
        </ows:Constraint>
      </ows:Post>
    </ows:HTTP>
  </ows:DCP>
</ows:Operation>

```

- The server will limit the number of CoverageDescription elements in DescribeEOCoverageSet responses:

```

<ows:Constraint name="CountDefault">
  <ows:NoValues/>
  <ows:DefaultValue>100</ows:DefaultValue>
</ows:Constraint>

```

- There is a StitchedMosaic available:

```

<wcs:CoverageSummary>
  <wcs:CoverageId>mosaic_MER_FRS_1P_reduced_RGB</wcs:CoverageId>
  <wcs:CoverageSubtype>RectifiedStitchedMosaic</wcs:CoverageSubtype>
</wcs:CoverageSummary>

```

- There is a DatasetSeries available:

```

<wcseo:DatasetSeriesSummary>
  <ows:WGS84BoundingBox>
    <ows:LowerCorner>-3.43798100 32.26454100</ows:LowerCorner>
    <ows:UpperCorner>27.96859100 46.21844500</ows:UpperCorner>
  </ows:WGS84BoundingBox>
  <wcseo:DatasetSeriesId>MER_FRS_1P_reduced</wcseo:DatasetSeriesId>
  <gml:TimePeriod gml:id="MER_FRS_1P_reduced_timeperiod">
    <gml:beginPosition>2006-08-16T09:09:29</gml:beginPosition>
    <gml:endPosition>2006-08-30T10:13:06</gml:endPosition>
  </gml:TimePeriod>
</wcseo:DatasetSeriesSummary>

```

DescribeCoverage

DescribeCoverage StitchedMosaic⁶⁹:

```

http://eoxserver.org/demo_stable/ows?
service=wcs&
version=2.0.1&
request=DescribeCoverage&
coverageid=mosaic_MER_FRS_1P_reduced_RGB

```

DescribeCoverage Dataset⁷⁰:

⁶⁹ http://eoxserver.org/demo_stable/ows?service=wcs&version=2.0.1&request=DescribeCoverage&coverageid=mosaic_MER_FRS_1P_reduced_RGB

⁷⁰ http://eoxserver.org/demo_stable/ows?service=wcs&version=2.0.1&request=DescribeCoverage&coverageid=MER_FRS_1PNPDE20060822_092058_000001972050_00308_23408_0077_uint16_reduced_compressed

```

http://eoxserver.org/demo_stable/ows?
  service=wcs&
  version=2.0.1&
  request=DescribeCoverage&
  coverageid=MER_FRS_1PNPDE20060822_092058_000001972050_00308_23408_0077_uint16_
  ↪reduced_compressed

```

DescribeEOCoverageSet

Dataset

DescribeEOCoverageSet Dataset⁷¹:

```

http://eoxserver.org/demo_stable/ows?
  service=wcs&
  version=2.0.1&
  request=DescribeEOCoverageSet&
  EOId=MER_FRS_1PNPDE20060822_092058_000001972050_00308_23408_0077_uint16_
  ↪reduced_compressed

```

StitchedMosaic

DescribeEOCoverageSet StitchedMosaic (4 Datasets returned)⁷²:

```

http://eoxserver.org/demo_stable/ows?
  service=wcs&
  version=2.0.1&
  request=DescribeEOCoverageSet&
  EOId=mosaic_MER_FRS_1P_reduced_RGB

```

DescribeEOCoverageSet StitchedMosaic, subset in time (3 Datasets returned)⁷³:

```

http://eoxserver.org/demo_stable/ows?
  service=wcs&
  version=2.0.1&
  request=DescribeEOCoverageSet&
  EOId=mosaic_MER_FRS_1P_reduced_RGB&
  subset=phenomenonTime("2006-08-01", "2006-08-22T09:22:00Z")

```

DescribeEOCoverageSet StitchedMosaic, subset in Lat and Long, containment contains (1 Dataset returned)⁷⁴:

```

http://eoxserver.org/demo_stable/ows?
  service=wcs&
  version=2.0.1&
  request=DescribeEOCoverageSet&
  EOId=mosaic_MER_FRS_1P_reduced_RGB&
  subset=Lat(32,47)&
  subset=Long(11,33)&
  containment=contains

```

⁷¹ http://eoxserver.org/demo_stable/ows?service=wcs&version=2.0.1&request=DescribeEOCoverageSet&EOId=MER_FRS_1PNPDE20060822_092058_000001972050_00308_23408_0077_uint16_reduced_compressed

⁷² http://eoxserver.org/demo_stable/ows?service=wcs&version=2.0.1&request=DescribeEOCoverageSet&EOId=mosaic_MER_FRS_1P_reduced_RGB

⁷³ [http://eoxserver.org/demo_stable/ows?service=wcs&version=2.0.1&request=DescribeEOCoverageSet&EOId=mosaic_MER_FRS_1P_reduced_RGB&subset=phenomenonTime\(%222006-08-01%22,%222006-08-22T09:22:00Z%22\)](http://eoxserver.org/demo_stable/ows?service=wcs&version=2.0.1&request=DescribeEOCoverageSet&EOId=mosaic_MER_FRS_1P_reduced_RGB&subset=phenomenonTime(%222006-08-01%22,%222006-08-22T09:22:00Z%22))

⁷⁴ [http://eoxserver.org/demo_stable/ows?service=wcs&version=2.0.1&request=DescribeEOCoverageSet&EOId=mosaic_MER_FRS_1P_reduced_RGB&subset=Lat\(32,47\)&subset=Long\(11,33\)&containment=contains](http://eoxserver.org/demo_stable/ows?service=wcs&version=2.0.1&request=DescribeEOCoverageSet&EOId=mosaic_MER_FRS_1P_reduced_RGB&subset=Lat(32,47)&subset=Long(11,33)&containment=contains)

DescribeEOCoverageSet StitchedMosaic, returned CoverageDescriptions limited to 2⁷⁵:

```
http://eoxserver.org/demo_stable/ows?
  service=wcs&
  version=2.0.1&
  request=DescribeEOCoverageSet&
  EOId=mosaic_MER_FRS_1P_reduced_RGB&
  count=2
```

DatasetSeries

DescribeEOCoverageSet DatasetSeries (5 Datasets returned)⁷⁶:

```
http://eoxserver.org/demo_stable/ows?
  service=wcs&
  version=2.0.1&
  request=describeecoverageset&
  eoid=MER_FRS_1P_reduced
```

DescribeEOCoverageSet DatasetSeries, trim subset in time (4 Datasets returned)⁷⁷:

```
http://eoxserver.org/demo_stable/ows?
  service=wcs&
  version=2.0.1&
  request=describeecoverageset&
  eoid=MER_FRS_1P_reduced&
  subset=phenomenonTime("2006-08-01", "2006-08-22T09:22:00Z")
```

DescribeEOCoverageSet DatasetSeries, slice subset in time (2 Dataset returned)⁷⁸:

```
http://eoxserver.org/demo_stable/ows?
  service=wcs&
  version=2.0.1&
  request=describeecoverageset&
  eoid=MER_FRS_1P_reduced&
  subset=phenomenonTime("2006-08-22T09:20:58Z")
```

DescribeEOCoverageSet DatasetSeries, trim subset in time trim, containment contains (2 Dataset returned)⁷⁹:

```
http://eoxserver.org/demo_stable/ows?
  service=wcs&
  version=2.0.1&
  request=describeecoverageset&
  eoid=MER_FRS_1P_reduced&
  subset=phenomenonTime("2006-08-01", "2006-08-22T09:22:00Z") &
  containment=contains
```

DescribeEOCoverageSet DatasetSeries, subset in Lat and Long (5 Datasets returned)⁸⁰:

```
http://eoxserver.org/demo_stable/ows?
  service=wcs&
```

⁷⁵ http://eoxserver.org/demo_stable/ows?service=wcs&version=2.0.1&request=DescribeEOCoverageSet&EOId=mosaic_MER_FRS_1P_reduced_RGB&count=2

⁷⁶ http://eoxserver.org/demo_stable/ows?service=wcs&version=2.0.1&request=describeecoverageset&eoid=MER_FRS_1P_reduced

⁷⁷ [http://eoxserver.org/demo_stable/ows?service=wcs&version=2.0.1&request=describeecoverageset&eoid=MER_FRS_1P_reduced&subset=phenomenonTime\(%222006-08-01%22,%222006-08-22T09:22:00Z%22\)](http://eoxserver.org/demo_stable/ows?service=wcs&version=2.0.1&request=describeecoverageset&eoid=MER_FRS_1P_reduced&subset=phenomenonTime(%222006-08-01%22,%222006-08-22T09:22:00Z%22))

⁷⁸ [http://eoxserver.org/demo_stable/ows?service=wcs&version=2.0.1&request=describeecoverageset&eoid=MER_FRS_1P_reduced&subset=phenomenonTime\(%222006-08-22T09:20:58Z%22\)](http://eoxserver.org/demo_stable/ows?service=wcs&version=2.0.1&request=describeecoverageset&eoid=MER_FRS_1P_reduced&subset=phenomenonTime(%222006-08-22T09:20:58Z%22))

⁷⁹ [http://eoxserver.org/demo_stable/ows?service=wcs&version=2.0.1&request=describeecoverageset&eoid=MER_FRS_1P_reduced&subset=phenomenonTime\(%222006-08-01%22,%222006-08-22T09:22:00Z%22\)&containment=contains](http://eoxserver.org/demo_stable/ows?service=wcs&version=2.0.1&request=describeecoverageset&eoid=MER_FRS_1P_reduced&subset=phenomenonTime(%222006-08-01%22,%222006-08-22T09:22:00Z%22)&containment=contains)

⁸⁰ [http://eoxserver.org/demo_stable/ows?service=wcs&version=2.0.1&request=describeecoverageset&eoid=MER_FRS_1P_reduced&subset=Lat\(32,47\)&subset=Long\(11,33\)](http://eoxserver.org/demo_stable/ows?service=wcs&version=2.0.1&request=describeecoverageset&eoid=MER_FRS_1P_reduced&subset=Lat(32,47)&subset=Long(11,33))

```

version=2.0.1&
request=describeecoverageset&
eoid=MER_FRS_1P_reduced&
subset=Lat(32,47)&
subset=Long(11,33)

```

DescribeEOCoverageSet DatasetSeries, subset in Lat and Long, containment contains (2 Dataset returned)⁸¹:

```

http://eoxserver.org/demo_stable/ows?
service=wcs&
version=2.0.1&
request=describeecoverageset&
eoid=MER_FRS_1P_reduced&
subset=Lat(32,47)&
subset=Long(11,33)&
containment=contains

```

GetCoverage

GetCoverage StitchedMosaic, full (GML incl. contributingFootprint & GeoTIFF)⁸²:

```

http://eoxserver.org/demo_stable/ows?
service=wcs&
version=2.0.1&
request=GetCoverage&
coverageid=mosaic_MER_FRS_1P_reduced_RGB&
format=image/tiff&
mediatype=multipart/mixed

```

GetCoverage Dataset, full (GML & GeoTIFF)⁸³:

```

http://eoxserver.org/demo_stable/ows?
service=wcs&
version=2.0.1&
request=GetCoverage&
coverageid=MER_FRS_1PNPDE20060822_092058_000001972050_00308_23408_0077_uint16_
↪reduced_compressed&
format=image/tiff&
mediatype=multipart/mixed

```

GetCoverage Dataset, subset in pixels⁸⁴:

```

http://eoxserver.org/demo_stable/ows?
service=wcs&
version=2.0.1&
request=GetCoverage&
coverageid=MER_FRS_1PNPDE20060822_092058_000001972050_00308_23408_0077_uint16_
↪reduced_compressed&
format=image/tiff&
mediatype=multipart/mixed&

```

⁸¹ [http://eoxserver.org/demo_stable/ows?service=wcs&version=2.0.1&request=describeecoverageset&eoid=MER_FRS_1P_reduced&subset=Lat\(32,47\)&subset=Long\(11,33\)&containment=contains](http://eoxserver.org/demo_stable/ows?service=wcs&version=2.0.1&request=describeecoverageset&eoid=MER_FRS_1P_reduced&subset=Lat(32,47)&subset=Long(11,33)&containment=contains)

⁸² http://eoxserver.org/demo_stable/ows?service=wcs&version=2.0.1&request=GetCoverage&coverageid=mosaic_MER_FRS_1P_reduced_RGB&format=image/tiff&mediatype=multipart/mixed

⁸³ [http://eoxserver.org/demo_stable/ows?service=wcs&version=2.0.1&request=GetCoverage&coverageid=MER_FRS_1PNPDE20060822_092058_000001972050_00308_23408_0077_uint16_reduced_compressed&format=image/tiff&mediatype=multipart/mixed&scalesize=x\(200\),y\(200\)](http://eoxserver.org/demo_stable/ows?service=wcs&version=2.0.1&request=GetCoverage&coverageid=MER_FRS_1PNPDE20060822_092058_000001972050_00308_23408_0077_uint16_reduced_compressed&format=image/tiff&mediatype=multipart/mixed&scalesize=x(200),y(200))

⁸⁴ [http://eoxserver.org/demo_stable/ows?service=wcs&version=2.0.1&request=GetCoverage&coverageid=MER_FRS_1PNPDE20060822_092058_000001972050_00308_23408_0077_uint16_reduced_compressed&format=image/tiff&mediatype=multipart/mixed&subset=x\(100,200\)&subset=y\(300,400\)](http://eoxserver.org/demo_stable/ows?service=wcs&version=2.0.1&request=GetCoverage&coverageid=MER_FRS_1PNPDE20060822_092058_000001972050_00308_23408_0077_uint16_reduced_compressed&format=image/tiff&mediatype=multipart/mixed&subset=x(100,200)&subset=y(300,400))

```
subset=x(100,200)&
subset=y(300,400)
```

GetCoverage Dataset, subset in epsg 4326⁸⁵:

```
http://eoxserver.org/demo_stable/ows?
  service=wcs&
  version=2.0.1&
  request=GetCoverage&
  coverageid=MER_FRS_1PNPDE20060822_092058_000001972050_00308_23408_0077_uint16_
  ↔reduced_compressed&
  format=image/tiff&
  mediatype=multipart/mixed&
  subset=Lat(40,41)&
  subset=Long(17,18)&
  subsettingCrs=http://www.opengis.net/def/crs/EPSG/0/4326
```

GetCoverage Dataset, full, OutputCRS epsg 3035⁸⁶:

```
http://eoxserver.org/demo_stable/ows?
  service=wcs&
  version=2.0.1&
  request=GetCoverage&
  coverageid=MER_FRS_1PNPDE20060822_092058_000001972050_00308_23408_0077_uint16_
  ↔reduced_compressed&
  format=image/tiff&
  mediatype=multipart/mixed&
  outputCrs=http://www.opengis.net/def/crs/EPSG/0/3035
```

GetCoverage Dataset, full, size 200x200⁸⁷:

```
http://eoxserver.org/demo_stable/ows?
  service=wcs&
  version=2.0.1&
  request=GetCoverage&
  coverageid=MER_FRS_1PNPDE20060822_092058_000001972050_00308_23408_0077_uint16_
  ↔reduced_compressed&
  format=image/tiff&
  mediatype=multipart/mixed&
  scalesize=x(200),y(200)
```

GetCoverage Dataset, full, size 200x400⁸⁸:

```
http://eoxserver.org/demo_stable/ows?
  service=wcs&
  version=2.0.1&
  request=GetCoverage&
  coverageid=MER_FRS_1PNPDE20060822_092058_000001972050_00308_23408_0077_uint16_
  ↔reduced_compressed&
  format=image/tiff&
```

⁸⁵ [http://eoxserver.org/demo_stable/ows?service=wcs&version=2.0.1&request=GetCoverage&coverageid=MER_FRS_1PNPDE20060822_092058_000001972050_00308_23408_0077_uint16_reduced_compressed&format=image/tiff&mediatype=multipart/mixed&subset=Lat\(40,41\)&subset=Long\(17,18\)&subsettingCrs=http://www.opengis.net/def/crs/EPSG/0/4326](http://eoxserver.org/demo_stable/ows?service=wcs&version=2.0.1&request=GetCoverage&coverageid=MER_FRS_1PNPDE20060822_092058_000001972050_00308_23408_0077_uint16_reduced_compressed&format=image/tiff&mediatype=multipart/mixed&subset=Lat(40,41)&subset=Long(17,18)&subsettingCrs=http://www.opengis.net/def/crs/EPSG/0/4326)

⁸⁶ [http://eoxserver.org/demo_stable/ows?service=wcs&version=2.0.1&request=GetCoverage&coverageid=MER_FRS_1PNPDE20060822_092058_000001972050_00308_23408_0077_uint16_reduced_compressed&format=image/tiff&mediatype=multipart/mixed&OutputCrs=http://www.opengis.net/def/crs/EPSG/0/3035&scalesize=x\(200\),y\(200\)](http://eoxserver.org/demo_stable/ows?service=wcs&version=2.0.1&request=GetCoverage&coverageid=MER_FRS_1PNPDE20060822_092058_000001972050_00308_23408_0077_uint16_reduced_compressed&format=image/tiff&mediatype=multipart/mixed&OutputCrs=http://www.opengis.net/def/crs/EPSG/0/3035&scalesize=x(200),y(200))

⁸⁷ [http://eoxserver.org/demo_stable/ows?service=wcs&version=2.0.1&request=GetCoverage&coverageid=MER_FRS_1PNPDE20060822_092058_000001972050_00308_23408_0077_uint16_reduced_compressed&format=image/tiff&mediatype=multipart/mixed&scalesize=x\(200\),y\(200\)](http://eoxserver.org/demo_stable/ows?service=wcs&version=2.0.1&request=GetCoverage&coverageid=MER_FRS_1PNPDE20060822_092058_000001972050_00308_23408_0077_uint16_reduced_compressed&format=image/tiff&mediatype=multipart/mixed&scalesize=x(200),y(200))

⁸⁸ [http://eoxserver.org/demo_stable/ows?service=wcs&version=2.0.1&request=GetCoverage&coverageid=MER_FRS_1PNPDE20060822_092058_000001972050_00308_23408_0077_uint16_reduced_compressed&format=image/tiff&mediatype=multipart/mixed&scalesize=x\(200\),y\(400\)](http://eoxserver.org/demo_stable/ows?service=wcs&version=2.0.1&request=GetCoverage&coverageid=MER_FRS_1PNPDE20060822_092058_000001972050_00308_23408_0077_uint16_reduced_compressed&format=image/tiff&mediatype=multipart/mixed&scalesize=x(200),y(400))

```
mediatype=multipart/mixed&
scaleSize=x(200),y(400)
```

GetCoverage Dataset, subset in bands⁸⁹:

```
http://eoxserver.org/demo_stable/ows?
service=wcs&
version=2.0.1&
request=GetCoverage&
coverageid=MER_FRS_1PNPDE20060822_092058_000001972050_00308_23408_0077_uint16_
↳reduced_compressed&
format=image/tiff&
mediatype=multipart/mixed&
rangesubset=MERIS_radiance_01_uint16:MERIS_radiance_03_uint16
```

GetCoverage POST/XML

GetCoverage requests with POST/XML encoding might look like this:

A simple request:

```
<wcs:GetCoverage service="WCS" version="2.0.1"
  xmlns:wcs="http://www.opengis.net/wcs/2.0">
  <wcs:CoverageId>mosaic_MER_FRS_1PNPDE20060816_090929_000001972050_00222_
↳23322_0058_RGB_reduced</wcs:CoverageId>
  <wcs:format>image/tiff</wcs:format>
  <wcs:mediaType>multipart/related</wcs:mediaType>
</wcs:GetCoverage>
```

With a subset in pixel coordinates:

```
<wcs:GetCoverage service="WCS" version="2.0.1"
  xmlns:wcs="http://www.opengis.net/wcs/2.0">
  <wcs:CoverageId>mosaic_MER_FRS_1PNPDE20060816_090929_000001972050_00222_
↳23322_0058_RGB_reduced</wcs:CoverageId>
  <wcs:DimensionTrim>
    <wcs:Dimension>x</wcs:Dimension>
    <wcs:TrimLow>0</wcs:TrimLow>
    <wcs:TrimHigh>99</wcs:TrimHigh>
  </wcs:DimensionTrim>
  <wcs:DimensionTrim>
    <wcs:Dimension>y</wcs:Dimension>
    <wcs:TrimLow>0</wcs:TrimLow>
    <wcs:TrimHigh>99</wcs:TrimHigh>
  </wcs:DimensionTrim>
  <wcs:format>image/tiff</wcs:format>
  <wcs:mediaType>multipart/related</wcs:mediaType>
</wcs:GetCoverage>
```

With a subset in geographic coordinates with bilinear interpolation:

```
<wcs:GetCoverage service="WCS" version="2.0.1"
  xmlns:wcs="http://www.opengis.net/wcs/2.0"
  xmlns:int="http://www.opengis.net/wcs/interpolation/1.0"
  xmlns:crs="http://www.opengis.net/wcs/crs/1.0">
  <wcs:Extension>
    <crs:subsettingCrs>http://www.opengis.net/def/crs/EPSG/0/4326</
↳crs:subsettingCrs>
```

⁸⁹ http://eoxserver.org/demo_stable/ows?service=wcs&version=2.0.1&request=GetCoverage&coverageid=MER_FRS_1PNPDE20060822_092058_000001972050_00308_23408_0077_uint16_reduced_compressed&format=image/tiff&mediatype=multipart/mixed&rangesubset=MERIS_radiance_01_uint16:MERIS_radiance_03_uint16

```

    <int:Interpolation>
      <int:globalInterpolation>http://www.opengis.net/def/interpolation/
      ↪OGC/1/bilinear</int:globalInterpolation>
    </int:Interpolation>
  </wgs:Extension>
  <wgs:CoverageId>mosaic_MER_FRS_1PNPDE20060816_090929_000001972050_00222_
  ↪23322_0058_RGB_reduced</wgs:CoverageId>
  <wgs:DimensionTrim>
    <wgs:Dimension>Long</wgs:Dimension>
    <wgs:TrimLow>20</wgs:TrimLow>
    <wgs:TrimHigh>22</wgs:TrimHigh>
  </wgs:DimensionTrim>
  <wgs:DimensionTrim>
    <wgs:Dimension>Lat</wgs:Dimension>
    <wgs:TrimLow>36</wgs:TrimLow>
    <wgs:TrimHigh>38</wgs:TrimHigh>
  </wgs:DimensionTrim>
  <wgs:format>image/tiff</wgs:format>
  <wgs:mediaType>multipart/related</wgs:mediaType>
</wgs:GetCoverage>

```

With a range-subset and pixel-subset:

```

<wgs:GetCoverage service="WCS" version="2.0.1"
  xmlns:wgs="http://www.opengis.net/wcs/2.0"
  xmlns:rsub="http://www.opengis.net/wcs/range-subsetting/1.0">
  <wgs:Extension>
    <rsub:RangeSubset>
      <rsub:RangeItem>
        <rsub:RangeComponent>MERIS_radiance_04_uint16</
  ↪rsub:RangeComponent>
      </rsub:RangeItem>
      <rsub:RangeItem>
        <rsub:RangeInterval>
          <rsub:startComponent>MERIS_radiance_05_uint16</
  ↪rsub:startComponent>
          <rsub:endComponent>MERIS_radiance_07_uint16</rsub:endComponent>
        </rsub:RangeInterval>
      </rsub:RangeItem>
    </rsub:RangeSubset>
  </wgs:Extension>
  <wgs:CoverageId>MER_FRS_1PNPDE20060822_092058_000001972050_00308_23408_
  ↪0077_uint16_reduced_compressed</wgs:CoverageId>
  <wgs:DimensionTrim>
    <wgs:Dimension>x</wgs:Dimension>
    <wgs:TrimLow>0</wgs:TrimLow>
    <wgs:TrimHigh>99</wgs:TrimHigh>
  </wgs:DimensionTrim>
  <wgs:DimensionTrim>
    <wgs:Dimension>y</wgs:Dimension>
    <wgs:TrimLow>0</wgs:TrimLow>
    <wgs:TrimHigh>99</wgs:TrimHigh>
  </wgs:DimensionTrim>
  <wgs:format>image/tiff</wgs:format>
  <wgs:mediaType>multipart/related</wgs:mediaType>
</wgs:GetCoverage>

```

With a set of GeoTIFF encoding parameters:

```

<wgs:GetCoverage service="WCS" version="2.0.1"
  xmlns:wgs="http://www.opengis.net/wcs/2.0"
  xmlns:geotiff="http://www.opengis.net/gmlcov/geotiff/1.0">
  <wgs:CoverageId>mosaic_MER_FRS_1PNPDE20060816_090929_000001972050_00222_
  ↪23322_0058_RGB_reduced</wgs:CoverageId>

```

```

<wcs:format>image/tiff</wcs:format>
<wcs:Extension>
  <geotiff:parameters>
    <geotiff:compression>Deflate</geotiff:compression>
    <geotiff:predictor>FloatingPoint</geotiff:predictor>
    <geotiff:interleave>Band</geotiff:interleave>
    <geotiff:tiling>true</geotiff:tiling>
    <geotiff:tilewidth>32</geotiff:tilewidth>
    <geotiff:tileheight>64</geotiff:tileheight>
  </geotiff:parameters>
</wcs:Extension>
</wcs:GetCoverage>

```

EO-WCS Request Parameters

Table of Contents

- *EO-WCS Request Parameters* (page 45)
 - *GetCapabilities* (page 45)
 - *DescribeCoverage* (page 46)
 - *DescribeEOCoverageSet* (page 47)
 - *GetCoverage* (page 48)

The following tables provide an overview over the available EO-WCS request parameters for each operation supported by EOxServer.

Please see EOxServer's *Demonstration* (page 37) for complete sample requests.

GetCapabilities

Table: “*EO-WCS GetCapabilities Request Parameters* (page 46)” below lists all parameters that are available with Capabilities requests.

Table 1.3: EO-WCS GetCapabilities Request Parameters

Parameter	Description / Subparameter	Allowed value(s) / Example	Mandatory (M) / Optional (O)
→ service	Requested service	WCS	M
→ request	Type of request	GetCapabilities	M
→ version ¹	Version number	2.0.1	O
→ acceptVersions ¹	Prioritized sequence of one or more specification versions accepted by the client, with preferred versions listed first (first supported version will be used) version1[,version2[,...]]	2.0.1, 1.1.2, 1.0.0	O
→ sections	Comma-separated unordered list of zero or more names of zero or more names of sections of service metadata document to be returned in service metadata document. Request only certain sections of Capabilities Document section1[,section2[,...]]	<ul style="list-style-type: none"> • DatasetSeriesSummary • CoverageSummary • Contents • All • ServiceIdentification • ServiceProvider • OperationsMetadata • Languages 	O
→ updateSequence	Date of last issued GetCapabilities request; to receive new document only if it has changed since	“2013-05-08”	O

DescribeCoverage

Table: “*EO-WCS DescribeCoverage Request Parameters* (page 46)” below lists all parameters that are available with DescribeCoverage requests.

Table 1.4: EO-WCS DescribeCoverage Request Parameters

Parameter	Description / Subparameter	Allowed value(s) / Example	Mandatory (M) / Optional (O)
→ service	Requested service	WCS	M
→ request	Type of request	DescribeCoverage	M
→ version ¹	Version number	2.0.1	M
→ coverageId	NCName(s): <ul style="list-style-type: none"> • valid coverageID of a Dataset • valid coverageID of a SticheMosaic 		M

¹ Version, acceptVersions: Support for EO-WCS is available only together with WCS version 2.0.1.

DescribeEOCoverageSet

Table: “EO-WCS DescribeEOCoverageSet Request Parameters (page 47)” below lists all parameters that are available with DescribeEOCoverageSet requests.

Table 1.5: EO-WCS DescribeEOCoverageSet Request Parameters

Parameter	Description / Subparameter	Allowed value(s) / Example	Mandatory (M) / Optional (O)
→ service	Requested service	WCS	M
→ request	Type of request	DescribeEOCoverageSet	M
→ version ¹	Version number	2.0.1	M
→ eoId	Valid eoId: <ul style="list-style-type: none"> • using the coverageId of a Dataset • using the eoId of a DatasetSeries • using the coverageId of a StitchedMosaic 		M
→ subset	Allows to constrain the request in each dimensions and define how these parameters are applied. The spatial constraint is expressed in WGS84, the temporal constraint in ISO 8601. Spatial trimming: Name of an coverage axis (Long or Lat) Temporal trimming: phenomenonTime Plus optional either: <ul style="list-style-type: none"> • containment = overlaps (default) • containment = contains Any combination thereof (but each value only once per request)	<ul style="list-style-type: none"> • Lat(32,47) • Long(11,33) • phenomenonTime(“2006-08-01”, “2006-08-22T09:22:00Z”) • Lat(32,47)&Long(11,33)&phenomenonTime(“2006-08-01”&“2006-08-22T09:22:00Z”)&containment=contains 	O
→ containment	see <i>subset</i> parameter	<ul style="list-style-type: none"> • overlaps (default) • contains 	O
→ section	see GetCapabilities	<ul style="list-style-type: none"> • DatasetSeriesSummary • CoverageSummary • All 	O
→ count	Limits the maximum number of DatasetDescriptions returned in the EOCoverageSetDescription.	10	O

GetCoverage

Table: “*EO-WCS GetCoverage Request Parameters* (page 49)” below lists all parameters that are available with GetCoverage requests.

Table 1.6: EO-WCS GetCoverage Request Parameters

Parameter	Description / Subparameter	Allowed value(s) / Example	Mandatory (M) / Optional (O)
→ service	Requested service	WCS	M
→ request	Type of request	GetCoverage	M
→ version ¹	Version number	2.0.1	M
→ coverageId	NCName(s): <ul style="list-style-type: none"> • valid coverageID of a Dataset • valid coverageID of a StickedMosaic 		M
→ format	Requested format of coverage to be returned, currently: <ul style="list-style-type: none"> • image/tiff • image/jpeg • image/png • image/gif 	image/tiff	M
→ mediatype	Coverage delivered directly as image file or enclosed in GML structure <ul style="list-style-type: none"> • not present or • multipart/mixed 	multipart/mixed	O
→ subset	Trimming of coverage dimension (no slicing allowed!) <ul style="list-style-type: none"> • the label of a coverage axis <ul style="list-style-type: none"> – The meaning of the subset can be altered by the subsettingCrs parameter. 	<ul style="list-style-type: none"> • x(400,200) • Lat(12,14) • Long(17,17.4) 	O
→ subsettingCrs	The CRS the subsets are expressed in. This also defines the output CRS, if no further outputCrs is specified. If no subsettingCrs is given, pixel coordinates are assumed.	http://www.opengis.net/def/crs/EPSG/0/4326	O
→ outputCrs	CRS for the requested output coverage <ul style="list-style-type: none"> • not present or • CRS 	http://www.opengis.net/def/crs/EPSG/0/3035	O
→ rangesubset	Subsetting in the range domain (e.g. Band-Subsetting).	<ul style="list-style-type: none"> • Blue,Green,Red • Band1:Band3,Band5,Band7:Band9 	O
→ scaleFactor	Scale the output by this factor. The 'scaleFactor' parameter requires MapServer v7.0.	<ul style="list-style-type: none"> • 0.5 • 1.25 	O
• → scaleAxes	Mutually exclusive per axis, either:	<ul style="list-style-type: none"> • scaleAxes=x(1.5),y(0.5) 	O

OpenSearch

Table of Contents

- *OpenSearch* (page 50)
 - *Introduction* (page 50)
 - *Setup* (page 50)
 - *Usage* (page 51)
 - * *Collection Search* (page 51)
 - * *Record Search* (page 52)
 - *Parameters* (page 53)
 - *Output Formats* (page 55)
 - * *ATOM and RSS* (page 55)
 - * *GeoJSON and KML* (page 55)
 - * *Enabling/Disabling Formats* (page 55)
 - *Future Work* (page 55)

Introduction

Since version 0.4, EOxServer features an OpenSearch 1.1 interface to allow the exploration of its contents in a different manner than by using the EO-WCS or WMS functionality.

In contrast to EO-WCS and WMS, the OpenSearch interface operates on metadata only and allows a performant view of the data, by using slimmer output formats such as GeoJSON or Atom/RSS XML structures.

In EOxServer, both the [Time](#)⁹¹ and the [Geo](#)⁹² extensions are implemented to limit the spatio-temporal scope of the search.

Setup

To enable the OpenSearch interface in the EOxServer instance, the `urls.py` has to be adjusted and the following line added:

```
from django.conf.urls import patterns, include, url
...
from eoxserver.services.opensearch.urls import urlpatterns as opensearch

urlpatterns = patterns('',
    ...
    url(r'^opensearch/', include(opensearch)),
    ...
)
```

the region of the disk file being mapped to the output pixel (or possibly just a sampling of them). Generally AVERAGE can be desirable for reducing noise in dramatically downsampled data, and can give something approximating anti-aliasing for black and white linework. BILINEAR will compute a linear interpolation of the four pixels around the target location. BILINEAR can be helpful when oversampling data to give a smooth appearance.

³ These parameters are only used in conjunction with GeoTIFF output. Thus the format parameter must be either 'image/tiff' or the "native" format of the coverage maps to GeoTIFF. The specification of this encoding extension can be found [here](#)

⁹¹ http://www.opensearch.org/Specifications/OpenSearch/Extensions/Time/1.0/Draft_1

⁹² http://www.opensearch.org/Specifications/OpenSearch/Extensions/Geo/1.0/Draft_2

This adds the necessary URLs and views to the instances setup to expose the interface to the users.

Additionally, the the string "eoxserver.services.opensearch.**" has to be added to the COMPONENTS of the settings.py file.

Usage

The OpenSearch implementation of EOxServer follows a two-step search approach:

1. the instance can be searched for collections
2. single collections can be searched for records

For each of those steps, the OpenSearch interface allows two interactions, the `description` and the `search`. The `description` operation returns an XML document with service metadata and parametrized endpoints for further searches. The `search` operation hosts the main searching functionality: the search parameters are sent the service, and the results are encoded and returned.

Collection Search

To get the description of the OpenSearch service running in your instance, you have to access the URL previously specified in the `urlpatterns`. In the *autotest instance* (page 129), this looks like this:

```
$ curl http://localhost/opensearch/
<?xml version='1.0' encoding='iso-8859-1'?>
<OpenSearchDescription xmlns="http://a9.com/-/spec/opensearch/1.1/" xmlns:xsi=
↪ "http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="">
  <ShortName/>
  <Description/>
  <Url type="application/atom+xml" rel="collection" template="http://localhost/
↪ opensearch/atom/?q={searchTerms?}&count={count?}&startIndex={startIndex?}
↪ &bbox={geo:box?}&geom={geo:geometry?}&lon={geo:lon?}&lat=
↪ {geo:lat?}&r={geo:radius?}&georel={geo:relation?}&uid={geo:uid?}&start=
↪ {time:start?}&end={time:end?}&timerel={time:relation?}"/>
  <Url type="application/vnd.geo+json" rel="collection" template="http://localhost/
↪ opensearch/json/?q={searchTerms?}&count={count?}&startIndex={startIndex?}
↪ &bbox={geo:box?}&geom={geo:geometry?}&lon={geo:lon?}&lat=
↪ {geo:lat?}&r={geo:radius?}&georel={geo:relation?}&uid={geo:uid?}&start=
↪ {time:start?}&end={time:end?}&timerel={time:relation?}"/>
  <Url type="application/vnd.google-earth.kml+xml" rel="collection" template=
↪ "http://localhost/opensearch/kml/?q={searchTerms?}&count={count?}&
↪ startIndex={startIndex?}&bbox={geo:box?}&geom={geo:geometry?}&lon=
↪ {geo:lon?}&lat={geo:lat?}&r={geo:radius?}&georel={geo:relation?}&
↪ uid={geo:uid?}&start={time:start?}&end={time:end?}&timerel=
↪ {time:relation?}"/>
  <Url type="application/rss+xml" rel="collection" template="http://localhost/
↪ opensearch/rss/?q={searchTerms?}&count={count?}&startIndex={startIndex?}&
↪ amp;bbox={geo:box?}&geom={geo:geometry?}&lon={geo:lon?}&lat={geo:lat?}
↪ &r={geo:radius?}&georel={geo:relation?}&uid={geo:uid?}&start=
↪ {time:start?}&end={time:end?}&timerel={time:relation?}"/>
  <Contact/>
  <LongName/>
  <Developer/>
  <Attribution/>
  <SyndicationRight>open</SyndicationRight>
  <AdultContent/>
  <Language/>
  <InputEncoding/>
  <OutputEncoding/>
</OpenSearchDescription>
```

As you can see, the description XML document contains a `Url` element for each registered output format. Each URL also has a set of parameter placeholders from which the actual query can be constructed. Most of the parameters are optional, as indicated by the suffixed `?` within the curly braces.

To perform a search for collections, a request template has to be used and filled with *parameters* (page 53). See this example, where a simple bounding box is used to limit the search:

```
$ curl http://localhost/opensearch/atom/?bbox=10,33,12,35
<feed xmlns:georss="http://www.georss.org/georss" xmlns:geo="http://a9.com/-/
↪opensearch/extensions/geo/1.0/" xmlns:opensearch="http://a9.com/-/spec/
↪opensearch/1.1/" xmlns:time="http://a9.com/-/opensearch/extensions/time/1.0/"
↪xmlns="http://www.w3.org/2005/Atom">
  <id>http://localhost/opensearch/atom/?bbox=10,33,12,35</id>
  <title>None Search</title>
  <link href="http://localhost/opensearch/atom/?bbox=10,33,12,35" rel="self"/>
  <description/>
  <opensearch:totalResults>1</opensearch:totalResults>
  <opensearch:startIndex>0</opensearch:startIndex>
  <opensearch:itemsPerPage>1</opensearch:itemsPerPage>
  <opensearch:Query role="request" geo:box="10,33,12,35"/>
  <link href="http://localhost/opensearch/" type="application/
↪opensearchdescription+xml" rel="search"/>
  <link href="http://localhost/opensearch/atom/?bbox=10,33,12,35" type=
↪"application/atom+xml" rel="self"/>
  <link href="http://localhost/opensearch/atom/?bbox=10%2C33%2C12%2C35" type=
↪"application/atom+xml" rel="first"/>
  <link href="http://localhost/opensearch/atom/?startIndex=1&bbox=10%2C33%2C12
↪%2C35" type="application/atom+xml" rel="last"/>
  <entry>
    <title>MER_FRS_1P_reduced_RGB</title>
    <id>MER_FRS_1P_reduced_RGB</id>
    <link href="http://localhost/opensearch/collections/MER_FRS_1P_reduced_RGB/"
↪rel="search"/>
    <georss:box>32.264541 -3.437981 46.218445 27.968591</georss:box>
  </entry>
</feed>
```

The resulting atom feed contains information used for paging and the matched collections. Each entry (or item in RSS) contains a rough metadata overview of the collection and a link to the collections OpenSearch description document, which can be used to make searches for records within the collection.

Record Search

Searching for records within a collection is very similar to searching for collections on the service itself. The first step is to obtain the OpenSearch description document for the collections:

```
$ curl http://localhost/opensearch/collections/MER_FRS_1P_reduced_RGB/
<?xml version='1.0' encoding='iso-8859-1'?>
<OpenSearchDescription xmlns="http://a9.com/-/spec/opensearch/1.1/" xmlns:xsi=
↪"http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="">
  <ShortName/>
  <Description/>
  <Url type="application/atom+xml" rel="results" template="http://localhost/
↪opensearch/collections/MER_FRS_1P_reduced_RGB/atom/?q={searchTerms}&count=
↪{count?}&startIndex={startIndex?}&bbox={geo:box?}&geom={geo:geometry?}
↪&lon={geo:lon?}&lat={geo:lat?}&r={geo:radius?}&georel=
↪{geo:relation?}&uid={geo:uid?}&start={time:start?}&end={time:end?}&
↪amp;timerel={time:relation?}"/>
  <Url type="application/vnd.geo+json" rel="results" template="http://localhost/
↪opensearch/collections/MER_FRS_1P_reduced_RGB/json/?q={searchTerms}&count=
↪{count?}&startIndex={startIndex?}&bbox={geo:box?}&geom={geo:geometry?}
↪&lon={geo:lon?}&lat={geo:lat?}&r={geo:radius?}&georel=
↪{geo:relation?}&uid={geo:uid?}&start={time:start?}&end={time:end?}&
↪amp;timerel={time:relation?}"/>
```

```

<Url type="application/vnd.google-earth.kml+xml" rel="results" template="http://
↳localhost/opensearch/collections/MER_FRS_1P_reduced_RGB/kml/?q={searchTerms?}&
↳amp;count={count?}&startIndex={startIndex?}&bbox={geo:box?}&geom=
↳{geo:geometry?}&lon={geo:lon?}&lat={geo:lat?}&r={geo:radius?}&georel=
↳{geo:relation?}&uid={geo:uid?}&start={time:start?}&end=
↳{time:end?}&timerel={time:relation?}"/>
<Url type="application/rss+xml" rel="results" template="http://localhost/
↳opensearch/collections/MER_FRS_1P_reduced_RGB/rss/?q={searchTerms?}&count=
↳{count?}&startIndex={startIndex?}&bbox={geo:box?}&geom={geo:geometry?}
↳&lon={geo:lon?}&lat={geo:lat?}&r={geo:radius?}&georel=
↳{geo:relation?}&uid={geo:uid?}&start={time:start?}&end={time:end?}&
↳amp;timerel={time:relation?}"/>
<Contact/>
<LongName/>
<Developer/>
<Attribution/>
<SyndicationRight>open</SyndicationRight>
<AdultContent/>
<Language/>
<InputEncoding/>
<OutputEncoding/>
</OpenSearchDescription>

```

Again, the result contains a list of URL templates, one for each enabled result format. These templates can be used to perform the searches for records. The following example uses a time span to limit the records:

```

$ curl "http://localhost/opensearch/collections/MER_FRS_1P_reduced_RGB/json/?
↳start=2006-08-16T09:09:29Z&end=2006-08-22T09:09:29Z"
{
  "type": "FeatureCollection",
  "bbox": [ 11.648344, 32.269746, 27.968591, 46.216558 ],
  "features": [
    { "type": "Feature", "properties": { "id": "mosaic_MER_FRS_1PNPDE20060816_090929_
↳000001972050_00222_23322_0058_RGB_reduced", "begin_time": "2006-08-16T09:09:29Z",
↳ "end_time": "2006-08-16T09:12:46Z" }, "bbox": [ 11.648344, 32.269746, 27.968591,
↳ 46.216558 ], "geometry": { "type": "MultiPolygon", "coordinates": [ [ [ [ 14.
↳322576, 46.216558 ], [ 14.889221, 46.152076 ], [ 15.714163, 46.044475 ], [ 16.
↳939196, 45.874384 ], [ 18.041168, 45.707637 ], [ 19.696621, 45.437661 ], [ 21.
↳061979, 45.188708 ], [ 22.14653, 44.985502 ], [ 22.972839, 44.817601 ], [ 24.
↳216794, 44.548719 ], [ 25.078471, 44.353026 ], [ 25.619454, 44.222401 ], [ 27.
↳096691, 43.869453 ], [ 27.968591, 43.648678 ], [ 27.608909, 42.914276 ], [ 26.
↳904154, 41.406745 ], [ 26.231198, 39.890887 ], [ 25.79281, 38.857425 ], [ 25.
↳159378, 37.327455 ], [ 24.607823, 35.91698 ], [ 24.126822, 34.659956 ], [ 23.
↳695477, 33.485864 ], [ 23.264471, 32.269746 ], [ 21.93772, 32.597366 ], [ 20.
↳490342, 32.937415 ], [ 18.720985, 33.329502 ], [ 17.307239, 33.615994 ], [ 16.
↳119969, 33.851259 ], [ 14.83709, 34.086159 ], [ 13.692708, 34.286728 ], [ 12.
↳702329, 34.450209 ], [ 11.648344, 34.612576 ], [ 11.818952, 35.404302 ], [ 12.
↳060892, 36.496444 ], [ 12.273682, 37.456615 ], [ 12.465752, 38.338768 ], [ 12.
↳658489, 39.179619 ], [ 12.861886, 40.085426 ], [ 13.125704, 41.224754 ], [ 13.
↳249298, 41.773101 ], [ 13.442094, 42.58703 ], [ 13.647311, 43.450338 ], [ 13.
↳749196, 43.879742 ], [ 13.904244, 44.51596 ], [ 14.076176, 45.247154 ], [ 14.
↳21562, 45.812577 ], [ 14.322576, 46.216558 ] ] ] ] }
  ]
}

```

Parameters

As mentioned before, EOxServers implementation of OpenSearch adheres to the core, and the time and geo extensions. Thus the interface allows the following parameters when searching for datasets:

Table 1.7: OpenSearch Search Request Parameters

Parameter (Replacement Tag)	Description	Example
→ q (searchTerms)	This parameter is currently not used.	
→ count	Number of returned elements as an integer	count=25
→ startIndex	The initial offset to get elements as an integer	startIndex=125
→ format	The output format of the search. Currently supported are “json”, “kml”, “atom”, and “rss”.	format=json
→ bbox (geo:box)	The geographical area expressed as a bounding box defined as “west,south,east,north” in EPSG:4326 decimal degrees.	bbox=-120.0,40.5,-110.5,43.8
→ lat and lon (geo:lat/geo:lon)	latitude and longitude geographical coordinate pair as decimal degrees in EPSG:4326.	lat=32.25&lon=125.654
→ r (geo:radius)	The radius parameter used with lat and lon parameters. Units are meters on along the earths surface.	lat=32.25&lon=125.654
→ geom (geo:geometry)	A custom geometry encoded as WKT. Supported are POINT, LINESTRING, POLYGON, MULTIPOINT, MULTILINESTRING, and MULTIPOLYGON. The geometry must be expressed in EPSG:4326.	geom=POINT(6 10) geom=LINESTRING(3 4,1 5,20 25)
→ georel (geo:relation)	The geospatial relation of the supplied geometry (or bounding box/circle) and the searched datasets geometry. This parameter allows the following values: - “intersects” (default): the passed geometry has to <ul style="list-style-type: none"> intersect with the datasets geometry • “contains”: the passed geometry has to fully enclose datasets geometry. Currently only PostgreSQL/PostGIS supports this relation for distance lookups. • “disjoint”: the passed geometry has no spatial overlap with the datasets geometry. 	georel=contains
→ uid (geo:uid)	This parameter allows to match a single record by its exact identifier. This is also used to allow links to searches with only a specific item, as used in the atom and RSS formats.	uid=MER_FRS_1P_reduced_RGB
→ start and end (time:start/time:end)	The start and end data/time of the given time interval encoded in ISO 8601 ⁹³ .	start=2006-08-16T09:09:29Z&end=2006-08-17
→ timerel (time:relation)	The temporal relation between the passed interval and the datasets time intervals. This parameter allows the following values: - “intersects”: the given interval has to	timerel>equals

Note: Unfortunately there are some known issues for certain parameters, especially concerning the `geo:radius` with the `geo:lat` and `geo:lon`: On certain platforms any distance based search results in an abort caused by [GEOS⁹⁴](#), the underlying geometric algorithm library.

All parameters are available for both collection and record searches.

Output Formats

EOxServer supports various output formats to encode the results of the searches. All formats are available for both collection and record searches.

ATOM and RSS

The EOxServer OpenSearch implementation tries to adhere the specification and recommendations for using OpenSearch with either of the two formats. Apart from the usual metadata links are added to the various enabled services like WMS and WCS wherever applicable. When searching for collections a link to the collections OpenSearch description document is also added.

GeoJSON and KML

These formats aim to provide only a compact metadata overview of the matched collections and records. Only the identifier, begin/end timestamps and the footprint geometry are included.

Enabling/Disabling Formats

With the steps described in [Setup](#) (page 50), all formats are enabled by default. To limit the available formats, the line added to the line `"eoxserver.services.opensearch.**"` of the COMPONENTS setting in the `settings.py` must be replaced by the following:

```
COMPONENTS = [
    ...
    "eoxserver.services.opensearch.v11.*",
    "eoxserver.services.opensearch.extensions.*",
    "eoxserver.services.opensearch.formats.<format>",
]
```

where `<format>` is one of `atom`, `geojson`, `kml` or `rss`. To enable more than one format, the last line can be repeated for each format.

Future Work

As of EOxServer version 0.4, it is planned to also implement support for the [OpenSearch EO⁹⁵](#) extension. This extension was held back on purpose, as the current data models do not include the necessary metadata fields.

Additionally, the aim is to support most of the required and recommended best practices of the [CEOS OpenSearch Best Practice Document⁹⁶](#).

⁹⁴ <https://trac.osgeo.org/geos/ticket/377>

⁹⁵ https://portal.opengeospatial.org/files/?artifact_id=61006

⁹⁶ https://earthdata.nasa.gov/files/CEOS_OpenSearch_Best_Practice_Doc-v.1.0.1_Jun2015.pdf

EOxServer Operators' Guide

Table of Contents

- *EOxServer Operators' Guide* (page 56)
 - *Basic Concepts* (page 57)
 - *Storage Backends* (page 57)
 - * *Local* (page 57)
 - * *FTP Repositories* (page 57)
 - * *Rasdaman Databases* (page 57)
 - *Data Items* (page 58)
 - *Coverages* (page 58)
 - * *Range Types* (page 58)
 - * *Rectified Datasets* (page 59)
 - * *Referenceable Datasets* (page 59)
 - * *Rectified Stitched Mosaics* (page 59)
 - * *Dataset Series* (page 59)
 - *Data Preparation and Supported Data Formats* (page 59)
 - * *Raster Data Formats* (page 59)
 - * *Raster Data Preparation* (page 60)
 - * *Metadata Formats* (page 60)
 - * *Metadata Preparation* (page 62)
 - *Admin Client* (page 62)
 - * *Creating a custom Range Type* (page 62)
 - * *Creating a Dataset* (page 64)
 - * *Creating a Dataset Series or a Stitched Mosaic* (page 64)
 - *Command Line Tools* (page 64)
 - * *eoxserver-instance.py* (page 64)
 - * *eoxs_dataset_register* (page 67)
 - * *eoxs_dataset_deregister* (page 67)
 - * *Updating Datasets* (page 67)
 - * *eoxs_collection_create* (page 67)
 - * *eoxs_collection_delete* (page 68)
 - * *eoxs_collection_link and eoxs_collection_unlink* (page 68)
 - * *eoxs_collection_purge* (page 68)
 - * *eoxs_collection_datasource* (page 68)
 - * *eoxs_collection_synchronize* (page 68)
 - * *eoxs_id_check* (page 69)
 - * *Range Type Handling* (page 70)

Basic Concepts

EOxServer is all about coverages - see the *EOxServer Basics* (page 1) for a short description.

In the language of the OGC Abstract Specification, coverages are mappings from a domain set that is related to some area of the Earth to a range set. So, the data model for coverages contains information about the structure of the domain set and of the range set (the so-called Range Type).

In the *Coverages* (page 58) section below you find more detailed information about what data and metadata is stored by EOxServer.

The actual data EOxServer deals with can be stored in different ways. These storage facilities are discussed below in the section on *Storage Backends* (page 57).

Operators have different possibilities to ingest data into the system. Using the *Admin Client* (page 62), you can edit the contents of the EOxServer database. Especially for batch processing using the *Command Line Tools* (page 64) may be preferable.

Storage Backends

EOxServer supports different kinds of data stores for coverage data (additional backends can be added as plugin):

- as an image file stored on the local file system
- as an image file stored on a remote FTP or HTTP server
- as a raster array in a *rasdaman*⁹⁷ database

Local

A path on the local filesystem is the most straightforward way to define the location of a resource. You can use relative paths as well as absolute paths. Please keep in mind that relative paths are interpreted as being relative to the working directory of the process EOxServer runs in. For Apache processes, for instance, this is usually the root directory /.

FTP Repositories

EOxServer allows to define locations on a remote FTP server. This is useful if you do not want to transfer a whole large archive to the machine EOxServer runs on. In that case you can define a remote path that consists of information about the FTP server and the path relative to the root directory of the FTP repository.

An FTP Storage record - as it is called in EOxServer - contains the URL of the server and optional port, username and password entries.

Resources stored on an FTP server are transferred only when they are needed. There is however a cache for transferred files on the machine EOxServer runs on.

Rasdaman Databases

The third backend supported at the moment are *rasdaman*⁹⁸ databases. A *rasdaman* location consists of *rasdaman* database connection information and the collection of the corresponding resource.

The *rasdaman* storage records contain hostname, port, database name, user and password entries.

⁹⁷ <http://www.rasdaman.org>

⁹⁸ <http://www.rasdaman.org>

The data is retrieved from the database using the rasdaman GDAL driver (see *Installation* (page 15) for further information).

Data Items

A data item describes a single resource located on a storage, where the “local” storage (the local filesystem) is assumed if no other storage is defined. The path of a data item is always relative to its storage and might in some special cases have a specific meaning. This is defined in the Storage plugin that handles the specific backend.

Each data item has a semantic, which defines the actual usage of this data item. This might be “metadata” for metadata files or “bands[1:3]” for raster data. The usage of this field is really generic and depends on the context.

The format of a data item has informative character of how it might be interpreted. Use default MIME types here.

Coverages

EOxServer coverages fall into three main categories:

- *Rectified Datasets* (page 59)
- *Referenceable Datasets* (page 59)
- *Rectified Stitched Mosaics* (page 59)

In addition there is the *Dataset Series* (page 59) type which corresponds to an inhomogeneous collection of coverages.

Every coverage is a set of associated Data Items which define where the actual data of the coverage can be found.

Additionally every coverage has associated EO Metadata, that defines the acquisition time and the area of interest within the coverage.

Range Types

Every coverage has a range type describing the structure of the data. Each range type has a given data type whereas the following data types are supported:

Data Type Name	Data Type Value
Unknown	0
Byte	1
UInt16	2
Int16	3
UInt32	4
Int32	5
Float32	6
Float64	7
CInt16	8
CInt32	9
CFloat32	10
CFloat64	11

A range type contains of one or more bands. For each band you may specify a name, an identifier and a definition that describes the property measured (e.g. radiation). Furthermore, you can define nil values for each band (i.e. values that indicate that there is no measurement at the given position).

This range type metadata is used in the coverage description metadata that is returned by WCS operations and for configuring WMS layers.

Note that WMS supports only one data type (Byte) and only Grayscale and RGB output. Any other range types will be mapped to these: for single-band coverages, Grayscale output is generated and RGB output using the first three bands for all others. Automatic scaling is applied when mapping from another data type to Byte. That

means the minimum-maximum interval for the given subset of the coverage is computed and mapped to the 0-255 interval supported by the Byte data type.

If you want to view other band combinations than the default ones, you can use the EO-WMS features implemented by EOxServer. For each coverage, an additional layers called `<coverage id>_bands` is provided for WMS 1.3. Using this layer and the `DIM_BANDS` KVP parameter you can select another combination of bands (either 1 or 3 bands).

Rectified Datasets

Rectified Datasets are EO coverages whose domain set is a rectified grid i.e. which are having a regular spacing in projected or geographic CRS. In practice, this applies to ortho-rectified satellite data. The rectified grid is described by the EPSG SRID of the coordinate reference system, the extent and pixel size of the coverage.

Rectified Datasets can be added to Dataset Series and Rectified Stitched Mosaics.

Referenceable Datasets

Referenceable Datasets are EO coverages whose domain set is a referenceable grid i.e. which are not rectified, but are associated with (one or more) coordinate transformation which relate the image to a projected or geographic CRS. That means that there is some general transformation between the grid cell coordinates and coordinates in an Earth-bound spatial reference system. This applies for satellite data in its original geometry.

At the moment, EOxServer supports only referenceable datasets that contain ground control points (GCPs) in the data files. Simple approximative transformations based on these GCPs are used to generate rectified views on the data for WMS and to calculate subset bounds for WCS GetCoverage requests. Note that these transformations can be very inaccurate in comparison to an actual ortho-rectification of the coverage.

Rectified Stitched Mosaics

Rectified Stitched Mosaics are EO coverages that are composed of a set of homogeneous Rectified Datasets. That means, the datasets must have the same range type and their domain sets must be subsets of the same rectified grid.

When creating a Rectified Stitched Mosaic a homogeneous coverage is generated from the contained Rectified Datasets. Where datasets overlap the most recent one as indicated by the acquisition timestamps in the EO meta-data is shown on top hiding the others.

Dataset Series

Any Rectified and Referenceable Datasets can be organized in Dataset Series. Multiple datasets which are spatially and/or temporally overlapping can be organized in a Dataset Series. Furthermore Stitched Mosaics can also be organized in Dataset Series.

Data Preparation and Supported Data Formats

EO Coverages consist of raster data and metadata. The way this data is stored can vary considerably. EOxServer supports a wide range of different data and metadata formats which are described below.

Raster Data Formats

EOxServer uses the [GDAL](http://www.gdal.org)⁹⁹ library for raster data handling. So does [MapServer](http://www.mapserver.org)¹⁰⁰ whose scripting API (Map-Script) is used by EOxServer as well. In principle, any [format supported by GDAL](http://www.gdal.org/formats_list.html)¹⁰¹ can be read by EOxServer

⁹⁹ <http://www.gdal.org>

¹⁰⁰ <http://www.mapserver.org>

¹⁰¹ http://www.gdal.org/formats_list.html

and registered in the database.

There is, however, one caveat. Most data formats are composed of bands which contain the data (e.g. ENVISAT N1, GeoTIFF, JPEG 2000). But some data formats (notably netCDF and HDF) have a different substructure: subdatasets. At the moment these data formats are only supported for data output, but not for data input.

For more information on configuration of supported raster file formats read “*Supported Raster File Formats and Their Configuration* (page 111)”.

Raster Data Preparation

Usually, raster data does not need to be prepared in a special way to be ingested into EOxServer.

If the raster data file is structured in subdatasets, though, as is the case with netCDF and HDF, you will have to convert it to another format. You can use the `gdal_translate` command for that task:

```
$ gdal_translate -of <Output Format> <Input File Name> <Output File Name>
```

You can display the list of possible output formats with:

```
$ gdalinfo --formats
```

For automatic registration of datasets, EOxServer relies on the geospatial metadata stored with the dataset, notably the EPSG ID of the coordinate reference system and the geospatial extent. In some cases the CRS information in the dataset does not contain the EPSG code. If you are using the command line interfaces of EOxServer you can specify an SRID with the `--default-srid` option. As an alternative you can try to add the corresponding information to the dataset, e.g. with:

```
$ gdal_translate -a_srs "+init=EPSG:<SRID>" <Input File Name> <Output File Name>
```

For performance reasons, especially if you are using WMS, you might also consider to add overviews to the raster data files using the `gdaladdo` command ([documentation](#)¹⁰²). Note however that this is supported only by a few formats like GeoTIFF and JPEG2000.

Metadata Formats

There are two possible ways to store metadata: the first one is to store it in the data file itself, the second one is to store it in an accompanying metadata file.

Only a subset of the supported raster data formats are capable of storing metadata in the data file. Furthermore there are no standards defining the semantics of the metadata for generic formats like GeoTIFF. For mission specific formats, however, there are thorough specifications in place.

EOxServer supports reading basic metadata from ENVISAT N1 files and files that have a similar metadata structure (e.g. a GeoTIFF file with the same metadata tags).

For other formats metadata files have to be provided. EOxServer supports two XML-based formats:

- OGC Earth Observation Profile for Observations and Measurements (OGC 10-157r2)
- an EOxServer native format

Here is an example for EO O&M:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<eop:EarthObservation gml:id="eop_ASA_WSM_1PNDPA20050331_075939_000000552036_00035_
↵16121_0775" xmlns:eop="http://www.opengis.net/eop/2.0" xmlns:gml="http://www.
↵opengis.net/gml/3.2" xmlns:om="http://www.opengis.net/om/2.0">
  <om:phenomenonTime>
    <gml:TimePeriod gml:id="phen_time_ASA_WSM_1PNDPA20050331_075939_000000552036_
↵00035_16121_0775">
```

¹⁰² <http://www.gdal.org/gdaladdo.html>

```

    <gml:beginPosition>2005-03-31T07:59:36Z</gml:beginPosition>
    <gml:endPosition>2005-03-31T08:00:36Z</gml:endPosition>
  </gml:TimePeriod>
</om:phenomenonTime>
<om:resultTime>
  <gml:TimeInstant gml:id="res_time_ASA_WSM_1PNDPA20050331_075939_000000552036_
↪00035_16121_0775">
    <gml:timePosition>2005-03-31T08:00:36Z</gml:timePosition>
  </gml:TimeInstant>
</om:resultTime>
<om:procedure />
<om:observedProperty />
<om:featureOfInterest>
  <eop:Footprint gml:id="footprint_ASA_WSM_1PNDPA20050331_075939_000000552036_
↪00035_16121_0775">
    <eop:multiExtentOf>
      <gml:MultiSurface gml:id="multisurface_ASA_WSM_1PNDPA20050331_075939_
↪000000552036_00035_16121_0775" srsName="http://www.opengis.net/def/crs/EPSG/0/
↪4326">
        <gml:surfaceMember>
          <gml:Polygon gml:id="polygon_ASA_WSM_1PNDPA20050331_075939_
↪000000552036_00035_16121_0775">
            <gml:exterior>
              <gml:LinearRing>
                <gml:posList>-33.03902600 22.30175400 -32.53056000 20.09945700 -
↪31.98492200 17.92562200 -35.16690300 16.72760500 -35.73368300 18.97694800 -36.
↪25910700 21.26212300 -33.03902600 22.30175400</gml:posList>
              </gml:LinearRing>
            </gml:exterior>
          </gml:Polygon>
        </gml:surfaceMember>
      </gml:MultiSurface>
    </eop:multiExtentOf>
  </eop:Footprint>
</om:featureOfInterest>
<om:result />
<eop:metaDataProperty>
  <eop:EarthObservationMetaData>
    <eop:identifier>ASA_WSM_1PNDPA20050331_075939_000000552036_00035_16121_0775</
↪eop:identifier>
    <eop:acquisitionType>NOMINAL</eop:acquisitionType>
    <eop:status>ARCHIVED</eop:status>
  </eop:EarthObservationMetaData>
</eop:metaDataProperty>
</eop:EarthObservation>

```

The native format has the following structure:

```

<Metadata>
  <EOID>some_unique_eoid</EOID>
  <BeginTime>YYYY-MM-DDTHH:MM:SSZ</BeginTime>
  <EndTime>YYYY-MM-DDTHH:MM:SSZ</EndTime>
  <Footprint>
    <Polygon>
      <Exterior>Mandatory - some_pos_list as all-space-delimited Lat Lon_
↪pairs (closed polygon i.e. 5 coordinate pairs for a rectangle) in EPSG:4326</
↪Exterior>
      [
        <Interior>Optional - some_pos_list as all-space-delimited Lat Lon_
↪pairs (closed polygon) in EPSG:4326</Interior>
        ...
      ]
    </Polygon>

```

```
</Footprint>
</Metadata>
```

The automatic registration tools for EOxServer (see below under *Command Line Tools* (page 64)) expect that the metadata file accompanying the data file has the same name with `.xml` as extension.

Metadata Preparation

EOxServer provides a tool to extract metadata from ENVISAT N1 files and convert it to EO O&M format. It can be found under `tools/gen_envisat_md.py`. It accepts an input path to an N1 file and stores the resulting XML file under the same path with the appropriate file name (i.e. replacing the `.N1` extension with `.xml`). Note that EOxServer must be in the Python path and the environment variable `DJANGO_SETTINGS_MODULE` must be set and point to a properly configured EOxServer instance.

Admin Client

The Admin Client is accessible via any standard web browser at the path `/admin` under the URL your instance is deployed or simply by following the `admin` link on the start page. *Deployment* (page 23) provides more details.

Use the username and password you provided during the `syncdb` step as described in the *Service Instance Creation and Configuration* (page 21) section.

Creating a custom Range Type

Before registering any data in EOxServer some vital information on the datasets has to be provided. Detailed information regarding the kind of data stored can be defined in the Range Type. A Range Type is a collection of bands which themselves are assigned to a specific Data Type (see *Range Types* (page 58)).

A simple standard PNG for example holds 4 bands (RGB + Alpha) each of them able to store 8 bit data. Therefore the Range Type would have to be defined with four bands (red, green, blue, alpha) each of them having 'Byte' as Data Type.

In our example we use the reduced MERIS RGB data provided in the autotest instance. `gdalinfo` provides us with the most important information:

```
[...]
Band 1 Block=541x5 Type=Byte, ColorInterp=Red
Band 2 Block=541x5 Type=Byte, ColorInterp=Green
Band 3 Block=541x5 Type=Byte, ColorInterp=Blue
```

In order to define a new Range Type we click on the "Add" button next to the "Range Types" in the home menu of the admin client. Here we define the name of the Range Type and add bands to it by clicking on "Add another band".

For each band in "Name", "Identifier" and "Description" you can enter the same content for now. The default "Definition" value for now can be "<http://www.opengis.net/def/property/OGC/0/Radiance>". "UOM" stands for "unit of measurement" which in our case is radiance defined by the value "W.m-2.Sr-1". For displaying the data correctly it is recommended to assign the respective value in "Color Interpretation". If your data is distributed in only a portion of the possible values of its data type it is best to define "Raw value min" and "Raw value max" to have a better visual representation in e.g WMS. You can add a Nilvalue set to each of the bands, which is explained in the next section.

With the "index" you can finetune the index of the band within the range type.

To define invalid values of the image, for each band a set of nil values can be defined. To create one navigate to `/admin/coverages/nilvalueset` and click on the button "Add Nil Value Set". Here you can define a name of the set (which you can later use to set it in the band) and set the nil value(s) definition and reason. You can also add additional nil values to the set by clicking "Add another Nil Value". To add the NilValue set to the band(s), you have to navigate back to your range type admin page and set the nilvalue set to your band.

E0xServer Admin Client Welcome, admin. Documentation / Change password / Log out

Home > Coverages > Range Types > Add Range Type

Add Range Type

Name:

Index	Name	Identifier	Description	Definition	Uom	Data type	Color interpretation
<input type="text" value="0"/>	<input type="text" value="red"/>	<input type="text" value="red"/>	Red Band	<input type="text" value="http://www.opengis.r"/>	<input type="text" value="W.m-2.5r-1"/>	<input type="text" value="Byte"/>	<input type="text" value="RedBand"/>
<input type="text" value="1"/>	<input type="text" value="green"/>	<input type="text" value="green"/>	Green Band	<input type="text" value="http://www.opengis.r"/>	<input type="text" value="W.m-2.5r-1"/>	<input type="text" value="Byte"/>	<input type="text" value="GreenBand"/>
<input type="text" value="2"/>	<input type="text" value="blue"/>	<input type="text" value="blue"/>	Blue Band	<input type="text" value="http://www.opengis.r"/>	<input type="text" value="W.m-2.5r-1"/>	<input type="text" value="Byte"/>	<input type="text" value="BlueBand"/>

[+ Add another Band](#)

E0xServer Admin Client Welcome, admin. Documentation / Change password / Log out

Home > Coverages > Nil Value Sets > RGB Nil Value Set (Byte)

Change Nil Value Set History

Name:

Data type:

Raw value	Reason	Delete?
<input type="text" value="http://www.opengis.net/def/nil/OGC/0/unknown (0)"/> <input type="text" value="0"/>	<input type="text" value="Unknown"/>	<input type="checkbox"/>
<input type="text"/>	<input type="text" value="-----"/>	<input type="checkbox"/>

[+ Add another Nil Value](#)

To list, export, and load range types using the command-line tools see *Range Type Handling* (page 70).

Creating a Dataset

To create a Rectified or Referenceable Dataset from the admin click on either of the “Add” buttons next to the corresponding dataset type in the home screen. For both Dataset types the following fields must be set:

- Identifier: a unique identifier for the Dataset
- Range Type
- Size for both X and Y axis
- The bounding box (min x, min y, max x, max y). The bounding box is expressed in the CRS defined by either “SRID” or “Projection” of which one *must* be specified

The following items *should* be set:

- Begin and end time: if available this should be set to let the various services allow a temporal search
- Footprint: this should be set as-well to let the various services perform spatial searches.

To link actual files containing data and metadata to the Dataset, we have to add Data Items. Each data item has a “location”, a “format” (mime-type) and a “semantic” (band data, metadata or anything else related).

The “location” is relative to either the “storage” or “package” if available, otherwise the location is treated a local (relative or absolute) path. A “Storage” defines a remote service like FTP, HTTP or similar. A package abstracts archives like TAR or ZIP files. Packages have a location themselves and can also reside on a storage or be located within another package themselves.

To add a local 15-bands GeoTIFF and a local metadata XML-file to the Dataset use the following values:

Location	Format	Semantic
path/to/data.tiff	image/tiff	bands[1:15]
path/to/metadata.xml	eogml	metadata

If the raster-data is distributed among several files you can use several data items with semantic bands [low:high] where low and high are the 1-based indices.

You can directly add the dataset to one or multiple collections in the “EO Object to Collection Relations” section.

Creating a Dataset Series or a Stitched Mosaic

A Dataset Series is a very basic type of collection that can contain Datasets, Stitched Mosaics and even other Dataset Series. The creation of a dataset series is fairly simple: In the admin click on “Add Dataset Series”, enter a valid identifier, add elements (in the “EO Object to Collection Relations” section) and click on “save”. The metadata (footprint, begin time and end time) are automatically collected upon the save.

The creation of a Stitched Mosaic is similar to the creation of a Dataset Series with some restrictions:

- the Range Type, overall size and exact bounding box must be specified (exactly as with Datasets)
- only Rectified Datasets that lie on the exact same grid can be added

Command Line Tools

eoxserver-instance.py

The first important command line tool is used for *Service Instance Creation and Configuration* (page 21) of EOxServer and is explained in the *Installation* (page 15) section of this user’ guide.

Home > Coverages > Rectified Datasets > Add Rectified Dataset

Add Rectified Dataset

Identifier:

Metadata

Geospatial metadata

Range type:  

Size x: Size y:

Min x: Min y:

Max x: Max y:

Srid: Projection:  

Begin time: Date: Today |  End time: Date: Today | 
Time: Now |  Time: Now | 

Footprint: 

Visible

Home > Coverages > Dataset Series > Add Dataset Series

Add Dataset Series

Identifier:

Metadata

Begin time: Date: Today | End time: Date: Today |
 Time: Now | Time: Now |

Footprint:



Delete all Features

Data sources

Pattern	Delete?
+ Add another Data Source	

EO Object to Collection Relations

Eo object	Delete?
MER_FRS_1PNPDE20060816_090929_000001972050_00222_23322_0058_uint16_reduced_compressed (Rectified Dataset)	
MER_FRS_1PNPDE20060822_092058_000001972050_00308_23408_0077_uint16_reduced_compressed (Rectified Dataset)	<input type="checkbox"/>
MER_FRS_1PNPDE20060830_100949_000001972050_00423_23523_0079_uint16_reduced_compressed (Rectified Dataset)	<input type="checkbox"/>
+ Add another Eo Object To Collection Relation	

eoxs_dataset_register

Besides this tool EOxServer adds some custom commands to Django's `manage.py` script. The `eoxs_dataset_register` command is detailed in the *Data Registration* (page 24) section.

eoxs_dataset_deregister

The `eoxs_dataset_deregister` command allows the de-registration of existing datasets (simple coverage types as Rectified and Referenceables datasets only) from an EOxServer instance including proper unlinking from relevant container types. The functionality of this command is complementary to the `eoxs_dataset_register` (page 67) command.

It is worth to mention that the de-registration does not remove physical data stored in the file system or different storage backende. Therefore an extra effort has to be spent to purge the physical data/meta-data files from their storage.

To de-register a dataset (coverage) identified by its (Coverage/EO) identifier the following command shall be invoked:

```
python manage.py eoxs_dataset_deregister <identifier>
```

The de-registration command allows convenient de-registration of an arbitrary number of datasets at the same time:

```
python manage.py eoxs_dataset_deregister <identifier> [<identifier> ...]
```

The `eoxs_dataset_deregister` does not allow the removing of container objects such as Rectified Stitched Mosaics or Dataset Series.

The `eoxs_dataset_deregister` command, by default, does not allow the de-registration of automatic datasets (i.e, datasets registered by the synchronisation process, see *What is synchronization?* (page 69)). Although this restriction can be overridden by the `--force` option, it is not recommended to do so.

Updating Datasets

There is currently no way how to update registered EOxServer datasets from the command line. In case such an action would be needed it is recommended to de-register the existing dataset first (see `eoxs_dataset_deregister` (page 67) command) and register it again with the updated parameters (see `eoxs_dataset_register` (page 67) command). Special attention should be paid to linking of the *updated* dataset to all the container objects during the registration as this information is removed by the de-registration.

eoxs_collection_create

The `eoxs_collection_create` command allows the creation of a dataset series with initial data sources or coverages included. In it's simplest use case, only the `--identifier` parameter is required, which has to be a valid and not yet taken identifier for the collection. By default a Dataset Series is created.

Range types for datasets can be read from configuration files that are accompanying them. There can be a configuration file for each dataset or one that applies to all datasets contained within a directory corresponding to a data source. Configuration files have the file extension `.conf`. The file name is the same as the one of the dataset (so the dataset `foo.tiff` needs to be accompanied by `foo.conf`) or `__default__.conf` if you want to use the config file for the whole directory. The syntax for the file is as follows:

```
[range_type]
range_type_name=<range type name>
```

Both approaches may be combine and configuration files produced only for some of the datasets in a directory and a default range type defined in `__default__.conf`. EOxServer will first look up the dataset configuration file and fall back to the default only if there is no individual `.conf` file.

Already registered datasets can be automatically added to the Dataset Series by using the `--add` option which takes an identifier of the Dataset or collection to be added. This option can be used multiple times.

If the collection is intended to be a sub-collection of another collection it can be inserted via the `--collection` parameter that also requires the identifier of the collection. Again, this parameter can be used multiple times.

eoxs_collection_delete

With this command an existing collection can be removed. When the `--force` switch is not set, only empty collections can be deleted. With the `--recursive` option all sub-collections will be deleted aswell.

This command does *never* remove any Datasets.

eoxs_collection_link and eoxs_collection_unlink

These two commands insert and remove links between objects and collections. To insert an object into a collection use the following command:

```
eoxs_collection_link --add <object-identifier> --collection <collection-identifier>
```

To do the opposite do the following:

```
eoxs_collection_unlink --remove <object-identifier> --collection <collection-  
↪identifier>
```

eoxs_collection_purge

To quickly remove the contents of a single collection from the database, the `eoxs_collection_purge` command can be used. This command deregisters all contained datasets of a collection. When the `--recursive` option is set, all contained sub-collections are purged aswell. Using the `--delete` option, the purged collections themselves are deleted too.

eoxs_collection_datasource

This command allows to add a datasource to a collection. A datasource consists of a primary `source` and zero or more secondary `templates`. The `source` should be a path using unix shell regular expressions to match files in the given directory structure. The `templates` are similar, but should make use of template tags that are then replaced the values of the `source`. Possible tags are:

- `{basename}`: the sources file basename (name without directory)
- `{root}`: like `{basename}`, but without file extension
- `{extension}`: the source files extension
- `{dirname}`: the directory path of the source file
- `{source}`: the full path of the source file

Example:

```
python manage.py eoxs_collection_datasource -i MER_FRS_1P \  
-s data/MER_FRS_1P_reduced/*tif \  
-t data/MER_FRS_1P_reduced/{root}.xml
```

eoxs_collection_synchronize

This command allows to synchronize a collection with the file system using its datasources.

What is synchronization?

In the context of EOxServer, synchronization is the process of updating the database models for container objects (such as `RectifiedStitchedMosaics` or `DatasetSeries`) according to changes in the file system.

Automatic datasets are deleted from the database, when their data files cannot be found in the file system. Similar, new datasets will be created when new files matching the search pattern in the subscribed directories are found.

When datasets are added to or deleted from a container object, the metadata (e.g the footprint of the features of interest or the time extent of the image) of the container is also likely to be adjusted.

Reasons for Synchronization

There are several occasions, where synchronization is necessary:

- A file has been added to a folder associated with a container
- A file from a folder associated with a container has been removed
- EO Metadata has been changed
- A regular check for database consistency

HowTo

Synchronization can be triggered by a custom Django admin command¹⁰³, called `eoxs_collection_synchronize`.

To start the synchronization process, navigate to your instances directory and type:

```
python manage.py eoxs_synchronize -i <ID> [ -i <ID> ... ]
```

whereas `<IDs>` are the coverage/EO IDs of the containers that shall be synchronized.

Alternatively, with the `-a` or `--all` option, all container objects in the database will be synchronized. This option is useful for a daily cron-job, ensuring the databases consistency with the file system.

```
python manage.py eoxs_collection_synchronize --all
```

The synchronization process may take some time, especially when FTP/Rasdaman storages are used and also depends on the number of synchronized objects.

eoxs_id_check

The `eoxs_check_id` commands allows checking about status of the queried coverage/EO identifier. The command returns the status via its return code (0 - True or 1 - False).

By default the command checks whether an identifier can be used (is available) as a new Coverage/Collection ID:

```
python manage.py eoxs_id_check <ID> && echo True || echo False
```

It is possible to check if the identifier is used for a specific type of object. For example, the following would check if the identifier is used for a Dataset Series:

```
python manage.py eoxs_id_check <ID> --type DatasetSeries && echo True || echo False
```

¹⁰³ <https://docs.djangoproject.com/en/1.4/ref/django-admin/>

Range Type Handling

The `eoxs_rangetypes_list` command, by default, lists the names of all registered range types:

```
python manage.py eoxs_rangetypes_list
```

In case of more range types details required verbose listing may be requested by `--details` option. When one or more range type names are specified the output will be limited to the specified range-types only:

```
python manage.py eoxs_rangetypes_list --details [<range-type-name> ...]
```

The same command can be also used to export rangetype in JSON format (`--json` option). Following example prints the selected RGB range type in JSON format:

```
python manage.py eoxs_rangetypes_list --json RGB
```

The output may be directly savaved to file by using the `-o` option. Following example saves all the registered range-types to a file named `rangetypes.json`:

```
python manage.py eoxs_rangetypes_list --json -o rangetypes.json
```

The rangetypes saved in JSON format can be loaded (e.g., by another *EOxServer* instance) by using of the `eoxs_rangetypes_load` command. By default, this command reads the JSON data from the standard input. To force the command to read the input from a file use `-i`

```
python manage.py eoxs_rangetypes_load -i rangetypes.json
```

Performance

The performance of different EOxServer tasks and services depends heavily on the hardware infrastructure and the data to be handled. Tests were made for two typical operator use cases:

- registering a dataset
- generating a mosaic

The tests for **registering datasets** were performed on a quad-core machine with 4 GB of RAM and with a SQLite/Spatialite database. The test datasets were 58 IKONOS multispectral (4-band 16-bit), 58 IKONOS panchromatic (1-band 16-bit) and 58 IKONOS pansharpened (3-band 8-bit) scenes in GeoTIFF format with file sizes ranging between 60 MB and 1.7 GB. The file size did not have any discernible impact on the time it took to register. The average registration took about 61 ms, meaning that registering nearly 1000 datasets per minute is possible.

The tests for the **generation of mosaics** were performed on a virtual machine with one CPU core allocated and 4 GB of RAM. Yet again, the input data were IKONOS scenes in GeoTIFF format.

Datasets	Data Type	Files	Input File Size	Tiles Generated	Time	GB per minute
IKONOS multispectral	4-band 16-bit	68	8.9 GB	8.819	10 m	0.89 GB
IKONOS panchromatic	1-band 16-bit	68	35.1 GB	126.750	1:05 h	0.54 GB
IKONOS pansharpened	3-band 8-bit	68	52.7 GB	126.750	1:46 h	0.49 GB

As the results show the file size of the input files has a certain impact on performance, but the effect seems to level off.

Regarding the performance of the services there are many influence factors:

- the hardware configuration of the machine
- the network connection bandwidth

- the database configuration (SQLite or PostGIS)
- the format and size of the raster data files
- the processing steps necessary to fulfill the request (e.g. resampling, reprojection)
- the coverage type (processing referenceable grid coverages is considerably more expensive than processing rectified grid coverages)
- the setup of IDM components (if any)

For hints on improving performance of the services see *Hardware Guidelines* (page 18) and *Data Preparation and Supported Data Formats* (page 59).

The Webclient Interface

Table of Contents

- *The Webclient Interface* (page 71)
 - *Enable the Webclient Interface* (page 71)
 - *Using the webclient interface* (page 71)

The webclient interface is an application running in the browser and provides a preview of all Datasets in a specified Dataset Series. It uses an [OpenLayers](#)¹⁰⁴ display to show a WMS view of the datasets within a map context. The background map tiles are provided by [EOX](#)¹⁰⁵.

It can further be used to provide a download mechanism for registered datasets.

Enable the Webclient Interface

To enable the webclient interface, several adjustments have to be made to the instances `settings.py` and `urls.py`.

First off, the `eoxserver.webclient` has to be inserted in the `INSTALLED_APPS` option of your `settings.py`. As the interface also requires several static files like style-sheets and script files, the option `STATIC_URL` has to be set to a path the webserver is able to serve, for example `/static/`. The static media files are located under `path/to/eoxserver/webclient/static` and can be collected via the `collectstatic` command¹⁰⁶.

To finally enable the webclient, a proper URL scheme has to be set up in `urls.py`. The following lines would enable the index and the webclient view on the URL `www.yourdomain.com/client`.

```
urlpatterns = patterns('',
    ...
    url(r'^client/', include("eoxserver.webclient.urls")),
    ...
)
```

Using the webclient interface

The webclient interface can be accessed via the given URL in `urls.py` as described in the instructions above, whereas the URL `www.yourdomain.com/client` would open an index view, displaying links to the webclient for every dataset series registered in the system. To view the webclient for a specific dataset series, use this URL: `www.yourdomain.com/client/<EOID>` where `<EOID>` is the EO-ID of the dataset series you want to inspect.

¹⁰⁴ <http://openlayers.org/>

¹⁰⁵ <https://maps.eox.at/>

¹⁰⁶ <https://docs.djangoproject.com/en/1.8/ref/contrib/staticfiles/#collectstatic>



Fig. 1.11: The webclient showing the contents of the autotest instance.

The map can be panned with via mouse dragging or the map-moving buttons in the upper left of the screen. Alternatively, the arrow keys can be used. The zoomlevel can be adjusted with the mouse scrolling wheel or the zoom-level buttons located directly below the pan control buttons.

A click on the small “+” sign on the upper right of the screen reveals the layer switcher control, where the preview and outline layers of the dataset series can be switched on or off.

The upper menu allows to switch the visibility of the “Layers”, “Tools” and “About” panels. The “Layers” panel allows to set the visibility of all the enabled layers of the instance. This includes all non-empty collections and all coverages that are `visible` but not in a collection. Also the background and the overlay can be altered.

The “Tools” panel allows to draw bounding boxes, manage selections and trigger the download. In order to download, first at least one bounding box must be drawn. Afterwards the download icon is clickable.

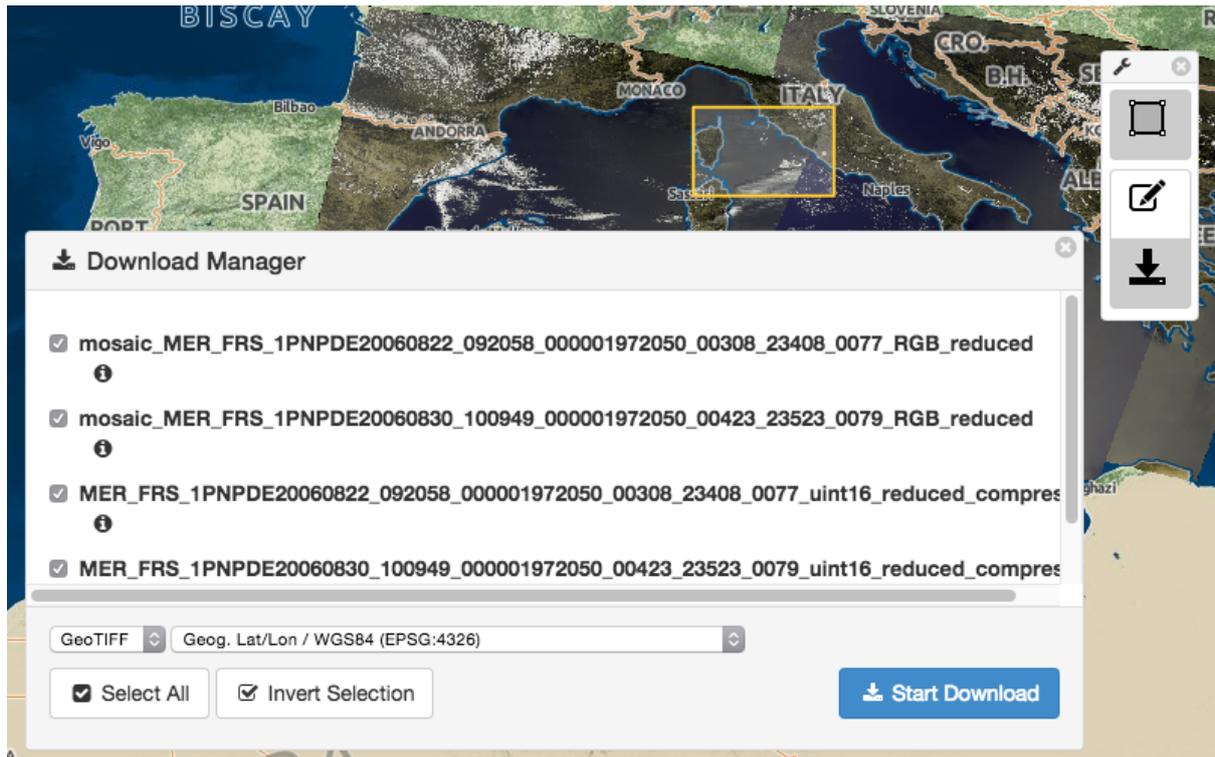


Fig. 1.12: *The download selection view.*

Upon clicking on the download icon, the download view is shown. It displays all the coverages available for download that are in the active layers and are intersecting with the spatio-temporal subsets. There, additional download options can be made:

- actually selecting coverages for download
- selecting an output format
- selecting an output projection

When all coverages to be downloaded are selected and all configuration is done a click on “Start Download” triggers the download of each coverage, subceted by the given spatial subsets.

The “About” panel shows general info of `EOxClient`¹⁰⁷, the software used to build the webclient.

In the bottom there is the timeslider widget. It is only shown if at least one layer is active. Like the map, it is “zoomable” (use the mousewheel when the mouse is over the timeslider) and “pannable” (the bar that contains the actual dates and times is the handle). It also allows to draw time intervals by dragging over the upper half of the widget. The upper half is also where coverages are displayed as colored dots or lines. The color of the dots/lines is the same as the color of its associated collection, whereas only active collections are visible on the timeslider.

¹⁰⁷ <https://github.com/EOX-A/EOxClient>

Hollow dots/lines mean that the coverage is currently not in the maps viewport. By clicking on a dot/line the map zooms to the coverages extent.

Identity Management System

Table of Contents

- *Identity Management System* (page 74)
 - *Installation and Configuration* (page 76)
 - * *Prerequisites* (page 76)
 - * *LDAP Directory* (page 76)
 - * *Authorisation Service* (page 77)
 - *XACML Policies for the Authorisation Service* (page 77)
 - * *General Configuration for CHARON services* (page 85)
 - *HTTP and SOAP Specific Components* (page 86)

The Identity Management System (IDMS) provides access control capabilities for security relevant data. The current IDMS supports EOxServer with a native security component for HTTP KVP and POST/XML protocol binding as well as external components for SOAP binding. The system is based on other free and open software projects, namely the [Charon Project](#)¹⁰⁸, the [Shibboleth framework](#)¹⁰⁹ and the [HMA Authentication Service](#)¹¹⁰. In the context of EOxServer, the SOAP support in the IDMS can be used to provide authentication and authorisation capabilities for the *SOAP Proxy* (page 100).

The IDMS uses two different schemes for authentication: The native EOxServer component relies on Shibboleth for Authentication, the SOAP components use the Charon framework.

The approach chosen for the SOAP part of the IDMS follows the OGC best practice document [User Management Interfaces for Earth Observation Services](#)¹¹¹ for the authentication concept. The authentication part is following the ideas of the [XACML data flow pattern](#)¹¹²: The IDMS authorisation part consists of a Policy Decision Point (PDP, here represented through the Charon Policy Management And Authentication Service) and the Policy Enforcement Point (PEP, represented through the Charon PEP Service). The following figure gives an overview of the IDMS SOAP part:

The HMA Authentication Service, or Security Token Service (STS), and the Charon PEP components were both modified in order to be compatible. This is a result of the ESA project [Open-standard Online Observation Service](#)¹¹³ (O3S). The STS now also supports SAML 2.0 security tokens, which the PEP components can interpret and validate. The IDMS supports trust relationships between identity providers and enforcement components on the basis of certificate stores.

The HTTP or native EOxServer part of the IDMS uses exactly the same scheme for authorisation as the SOAP part, but uses the Shibboleth federated identity management system for authentication.

Two requirements must be met to use the IDMS in this case:

- A Shibboleth Identity Provider (IdP) must be available for authentication
- A Shibboleth Service Provider (SP) must be installed and configured in an [Apache HTTP Server](#)¹¹⁴ to protect the EOxServer resource.

¹⁰⁸ <http://www.enviromatics.net/charon/>

¹⁰⁹ <http://shibboleth.internet2.edu/>

¹¹⁰ <http://wiki.services.eoportal.org/tiki-index.php?page=HMA+Authentication+Service>

¹¹¹ http://portal.opengeospatial.org/files/?artifact_id=40677

¹¹² http://docs.oasis-open.org/xacml/2.0/access_control-xacml-2.0-core-spec-os.pdf

¹¹³ <http://wiki.services.eoportal.org/tiki-index.php?page=O3S>

¹¹⁴ <http://httpd.apache.org/>

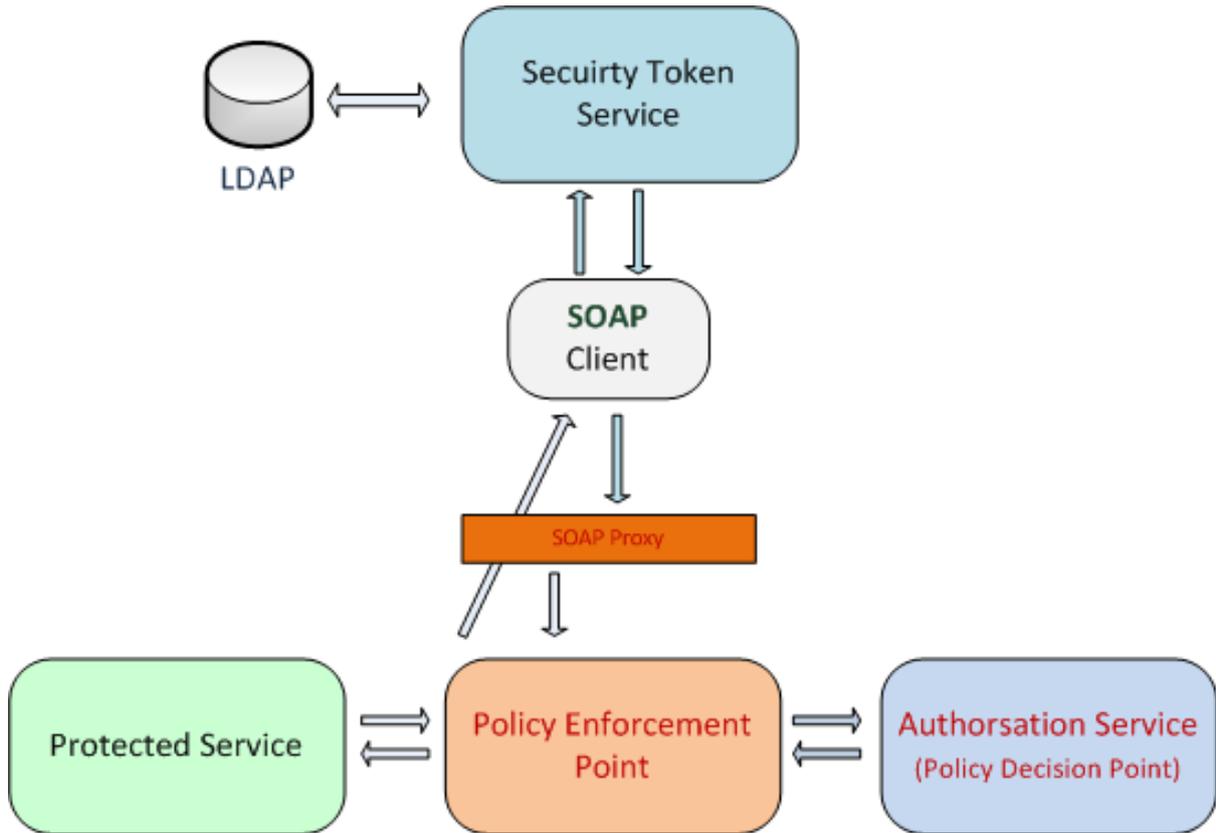


Fig. 1.13: IDMS SOAP Access Control Overview

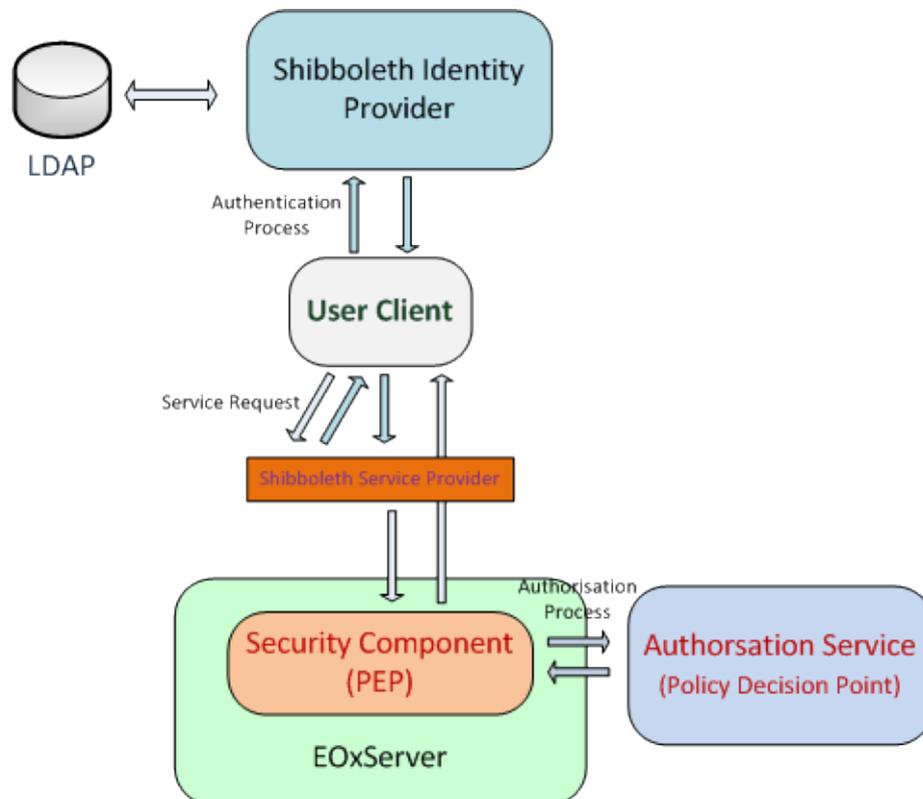


Fig. 1.14: IDMS EOxServer Access Control Overview

A user has to authenticate at an IdP in order to perform requests to an EOxServer with access control enabled. The IdP issues a SAML token which will be validated by the SP.

If the user is valid, the SP adds the user attributes received from the IdP to the HTTP Header of the original service requests and conveys it to the protected EOxServer instance. The whole process ensures, that only authenticated users can access the data and services provided by EOxServer. The attributes from Shibboleth are used by the EOxServer security components to make a XACMLAuthzDecisionQuery to the Charon Authorisation Service.

Installation and Configuration

The following services are needed both for the HTTP and the SOAP security part:

- Charon *Authorisation Service* (page 77).
- *LDAP Directory* (page 76).

Prerequisites

Download locations for the IDMS components:

- Shibboleth: <http://shibboleth.internet2.edu/downloads.html>
- CHARON Authorisation Service: <http://www.enviromatics.net/charon/> or <http://packages.eox.at/idm/>
- Security Token Service: <http://packages.eox.at/idm/>
- PEP Service: <http://packages.eox.at/idm/>
- EOxServer: <http://eoxserver.org/wiki/Download>

The following software is needed to run the IDMS:

- A *LDAP Directory* (page 76).
- Java¹¹⁵ JDK 6 or higher
- Apache Tomcat¹¹⁶ 6 or higher
- Apache Axis2¹¹⁷ 1.4.1 or higher
- MySQL¹¹⁸ 5
- Apache HTTP Server¹¹⁹ 2.x

The following software is needed to build the IDMS components:

- Java¹²⁰ SDK 6 or higher
- Apache Ant¹²¹ 1.6.2 or higher
- Apache Maven¹²² 2 or higher

LDAP Directory

The IDMS uses a LDAP directory to store user data (attributes, passwords, etc). You can use any directory implementation, supporting the Lightweight Directory Access Protocol (v3).

Known working implementations are:

¹¹⁵ <http://www.oracle.com/technetwork/java/index.html>

¹¹⁶ <http://tomcat.apache.org/>

¹¹⁷ <http://axis.apache.org/axis2/java/core/>

¹¹⁸ <http://dev.mysql.com/downloads/>

¹¹⁹ <http://httpd.apache.org/>

¹²⁰ <http://www.oracle.com/technetwork/java/index.html>

¹²¹ <http://ant.apache.org/>

¹²² <http://maven.apache.org/>

- [Apache Directory Service](#)¹²³
- [OpenLDAP](#)¹²⁴

A good graphical client for LDAP directories is the [Apache Directory Studio](#)¹²⁵.

Authorisation Service

Before installing the Authorisation Service, refer to the *General Configuration for CHARON services* (page 85).

The Authorisation Service is responsible for the authorisation of service requests. It makes use of XACML¹²⁶, a XML based language for access policies. The Authorisation Service is part of the CHAORN¹²⁷ project.

The Authorisation Service relies on a MySQL database to store all XACML policies. So in order to install the Authorisation Service, you first need to prepare a MySQL database:

- Install the MySQL database on your system.
- Change the `root` password. You can use the command line for this:

```
mysqladmin -u root password 'root' -p
```

- Run the SQL script bundle with the Authorisation Service in order to create the policy database:

```
mysql -u root -h localhost -p < PolicyAuthorService.sql
```

The Service needs the following additional dependencies in the `${AXIS2_HOME}\lib` folder:

- `mysql-connector-java-5.1.6.jar`
- `spring-2.5.1.jar`

The next step is deploying the Authorisation Service, therefore extract the ZIP archive into the directory of your `${AXIS2_HOME}`.

Now you have to configure the service. All configuration files are in the `${AXIS2_HOME}/WEB-INF/classes` folder and its sub-folders.

- Open the `PolicyAuthorService.properties` and change the `axisURL` parameter to the URL where you are actually deploying your service.
- You can change the database connection in the `config/GeoPDP.xml` configuration file if necessary.

To add new XACML policies to the Authorisation Service, refer to the *XACML Policies for the Authorisation Service* (page 77).

XACML Policies for the Authorisation Service

As mentioned before, the Charon Authorisation Service uses a MySQL database to store all XACML policies. The policies are stored in the database `policy_author` and the table `policy`. To add new policies, use an SQL client

```
INSERT INTO policy(policy) VALUES (' your xacml policy')
```

An XACML policy usually consists of a policy wide target and several specific rules. The three main identifiers are subjects, targets and actions. Subjects (or users) can be identified through the “asserted user attributes” which are provided by the Shibboleth framework. The EOxServer security components also provide an attribute `REMOTE_ADDR` for subjects, which contains the IP address of the user. The resource is mainly identified through the attribute `urn:oasis:names:tc:xacml:1.0:resource:resource-id`, which is the service address of the secured service in case of an secured SOAP service and the host name or a ID set in the configuration

¹²³ <http://directory.apache.org/>

¹²⁴ <http://openldap.org>

¹²⁵ <http://directory.apache.org/studio/>

¹²⁶ <http://www.oasis-open.org/committees/xacml/#XACML20>

¹²⁷ <http://www.enviromatics.net/charon/index.html>

in case of the EOxServer. The EOxServer also provides the attributes `serverName` (the host name) and `serviceType` (type of the service, i.e. `wcs` or `wms`). The action identifies the operation performed on the service, i.e. `getcapabilities` or `getcoverage`. In the following there are two example policies for the EOxServer WMS and WCS. Please note the comments inline.

A XACML policy to permit a user “`wms_user`” full access to the EOxServer WMS:

```
<?xml version="1.0" encoding="UTF-8"?>
<Policy
  xsi:schemalocation="urn:oasis:names:tc:xacml:2.0:policy:schema:os http://docs.
  ↪oasis-open.org/xacml/access_control-xacml-2.0-policy-schema-os.xsd"
  PolicyId="wms_user_policy"
  RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0:rule-combining-
  ↪algorithm:permit-overrides"
  xmlns="urn:oasis:names:tc:xacml:2.0:policy:schema:os"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:ns="http://www.enviromatics.net/WS/
  ↪PolicyManagementAndAuthorisationService/types /2.0">

  <Target>
    <Subjects>
      <Subject>
        <!-- Here we specify the user who has access to the service. Default_
        ↪identifier is the uid attribute -->
        <SubjectMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
          <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">wms_
          ↪user</AttributeValue>
          <SubjectAttributeDesignator DataType="http://www.w3.org/2001/XMLSchema
          ↪#string" AttributeId="uid"/>
        </SubjectMatch>
      </Subject>
    </Subjects>
    <Resources>
      <Resource>
        <!-- The attribute urn:oasis:names:tc:xacml:1.0:resource:resource-id_
        ↪specifies the protected server (default is the hostname) -->
        <ResourceMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal
        ↪">
          <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
          ↪eoxserver.example.com</AttributeValue>
          <ResourceAttributeDesignator DataType="http://www.w3.org/2001/XMLSchema
          ↪#string" AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"/>
        </ResourceMatch>

        <!-- The attribute serviceType specifies the protected service (wms or_
        ↪wcs) -->
        <ResourceMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal
        ↪">
          <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">wms
          ↪</AttributeValue>
          <ResourceAttributeDesignator DataType="http://www.w3.org/2001/XMLSchema
          ↪#string" AttributeId="serviceType"/>
        </ResourceMatch>
      </Resource>
    </Resources>
  </Target>

  <!--
  In the following rules we allow the specified user to perform selected_
  ↪operations
  on the service.
  -->
```

```

<!--
GetCapabilities
-->

<Rule RuleId="PermitGetCapabilitiesCC" Effect="Permit">
  <Target>
    <Actions>
      <Action>
        <ActionMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-
↪equal">
          <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
↪GetCapabilities</AttributeValue>
          <ActionAttributeDesignator AttributeId="urn:oasis:names:tc:xacml:1.
↪0:action:action" DataType="http://www.w3.org/2001/XMLSchema#string"/>
        </ActionMatch>
      </Action>
    </Actions>
  </Target>
</Rule>

<Rule RuleId="PermitGetCapabilitiesSC" Effect="Permit">
  <Target>
    <Actions>
      <Action>
        <ActionMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-
↪equal">
          <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
↪getcapabilities</AttributeValue>
          <ActionAttributeDesignator AttributeId="urn:oasis:names:tc:xacml:1.
↪0:action:action" DataType="http://www.w3.org/2001/XMLSchema#string"/>
        </ActionMatch>
      </Action>
    </Actions>
  </Target>
</Rule>

<!--
GetMap
-->

<Rule RuleId="GetMapCC" Effect="Permit">
  <Target>
    <Actions>
      <Action>
        <ActionMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-
↪equal">
          <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
↪GetMap</AttributeValue>
          <ActionAttributeDesignator AttributeId="urn:oasis:names:tc:xacml:1.
↪0:action:action" DataType="http://www.w3.org/2001/XMLSchema#string"/>
        </ActionMatch>
      </Action>
    </Actions>
  </Target>
</Rule>

<Rule RuleId="GetMapSC" Effect="Permit">
  <Target>
    <Actions>
      <Action>
        <ActionMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-
↪equal">

```

```

        <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
↪getmap</AttributeValue>
        <ActionAttributeDesignator AttributeId="urn:oasis:names:tc:xacml:1.
↪0:action:action" DataType="http://www.w3.org/2001/XMLSchema#string"/>
        </ActionMatch>
    </Action>
</Actions>
</Target>
</Rule>

<!--
GetFeatureInfo
-->

<Rule RuleId="GetFeatureInfoCC" Effect="Permit">
<Target>
<Actions>
<Action>
    <ActionMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-
↪equal">
        <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
↪GetFeatureInfo</AttributeValue>
        <ActionAttributeDesignator AttributeId="urn:oasis:names:tc:xacml:1.
↪0:action:action" DataType="http://www.w3.org/2001/XMLSchema#string"/>
        </ActionMatch>
    </Action>
</Actions>
</Target>
</Rule>

<Rule RuleId="GetFeatureInfoSC" Effect="Permit">
<Target>
<Actions>
<Action>
    <ActionMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-
↪equal">
        <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
↪getfeatureinfo</AttributeValue>
        <ActionAttributeDesignator AttributeId="urn:oasis:names:tc:xacml:1.
↪0:action:action" DataType="http://www.w3.org/2001/XMLSchema#string"/>
        </ActionMatch>
    </Action>
</Actions>
</Target>
</Rule>

<!--
DescribeLayer
-->

<Rule RuleId="DescribeLayerCC" Effect="Permit">
<Target>
<Actions>
<Action>
    <ActionMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-
↪equal">
        <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
↪DescribeLayer</AttributeValue>
        <ActionAttributeDesignator AttributeId="urn:oasis:names:tc:xacml:1.
↪0:action:action" DataType="http://www.w3.org/2001/XMLSchema#string"/>
        </ActionMatch>
    </Action>
</Actions>
</Target>
</Rule>

```

```

</Target>
</Rule>

<Rule RuleId="DescribeLayerSC" Effect="Permit">
  <Target>
    <Actions>
      <Action>
        <ActionMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-
↪equal">
          <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
↪describelayer</AttributeValue>
          <ActionAttributeDesignator AttributeId="urn:oasis:names:tc:xacml:1.
↪0:action:action" DataType="http://www.w3.org/2001/XMLSchema#string"/>
        </ActionMatch>
      </Action>
    </Actions>
  </Target>
</Rule>

<!--
GetLegendGraphic
-->

<Rule RuleId="GetLegendGraphicCC" Effect="Permit">
  <Target>
    <Actions>
      <Action>
        <ActionMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-
↪equal">
          <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
↪GetLegendGraphic</AttributeValue>
          <ActionAttributeDesignator AttributeId="urn:oasis:names:tc:xacml:1.
↪0:action:action" DataType="http://www.w3.org/2001/XMLSchema#string"/>
        </ActionMatch>
      </Action>
    </Actions>
  </Target>
</Rule>

<Rule RuleId="GetLegendGraphicSC" Effect="Permit">
  <Target>
    <Actions>
      <Action>
        <ActionMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-
↪equal">
          <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
↪getlegendgraphic</AttributeValue>
          <ActionAttributeDesignator AttributeId="urn:oasis:names:tc:xacml:1.
↪0:action:action" DataType="http://www.w3.org/2001/XMLSchema#string"/>
        </ActionMatch>
      </Action>
    </Actions>
  </Target>
</Rule>

<!--
GetStyles
-->

<Rule RuleId="GetStylesCC" Effect="Permit">
  <Target>
    <Actions>
      <Action>

```

```

        <ActionMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-
↪equal">
            <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
↪GetStyles</AttributeValue>
            <ActionAttributeDesignator AttributeId="urn:oasis:names:tc:xacml:1.
↪0:action:action" DataType="http://www.w3.org/2001/XMLSchema#string"/>
        </ActionMatch>
    </Action>
</Actions>
</Target>
</Rule>

<Rule RuleId="GetStylesSC" Effect="Permit">
<Target>
    <Actions>
    <Action>
        <ActionMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-
↪equal">
            <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
↪getstyles</AttributeValue>
            <ActionAttributeDesignator AttributeId="urn:oasis:names:tc:xacml:1.
↪0:action:action" DataType="http://www.w3.org/2001/XMLSchema#string"/>
        </ActionMatch>
    </Action>
    </Actions>
</Target>
</Rule>
</Policy>

```

A XACML policy to permit a user “wcs_user” full access to the EOxServer WCS:

```

<?xml version="1.0" encoding="UTF-8"?>
<Policy
    xsi:schemalocation="urn:oasis:names:tc:xacml:2.0:policy:schema:os http://docs.
↪oasis-open.org/xacml/access_control-xacml-2.0-policy-schema-os.xsd"
    PolicyId="wcs_user_policy"
    RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0:rule-combining-
↪algorithm:permit-overrides"
    xmlns="urn:oasis:names:tc:xacml:2.0:policy:schema:os"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:ns="http://www.enviromatics.net/WS/
↪PolicyManagementAndAuthorisationService/types /2.0">

    <Target>
    <Subjects>
    <Subject>
        <!-- Here we specify the user who has access to the service. Default_
↪identifier is the uid attribute -->
        <SubjectMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
            <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">wcs_
↪user</AttributeValue>
            <SubjectAttributeDesignator DataType="http://www.w3.org/2001/XMLSchema
↪#string" AttributeId="uid"/>
        </SubjectMatch>
    </Subject>
    </Subjects>
    <Resources>
    <Resource>
        <!-- The attribute urn:oasis:names:tc:xacml:1.0:resource:resource-id_
↪specifies the protected server (default is the hostname) -->
        <ResourceMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal
↪">

```

```

        <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
↪eoxserver.example.com</AttributeValue>
        <ResourceAttributeDesignator DataType="http://www.w3.org/2001/XMLSchema
↪#string" AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"/>
        </ResourceMatch>

        <!-- The attribute serviceType specifies the protected service (wms or
↪wcs) -->
        <ResourceMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal
↪">
            <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">wcs
↪</AttributeValue>
            <ResourceAttributeDesignator DataType="http://www.w3.org/2001/XMLSchema
↪#string" AttributeId="serviceType"/>
        </ResourceMatch>
    </Resource>
</Resources>
</Target>

<!--
GetCapabilities
-->

<Rule RuleId="PermitGetCapabilitiesCC" Effect="Permit">
<Target>
    <Actions>
    <Action>
        <ActionMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-
↪equal">
            <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
↪GetCapabilities</AttributeValue>
            <ActionAttributeDesignator AttributeId="urn:oasis:names:tc:xacml:1.
↪0:action:action" DataType="http://www.w3.org/2001/XMLSchema#string"/>
        </ActionMatch>
    </Action>
    </Actions>
</Target>
</Rule>

<Rule RuleId="PermitGetCapabilitiesSC" Effect="Permit">
<Target>
    <Actions>
    <Action>
        <ActionMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-
↪equal">
            <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
↪getcapabilities</AttributeValue>
            <ActionAttributeDesignator AttributeId="urn:oasis:names:tc:xacml:1.
↪0:action:action" DataType="http://www.w3.org/2001/XMLSchema#string"/>
        </ActionMatch>
    </Action>
    </Actions>
</Target>
</Rule>

<!--
DescribeCoverage
-->

<Rule RuleId="DescribeCoverageCC" Effect="Permit">
<Target>

```

```

    <Actions>
    <Action>
      <ActionMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-
↪equal">
        <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
↪DescribeCoverage</AttributeValue>
        <ActionAttributeDesignator AttributeId="urn:oasis:names:tc:xacml:1.
↪0:action:action" DataType="http://www.w3.org/2001/XMLSchema#string"/>
        </ActionMatch>
      </Action>
    </Actions>
  </Target>
</Rule>

<Rule RuleId="DescribeCoverageSC" Effect="Permit">
  <Target>
    <Actions>
    <Action>
      <ActionMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-
↪equal">
        <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
↪describecoverage</AttributeValue>
        <ActionAttributeDesignator AttributeId="urn:oasis:names:tc:xacml:1.
↪0:action:action" DataType="http://www.w3.org/2001/XMLSchema#string"/>
        </ActionMatch>
      </Action>
    </Actions>
  </Target>
</Rule>

<!--
  GetCoverage
-->

<Rule RuleId="DescribeCoverageCC" Effect="Permit">
  <Target>
    <Actions>
    <Action>
      <ActionMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-
↪equal">
        <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
↪GetCoverage</AttributeValue>
        <ActionAttributeDesignator AttributeId="urn:oasis:names:tc:xacml:1.
↪0:action:action" DataType="http://www.w3.org/2001/XMLSchema#string"/>
        </ActionMatch>
      </Action>
    </Actions>
  </Target>
</Rule>

<Rule RuleId="GetCoverageSC" Effect="Permit">
  <Target>
    <Actions>
    <Action>
      <ActionMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-
↪equal">
        <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
↪getcoverage</AttributeValue>
        <ActionAttributeDesignator AttributeId="urn:oasis:names:tc:xacml:1.
↪0:action:action" DataType="http://www.w3.org/2001/XMLSchema#string"/>
        </ActionMatch>
      </Action>
    </Actions>
  </Target>
</Rule>

```

```

</Target>
</Rule>

<!--
DescribeEOCoverageSet
-->

<Rule RuleId="DescribeEOCoverageSetCC" Effect="Permit">
  <Target>
    <Actions>
      <Action>
        <ActionMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-
↪equal">
          <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
↪DescribeEOCoverageSet</AttributeValue>
          <ActionAttributeDesignator AttributeId="urn:oasis:names:tc:xacml:1.
↪0:action:action" DataType="http://www.w3.org/2001/XMLSchema#string"/>
        </ActionMatch>
      </Action>
    </Actions>
  </Target>
</Rule>

<Rule RuleId="DescribeEOCoverageSetSC" Effect="Permit">
  <Target>
    <Actions>
      <Action>
        <ActionMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-
↪equal">
          <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
↪describeeocoverageset</AttributeValue>
          <ActionAttributeDesignator AttributeId="urn:oasis:names:tc:xacml:1.
↪0:action:action" DataType="http://www.w3.org/2001/XMLSchema#string"/>
        </ActionMatch>
      </Action>
    </Actions>
  </Target>
</Rule>
</Policy>

```

General Configuration for CHARON services

- The Charon services need the `acs-xbeans-1.0.jar` dependency in the `\lib` folder of your Axis2 installation (presumably the `webapps/axis2` of your Apache Tomcat installation).
- Furthermore, you have to activate the `EIGSecurityHandler` in the **Global Modules** section of your axis2 configuration (`${AXIS2_HOME}/WEB-INF/conf/axis2.xml`).
- You may configure the logging for the services in the Log4J configuration file (`${AXIS2_HOME}/WEB-INF/classes/log4j.properties`).

Both, the Security Token Service and the PEP service make use of Java Keystores: The IDMS uses Keystores to store a) the certificate used by the Security Token Service for signing SAML tokens and b) the public keys of those authenticating authorities trusted by the Policy Enforcement Point. The `keytool` of the Java distribution can be used to create and manipulate Java Keystores:

- The following command creates a new Keystore with the password `:secret:` and a suitable key pair with the alias `:authenticate:` for the Security Token Service:

```
keytool -genkey -alias authenticate -keyalg RSA -keystore
keystore.jks -storepass secret -validity 360
```

- The following command exports the public certificate from a key pair :authenticate: to the file authn.crt:

```
keytool -export -alias authenticate -file authn.crt -keystore
keystore.jks
```

- The following command imports a certificate to a Keystore:

```
keytool -import -alias trusted_sts -file authn.crt -keystore
keystore.jks
```

You can use the Apache HTTP Server as a proxy, it will enable your services running in Tomcat to be accessible over the Apache server. This can be useful when your services have to be accessible over the HTTP standard port 80:

- First you have to enable mod_proxy_ajp and mod_proxy.
- Create a virtual host in your httpd.conf:

```
<VirtualHost *:80>
  ServerName server.example.com

  <Proxy *>
    AddDefaultCharset Off
    Order deny,allow
    Allow from all
  </Proxy>

  ProxyPass /services/AuthenticationService ajp://localhost:8009/
↪axis2/services/AuthenticationService
  ProxyPassReverse /services/AuthenticationService ajp://
↪localhost:8009/axis2/services/AuthenticationService

</VirtualHost>
```

- The ProxyPass and ProxyPassReverse directives have to point to your services. Please note that the Tomcat server hosting your services must have the AJP interface enabled.

HTTP and SOAP Specific Components

For the installation and configuration please refer to the HTTP or SOAP specific documentation:

HTTP Components

Table of Contents

- *HTTP Components* (page 86)
 - *Shibboleth Identity Provider* (page 87)
 - *Shibboleth Service Provider* (page 92)
 - *Configure Shibboleth SP and IdP* (page 95)
 - *Configure the EOxServer Security Components* (page 96)
 - * *General Configuration Options* (page 96)
 - * *Adding new Subject attributes to the EOxServer Security Components* (page 97)

The following services are needed for the HTTP security part:

- Charon Authorisation Service
- Shibboleth Service Provider

- Shibboleth Identity Provider
- EOxServer

To install and configure the HTTP security components, you have to follow these steps:

1. Install the Charon *Authorisation Service* (page 77).
2. Install the *Shibboleth Identity Provider* (page 87).
3. Install the *Shibboleth Service Provider* (page 92).
4. Follow the documentation of section *Configure Shibboleth SP and IdP* (page 95).
5. Follow the documentation of section *Configure the EOxServer Security Components* (page 96).

Shibboleth Identity Provider

The Shibboleth IdP is implemented as an Java Servlet, thus it needs an installed Servlet container. The Shibboleth project offers [an installation manual for the Shibboleth IdP on their website](#)¹²⁸. This documentation will provide help for the basic configuration to get the authentication process working with your EOxServer instance and also the installation process for the use with Tomcat and Apache HTTPD. Before you begin with your installation, set up your Tomcat servlet container and install and configure an LDAP service.

Important URLs for your Shibboleth IDP:

- Status message: `https://${IDPHOST}/idp/profile/Status`
- Information page: `https://${IDPHOST}/idp/status`
- Metadata: `https://${IDPHOST}/idp/profile/Metadata/SAML`

Warning: IdP resource paths are case sensitive!

- [Download](#)¹²⁹ the IdP and unzip the archive.
- Run either `./install.sh` (on Linux/Unix systems) or `install.bat` (on Windows systems).
- Follow the on-screen instructions of the script.

Your `${IDP_HOME}` directory contains the following directories:

- `bin`: This directory contains various tools useful in running, testing, or deploying the IdP
- `conf`: This directory contains all the configuration files for the IdP
- `credentials`: This is where the IdP's signing and encryption credential, called `idp.key` and `idp.crt`, is stored
- `lib`: This directory contains various code libraries used by the tools in `bin/`
- `logs`: This directory contains the log files for the IdP. **Don't forget to make this writeable for your Tomcat server!**
- `metadata`: This is the directory in which the IdP will store its metadata, in a file called `idp-metadata.xml`. It is recommended you store any other retrieved metadata here as well.
- `war`: This contains the web application archive (war) file that you will deploy into the servlet container

The next step is to deploy the IdP into your Tomcat:

- Increase the memory reserved for Tomcat. Recommended values are `-Xmx512m -XX:MaxPermSize=128m`.
- Add the libraries endorsed by the Shibboleth project to your endorsed Tomcat directories: `-Djava.endorsed.dirs=${IDP_HOME}/lib/endorsed/`
- Create a new XML document `idp.xml` in `${TOMCAT_HOME}/conf/Catalina/localhost/`.

¹²⁸ <https://wiki.shibboleth.net/confluence/display/SHIB2/IdPInstall>

¹²⁹ <http://shibboleth.internet2.edu/downloads.html>

- Insert the following content:

```
<Context docBase="${IDP_HOME}/war/idp.war"
  privileged="true"
  antiResourceLocking="false"
  antiJARLocking="false"
  unpackWAR="false"
  swallowOutput="true" />
```

- Don't forget to replace `${IDP_HOME}` with the appropriate path.

To use the Apache HTTP server as a proxy for your IdP, you have to generate a certificate and a key file for SSL/TLS first.

- Generate a private key:

```
openssl genrsa -des3 -out server.key 1024
```

- Generate a CSR (Certificate Signing Request):

```
openssl req -new -key server.key -out server.csr
```

- Make a copy from the the original server key:

```
cp server.key copy_of_server.key
```

- Remove the Passphrase from your Key:

```
openssl rsa -in copy_of_server.key -out server.key
```

- Generating a Self-Signed Certificate:

```
openssl x509 -req -days 365 -in server.csr -signkey server.key
-out server.crt
```

The next step is to configure your Apache HTTP Server:

- First you have to enable `mod_proxy_ajp`, `mod_proxy` and `mod_ssl`.
- Create a new configuration file for your SSL hosts (for example `ssl_hosts.conf`).
- Add a new virtual host in your new hosts file. Please note the comments in the virtual host configuration.

```
<VirtualHost _default_:443>

  # Set appropriate document root here
  DocumentRoot "/var/www/"

  # Set your designated IDP host here
  ServerName ${IDP_HOST}

  # Set your designated logging directory here
  ErrorLog logs/ssl_error_log
  TransferLog logs/ssl_access_log
  LogLevel warn

  SSLEngine on

  SSLProtocol all -SSLv2

  # Important: mod_ssl should not verify the provided certificates
  SSLVerifyClient optional_no_ca

  SSLCipherSuite ALL:!ADH:!EXPORT:!SSLv2:RC4+RSA:+HIGH:+MEDIUM:+LOW

  # Set the correct paths to your certificate and key here
  SSLCertificateFile ${IDP_HOST_CERTIFICATE}
  SSLCertificateKeyFile ${IDP_HOST_CERTIFICATE_KEY}
```

```

<Files ~ "\.(cgi|shtml|phtml|php3?)$" >
    SSLOptions +StdEnvVars
</Files>
<Directory "/var/www/cgi-bin">
    SSLOptions +StdEnvVars
</Directory>

# AJP Proxy to your IDP servlet
ProxyPass /idp/ ajp://localhost:8009/idp/
ProxyPassReverse /idp ajp://localhost:8009/idp

SetEnvIf User-Agent ".*MSIE.*" nokeepalive ssl-unclean-shutdown_
↪downgrade-1.0 force-response-1.0

CustomLog logs/ssl_request_log "%t %h %{SSL_PROTOCOL}x %{SSL_
↪CIPHER}x \"%r\" %b"
</VirtualHost>

```

- Restart your HTTP server.

The next step is to configure our IdP Service with an LDAP service. Please keep in mind that this documentation can only give a small insight into all configuration possibilities of Shibboleth.

Open the handler.xml

- Add a new LoginHandler

```

<LoginHandler xsi:type="UsernamePassword"
    jaasConfigurationLocation="file://${IDP_HOME}/conf/login.
↪config">
    <AuthenticationMethod>urn:oasis:names:tc:SAML:2.
↪0:ac:classes:PasswordProtectedTransport</AuthenticationMethod>
</LoginHandler>

```

- Remove (or comment out) the LoginHandler element of type RemoteUser.

Open the login.config and comment out or delete the other entries that might exist. Add your own LDAP configuration:

```

ShibUserPassAuth {
    edu.vt.middleware.ldap.jaas.LdapLoginModule required
    host="${LDAP_HOST}"
    port="${LDAP_PORT}"
    serviceUser="${LDAP_ADMIN}"
    serviceCredential="${LDAP_ADMIN_PASSWORD}"
    base="${LDAP_USER_BASE}"
    ssl="false"
    userField="uid"
    subtreeSearch="true";
};

```

Enable your LDAP directory as attribute provider:

- Open the attribute-resolver.xml.
- Add your LDAP:

```

<resolver:DataConnector id="localLDAP" xsi:type="LDAPDirectory"
    xmlns="urn:mace:shibboleth:2.0:resolver:dc" ldapURL="ldap://$
↪{LDAP_HOST}:${LDAP_PORT}"
    baseDN="${LDAP_USER_BASE}" principal="${LDAP_ADMIN}"
    principalCredential="${LDAP_ADMIN_PASSWORD}">

```

```

<FilterTemplate>
  <![CDATA[
    (uid=$requestContext.principalName)
  ]]>
</FilterTemplate>
</resolver:DataConnector>

```

- Configure the IdP to retrieve the attributes by adding new attribute definitions:

```

<resolver:AttributeDefinition id="transientId" xsi:type="ad:TransientId"
↪">
  <resolver:AttributeEncoder xsi:type="enc:SAML1StringNameIdentifier"
    nameFormat="urn:mace:shibboleth:1.0:nameIdentifier"/>
  <resolver:AttributeEncoder xsi:type="enc:SAML2StringNameID"
    nameFormat="urn:oasis:names:tc:SAML:2.0:nameid-format:transient"
↪"/>
</resolver:AttributeDefinition>

<resolver:AttributeDefinition id="displayName" xsi:type="Simple"
  xmlns="urn:mace:shibboleth:2.0:resolver:ad" sourceAttributeID=
↪"displayName">
  <resolver:Dependency ref="localLDAP"/>
  <resolver:AttributeEncoder xsi:type="SAML1String"
    xmlns="urn:mace:shibboleth:2.0:attribute:encoder"
    name="urn:mace:dir:attribute-def:displayName"/>
  <resolver:AttributeEncoder xsi:type="SAML2String"
    xmlns="urn:mace:shibboleth:2.0:attribute:encoder"
    name="urn:oid:2.16.840.1.113730.3.1.241" friendlyName=
↪"displayName"/>
</resolver:AttributeDefinition>

<resolver:AttributeDefinition id="givenName" xsi:type="Simple"
  xmlns="urn:mace:shibboleth:2.0:resolver:ad" sourceAttributeID=
↪"givenName">
  <resolver:Dependency ref="localLDAP"/>
  <resolver:AttributeEncoder xsi:type="SAML1String"
    xmlns="urn:mace:shibboleth:2.0:attribute:encoder"
    name="urn:mace:dir:attribute-def:givenName"/>
  <resolver:AttributeEncoder xsi:type="SAML2String"
    xmlns="urn:mace:shibboleth:2.0:attribute:encoder" name=
↪"urn:oid:2.5.4.42"
    friendlyName="givenName"/>
</resolver:AttributeDefinition>

<resolver:AttributeDefinition id="description" xsi:type="Simple"
  xmlns="urn:mace:shibboleth:2.0:resolver:ad" sourceAttributeID=
↪"description">
  <resolver:Dependency ref="localLDAP"/>
  <resolver:AttributeEncoder xsi:type="SAML1String"
    xmlns="urn:mace:shibboleth:2.0:attribute:encoder"
    name="urn:mace:dir:attribute-def:description"/>
  <resolver:AttributeEncoder xsi:type="SAML2String"
    xmlns="urn:mace:shibboleth:2.0:attribute:encoder" name=
↪"urn:oid:2.5.4.13"
    friendlyName="description"/>
</resolver:AttributeDefinition>

<resolver:AttributeDefinition id="cn" xsi:type="Simple"
  xmlns="urn:mace:shibboleth:2.0:resolver:ad" sourceAttributeID="cn">
  <resolver:Dependency ref="localLDAP"/>
  <resolver:AttributeEncoder xsi:type="SAML1String"
    xmlns="urn:mace:shibboleth:2.0:attribute:encoder" name=
↪"urn:mace:dir:attribute-def:cn"/>

```

```

    <resolver:AttributeEncoder xsi:type="SAML2String"
      xmlns="urn:mace:shibboleth:2.0:attribute:encoder" name=
↪ "urn:oid:2.5.4.3"
      friendlyName="cn"/>
  </resolver:AttributeDefinition>

  <resolver:AttributeDefinition id="sn" xsi:type="Simple"
    xmlns="urn:mace:shibboleth:2.0:resolver:ad" sourceAttributeID="sn">
    <resolver:Dependency ref="localLDAP"/>
    <resolver:AttributeEncoder xsi:type="SAML1String"
      xmlns="urn:mace:shibboleth:2.0:attribute:encoder" name=
↪ "urn:mace:dir:attribute-def:sn"/>
    <resolver:AttributeEncoder xsi:type="SAML2String"
      xmlns="urn:mace:shibboleth:2.0:attribute:encoder" name=
↪ "urn:oid:2.5.4.4"
      friendlyName="sn"/>
  </resolver:AttributeDefinition>

  <resolver:AttributeDefinition id="uid" xsi:type="Simple"
    xmlns="urn:mace:shibboleth:2.0:resolver:ad" sourceAttributeID="uid"
↪ ">
    <resolver:Dependency ref="localLDAP"/>
    <resolver:AttributeEncoder xsi:type="SAML1String"
      xmlns="urn:mace:shibboleth:2.0:attribute:encoder" name=
↪ "urn:mace:dir:attribute-def:uid"/>
    <resolver:AttributeEncoder xsi:type="SAML2String"
      xmlns="urn:mace:shibboleth:2.0:attribute:encoder" name=
↪ "urn:oid:2.5.4.45"
      friendlyName="uid"/>
  </resolver:AttributeDefinition>

```

Add the new attributes to your `attribute-filter.xml` by adding a new `AttributeFilterPolicy`:

```

<afp:AttributeFilterPolicy id="attribFilter">
  <afp:PolicyRequirementRule xsi:type="basic:ANY"/>

  <afp:AttributeRule attributeID="givenName">
    <afp:PermitValueRule xsi:type="basic:ANY"/>
  </afp:AttributeRule>

  <afp:AttributeRule attributeID="displayName">
    <afp:PermitValueRule xsi:type="basic:ANY"/>
  </afp:AttributeRule>

  <afp:AttributeRule attributeID="description">
    <afp:PermitValueRule xsi:type="basic:ANY"/>
  </afp:AttributeRule>

  <afp:AttributeRule attributeID="cn">
    <afp:PermitValueRule xsi:type="basic:ANY"/>
  </afp:AttributeRule>

  <afp:AttributeRule attributeID="sn">
    <afp:PermitValueRule xsi:type="basic:ANY"/>
  </afp:AttributeRule>

  <afp:AttributeRule attributeID="uid">
    <afp:PermitValueRule xsi:type="basic:ANY"/>
  </afp:AttributeRule>

</afp:AttributeFilterPolicy>

```

Now you have to check if the generated metadata is correct. To do this, open the `idp-metadata.xml` file.

Known issues are:

- Incorrect ports: For example port 8443 at the AttributeService Bindings instead of no specific port.
- Wrong X509Certificate for Attribute Resolver. Use your previously generated SSL/TLS `${IDP_HOST_CERTIFICATE}` instead.

After this, restart your Shibboleth IdP.

Shibboleth Service Provider

The installation procedure for the Shibboleth SP is different for all supported Operating Systems. The project describes the different installation methods in an [own installation manual](#)¹³⁰. This documentation will provide help for the basic configuration to get the authentication process working with your EOxServer instance.

Important URLs for your Shibboleth SP:

- Status page: `https://${SPHOST}/Shibboleth.sso/Status`
- Metadata: `https://${SPHOST}/Shibboleth.sso/Metadata`
- Session summary: `https://${SPHOST}/Shibboleth.sso/Session`
- Local logout: `https://${SPHOST}/Shibboleth.sso/Logout`

Warning: SP resource paths are case sensitive!

STEP 1

The Shibboleth SP has two relevant configuration files. We begin with the `attribute-map.xml` file, where we configure the mapping of the attributes received from the IdP to the secured service (in our case the EOxServer):

```
<Attributes xmlns="urn:mace:shibboleth:2.0:attribute-map" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <!-- First some useful eduPerson attributes that many sites might use. -->
  <Attribute name="urn:mace:dir:attribute-def:eduPersonPrincipalName" id="eppn">
    <AttributeDecoder xsi:type="ScopedAttributeDecoder"/>
  </Attribute>
  <Attribute name="urn:oid:1.3.6.1.4.1.5923.1.1.1.6" id="eppn">
    <AttributeDecoder xsi:type="ScopedAttributeDecoder"/>
  </Attribute>
  <Attribute name="urn:mace:dir:attribute-def:eduPersonScopedAffiliation" id="
  ↪affiliation">
    <AttributeDecoder xsi:type="ScopedAttributeDecoder" caseSensitive="false"/>
  </Attribute>
  <Attribute name="urn:oid:1.3.6.1.4.1.5923.1.1.1.9" id="affiliation">
    <AttributeDecoder xsi:type="ScopedAttributeDecoder" caseSensitive="false"/>
  </Attribute>
  <Attribute name="urn:mace:dir:attribute-def:eduPersonAffiliation" id="unscoped-
  ↪affiliation">
    <AttributeDecoder xsi:type="StringAttributeDecoder" caseSensitive="false"/>
  </Attribute>
  <Attribute name="urn:oid:1.3.6.1.4.1.5923.1.1.1.1" id="unscoped-affiliation">
    <AttributeDecoder xsi:type="StringAttributeDecoder" caseSensitive="false"/>
  </Attribute>
  <Attribute name="urn:mace:dir:attribute-def:eduPersonEntitlement" id="
  ↪entitlement"/>
  <Attribute name="urn:oid:1.3.6.1.4.1.5923.1.1.1.7" id="entitlement"/>
</Attributes>
```

¹³⁰ <https://wiki.shibboleth.net/confluence/display/SHIB2/Installation>

```

<!-- A persistent id attribute that supports personalized anonymous access. -->

<!-- First, the deprecated/incorrect version, decoded as a scoped string: -->
<Attribute name="urn:mace:dir:attribute-def:eduPersonTargetedID" id="targeted-
↪id">
  <AttributeDecoder xsi:type="ScopedAttributeDecoder"/>
  <!-- <AttributeDecoder xsi:type="NameIDFromScopedAttributeDecoder"
↪formatter="$NameQualifier!$SPNameQualifier!$Name" defaultQualifiers="true"/> -->
</Attribute>

<!-- Second, an alternate decoder that will decode the incorrect form into the
↪newer form. -->
<!--
<Attribute name="urn:mace:dir:attribute-def:eduPersonTargetedID" id=
↪"persistent-id">
  <AttributeDecoder xsi:type="NameIDFromScopedAttributeDecoder" formatter="
↪$NameQualifier!$SPNameQualifier!$Name" defaultQualifiers="true"/>
</Attribute>
-->

<!-- Third, the new version (note the OID-style name): -->
<Attribute name="urn:oid:1.3.6.1.4.1.5923.1.1.1.10" id="persistent-id">
  <AttributeDecoder xsi:type="NameIDAttributeDecoder" formatter="
↪$NameQualifier!$SPNameQualifier!$Name" defaultQualifiers="true"/>
</Attribute>

<!-- Fourth, the SAML 2.0 NameID Format: -->
<Attribute name="urn:oasis:names:tc:SAML:2.0:nameid-format:persistent" id=
↪"persistent-id">
  <AttributeDecoder xsi:type="NameIDAttributeDecoder" formatter="
↪$NameQualifier!$SPNameQualifier!$Name" defaultQualifiers="true"/>
</Attribute>

<!--Examples of LDAP-based attributes, uncomment to use these... -->
<Attribute name="urn:mace:dir:attribute-def:cn" id="cn"/>
<Attribute name="urn:mace:dir:attribute-def:sn" id="sn"/>
<Attribute name="urn:mace:dir:attribute-def:givenName" id="givenName"/>
<Attribute name="urn:mace:dir:attribute-def:mail" id="mail"/>
<Attribute name="urn:mace:dir:attribute-def:telephoneNumber" id=
↪"telephoneNumber"/>
  <Attribute name="urn:mace:dir:attribute-def:title" id="title"/>
  <Attribute name="urn:mace:dir:attribute-def:initials" id="initials"/>
  <Attribute name="urn:mace:dir:attribute-def:description" id="description"/>
  <Attribute name="urn:mace:dir:attribute-def:carLicense" id="carLicense"/>
  <Attribute name="urn:mace:dir:attribute-def:departmentNumber" id=
↪"departmentNumber"/>
  <Attribute name="urn:mace:dir:attribute-def:displayName" id="displayName"/>
  <Attribute name="urn:mace:dir:attribute-def:employeeNumber" id="employeeNumber
↪"/>
  <Attribute name="urn:mace:dir:attribute-def:employeeType" id="employeeType"/>
  <Attribute name="urn:mace:dir:attribute-def:preferredLanguage" id=
↪"preferredLanguage"/>
  <Attribute name="urn:mace:dir:attribute-def:manager" id="manager"/>
  <Attribute name="urn:mace:dir:attribute-def:seeAlso" id="seeAlso"/>
  <Attribute name="urn:mace:dir:attribute-def:facsimileTelephoneNumber" id=
↪"facsimileTelephoneNumber"/>
  <Attribute name="urn:mace:dir:attribute-def:street" id="street"/>
  <Attribute name="urn:mace:dir:attribute-def:postOfficeBox" id="postOfficeBox"/>
  <Attribute name="urn:mace:dir:attribute-def:postalCode" id="postalCode"/>
  <Attribute name="urn:mace:dir:attribute-def:st" id="st"/>
  <Attribute name="urn:mace:dir:attribute-def:l" id="l"/>
  <Attribute name="urn:mace:dir:attribute-def:o" id="o"/>
  <Attribute name="urn:mace:dir:attribute-def:ou" id="ou"/>

```

```

    <Attribute name="urn:mace:dir:attribute-def:businessCategory" id=
↪"businessCategory"/>
    <Attribute name="urn:mace:dir:attribute-def:physicalDeliveryOfficeName" id=
↪"physicalDeliveryOfficeName"/>

    <Attribute name="urn:oid:2.5.4.3" id="cn"/>
    <Attribute name="urn:oid:2.5.4.4" id="sn"/>
    <Attribute name="urn:oid:2.5.4.42" id="givenName"/>
    <Attribute name="urn:oid:0.9.2342.19200300.100.1.3" id="mail"/>
    <Attribute name="urn:oid:2.5.4.20" id="telephoneNumber"/>
    <Attribute name="urn:oid:2.5.4.12" id="title"/>
    <Attribute name="urn:oid:2.5.4.43" id="initials"/>
    <Attribute name="urn:oid:2.5.4.13" id="description"/>
    <Attribute name="urn:oid:2.16.840.1.113730.3.1.1" id="carLicense"/>
    <Attribute name="urn:oid:2.16.840.1.113730.3.1.2" id="departmentNumber"/>
    <Attribute name="urn:oid:2.16.840.1.113730.3.1.3" id="employeeNumber"/>
    <Attribute name="urn:oid:2.16.840.1.113730.3.1.4" id="employeeType"/>
    <Attribute name="urn:oid:2.16.840.1.113730.3.1.39" id="preferredLanguage"/>
    <Attribute name="urn:oid:2.16.840.1.113730.3.1.241" id="displayName"/>
    <Attribute name="urn:oid:0.9.2342.19200300.100.1.10" id="manager"/>
    <Attribute name="urn:oid:2.5.4.34" id="seeAlso"/>
    <Attribute name="urn:oid:2.5.4.23" id="facsimileTelephoneNumber"/>
    <Attribute name="urn:oid:2.5.4.9" id="street"/>
    <Attribute name="urn:oid:2.5.4.18" id="postOfficeBox"/>
    <Attribute name="urn:oid:2.5.4.17" id="postalCode"/>
    <Attribute name="urn:oid:2.5.4.8" id="st"/>
    <Attribute name="urn:oid:2.5.4.7" id="l"/>
    <Attribute name="urn:oid:2.5.4.10" id="o"/>
    <Attribute name="urn:oid:2.5.4.11" id="ou"/>
    <Attribute name="urn:oid:2.5.4.15" id="businessCategory"/>
    <Attribute name="urn:oid:2.5.4.19" id="physicalDeliveryOfficeName"/>

    <Attribute name="urn:oid:2.5.4.45" id="uid"/>
</Attributes>

```

The next step is to edit the `shibboleth2.xml` file: Locate the element `ApplicationDefaults` and set the value of the attribute `entityID` to `${SP_HOST}\Shibboleth`.

STEP 2

The next step is to configure your Apache HTTP Server. To do this, you have to generate a certificate and a key file for your SSL/TLS Shibboleth SP Host first (see Shibboleth IdP section). Then add a virtual host to your Apache HTTP Server:

```

<VirtualHost _default_:443>

    # Include the apache22.conf from Shibboleth
    include ${SP_HOME}/apache22.conf

    # Set appropriate document root here
    DocumentRoot "/var/www/"

    # Set your designated IDP host here
    ServerName ${IDP_HOST}

    # Set your designated logging directory here
    ErrorLog logs/ssl_error_log
    TransferLog logs/ssl_access_log
    LogLevel warn

    SSLEngine on

    SSLProtocol all -SSLv2

```

```

# Important: mod_ssl should not verify the provided certificates
SSLVerifyClient optional_no_ca

SSLCipherSuite ALL:!ADH:!EXPORT:!SSLv2:RC4+RSA:+HIGH:+MEDIUM:+LOW

# Set the correct paths to your certificate and key here
SSLCertificateFile    ${SP_HOST_CERTIFICATE}
SSLCertificateKeyFile ${SP_HOST_CERTIFICATE_KEY}

<Files ~ "\.(cgi|shtml|phtml|php3?)$" >
    SSLOptions +StdEnvVars
</Files>
<Directory "/var/www/cgi-bin">
    SSLOptions +StdEnvVars
</Directory>

SetEnvIf User-Agent ".*MSIE.*" nokeepalive ssl-unclean-shutdown downgrade-1.0
↪force-response-1.0

CustomLog logs/ssl_request_log "%t %h %{SSL_PROTOCOL}x %{SSL_CIPHER}x \"%r\" %b
↪"

</VirtualHost>

```

STEP 3

Open `shibboleth2.xml` and change the `entityID` in the element `ApplicationDefaults` to your `${SP_HOST}`. Restart your SP and try to access your SP Metadata `https://${SPHOST}/Shibboleth.sso/Metadata`

Configure Shibboleth SP and IdP

- Download SP Metadata and store it locally as `${SP_METADATA_FILE}`.
- Open the `relying-party.xml` of the Shibboleth IdP and change the Metadata Provider entry to

```

<!-- MetadataProvider the combining other MetadataProviders -->
<metadata:MetadataProvider id="ShibbolethMetadata" xsi:type=
↪"metadata:ChainingMetadataProvider">

    <metadata:MetadataProvider id="IdPMD" xsi:type=
↪"metadata:ResourceBakedMetadataProvider">
        <!-- This is usually set correctly by the IdP installation
↪script -->
        <metadata:MetadataResource xsi:type=
↪"resource:FilesystemResource"
            file="${IDP_METADATA_FILE}"/>
    </metadata:MetadataProvider>

    <!-- This is the new MetadataProvider for your SP metadata -->
    <MetadataProvider id="URLMD" xsi:type="FilesystemMetadataProvider"
        xmlns="urn:mace:shibboleth:2.0:metadata"
        metadataFile="${SP_METADATA_FILE}">

        <MetadataFilter xsi:type="ChainingFilter" xmlns=
↪"urn:mace:shibboleth:2.0:metadata">
            <MetadataFilter xsi:type="EntityRoleWhiteList"
                xmlns="urn:mace:shibboleth:2.0:metadata">

```

```

        <RetainedRole>samlmd:SPSSODescriptor</RetainedRole>
      </MetadataFilter>
    </MetadataFilter>

    </MetadataProvider>
  </metadata:MetadataProvider>

```

- Add the `${SP_HOST_CERTIFICATE}` to your Java Keystore:

```
keytool -import -file ${SP_HOST_CERTIFICATE} -alias ${SP_HOST}
-keystore ${JAVA_JRE_HOME}\lib\security\cacerts
```

- Open `shibboleth2.xml` of your Shibboleth SP add a new SessionInitiator to the Sessions element:

```

<!-- Default example directs to a specific IdP's SSO service (favoring
↳SAML 2 over Shib 1). -->
<SessionInitiator type="Chaining" Location="/Login"
  isDefault="true" id="Intranet" relayState="cookie"
  entityID="https://{IDP_HOST}/idp/shibboleth">
  <SessionInitiator type="SAML2" acsIndex="1"
    template="bindingTemplate.html"/>
  <SessionInitiator type="Shib1" acsIndex="5"/>
</SessionInitiator>

```

- Then add a new MetadataProvider:

```

<!-- Chains together all your metadata sources. -->
<MetadataProvider type="Chaining">
  <MetadataProvider type="XML"
    uri="https://{IDP_HOST}/idp/profile/Metadata/
↳SAML"
    backingFilePath="federation-metadata.xml"
    reloadInterval="7200">
  </MetadataProvider>
</MetadataProvider>

```

Alternatively you can reference the metadata from your local IdP:

```

<!-- Chains together all your metadata sources. -->
<MetadataProvider type="Chaining">
  <MetadataProvider type="XML"
    path="${IDP_HOME}/metadata/idp-metadata.xml"
  </MetadataProvider>
</MetadataProvider>

```

- Restart your IdP, the SP and the Apache HTTPD

Configure the EOxServer Security Components

This section describes the configuration of the EOxServer security components.

General Configuration Options

The configuration of the EOxServer security components is done in the `eoxserver.conf` configuration file of your EOxServer instance. All security related configuration is done in the section `[services.auth.base]`:

- `pdp_type`: Determines the Policy Decision Point type; defaults to `none` which deactivates authorisation. Currently, only the type `charonpdp` is implemented.

- `authz_service`: The URL of the Authorisation Service.
- `attribute_mapping`: The file path to a dictionary with a mapping from identity attributes received from the Shibboleth IdP to a XACMLAuthzDecisionQuery. If the key is set to `default`, a standard dictionary is used.
- `serviceID`: Identifier for the EOxServer instance to an external Authorisation Service. Is used as resource ID in an XACMLAuthzDecisionQuery. If the key is set to `default`, the host name will be used.
- `allowLocal`: If set to `True`, the security components will allow access to requests from the local machine. *Use with care!*

Adding new Subject attributes to the EOxServer Security Components

In order to register new Subject attributes from your LDAP to the IDMS, you have to configure the Shibboleth IdP, the Shibboleth SP, and the EOxServer. Let's assume we want to add the new attribute `foo`.

Shibboleth IdP

Add a new `AttributeResolver` to your `attribute-resolver.xml` configuration file:

```
<resolver:AttributeDefinition id="foo" xsi:type="Simple"
  xmlns="urn:mace:shibboleth:2.0:resolver:ad" sourceAttributeID="description">
  <resolver:Dependency ref="localLDAP"/>
  <resolver:AttributeEncoder xsi:type="SAML1String"
    xmlns="urn:mace:shibboleth:2.0:attribute:encoder"
    name="urn:mace:dir:attribute-def:description"/>
  <resolver:AttributeEncoder xsi:type="SAML2String"
    xmlns="urn:mace:shibboleth:2.0:attribute:encoder" name="foo"
    friendlyName="foo"/>
</resolver:AttributeDefinition>
```

Add or extend a `AttributeFilterPolicy` in your `attribute-filter.xml` configuration file:

```
<afp:AttributeFilterPolicy id="fooFilter">
  <afp:PolicyRequirementRule xsi:type="basic:ANY"/>

  <afp:AttributeRule attributeID="foo">
    <afp:PermitValueRule xsi:type="basic:ANY"/>
  </afp:AttributeRule>
</afp:AttributeFilterPolicy>
```

Shibboleth SP

Add the new attribute to the `attribute-map.xml`

```
<Attribute name="foo" id="foo"/>
```

EOxServer

- Make a copy of the default attribute dictionary (`{%EOXSERVER_CODE_DIRECTORY}/conf/defaultAttributeDictionary`).
- Add the attribute:

```
foo=foo
```

- Register the new dictionary in the EOxServer configuration.

SOAP Components

Table of Contents

- *SOAP Components* (page 97)
 - *Security Token Service* (page 98)
 - *Policy Enforcement Point Service* (page 99)
 - *SOAP Security Proxy* (page 99)
 - * *Generating the Proxy* (page 99)
 - * *Installing the Proxy* (page 100)

The following services are needed for the SOAP security part: The following services are needed for the SOAP security part:

- Security Token Service
- Charon Authorisation Service
- Policy Enforcement Point Service
- SOAP Security Proxy

To install and configure the HTTP security components, you have to follow these steps:

1. Install the Charon *Authorisation Service* (page 77).
2. Install the *Security Token Service* (page 98).
3. Install the *Policy Enforcement Point Service* (page 99).
4. Install the *SOAP Security Proxy* (page 99).

Security Token Service

The Security Token Service (STS) is responsible for the authentication of users and is documented and specified in the OASIS *WS-Trust*¹³¹ specification. The authentication assertion produced by the STS is formulated in the *Security Assertion Markup Language*¹³². A client trying to access a service secured by the IDMS has to embed this assertion in every service request.

The STS implementation used by the IDMS is the *HMA Authentication Service*¹³³. Please refer to the documentation included in the `\docs` folder of the HMA Authentication Service package how to compile the service. This document will only deal on how to install the service. To deploy the service successfully, you first have to install and configure an LDAP service. Then proceed with the following steps:

- Put the `authentication_v2.1.aar` folder in the `${AXIS2_HOME}/WEB-INF/services/` folder. The `authentication_v2.1.aar` folder contains all configuration files for the STS.
- The main configuration of the service takes place in the `authentication-service.properties`.
- Using the `saml-ldap-attributes-mapping.properties`, you can map your LDAP attributes to SAML attributes if necessary.
- You may configure the logging behaviour in the Log4J configuration file in `authentication-service-log4j.properties`.

Following properties can be set in the `authentication-service.properties` configuration file:

LDAPURL URL to the LDAP service.

LDAPSearchContext Search context for users.

LDAPPrincipal The “*user name*” used by the STS to access the LDAP service.

¹³¹ <http://docs.oasis-open.org/ws-sx/ws-trust/200512/ws-trust-1.3-os.html>

¹³² <http://www.oasis-open.org/committees/download.php/3406/oasis-sstc-saml-core-1.1.pdf>

¹³³ <http://wiki.services.eoportal.org/tiki-index.php?page=HMA+Authentication+Service>

LDAPCredentials The password used in combination with `LDAPPrincipal`

KEYSTORE_LOCATION Path to the Keystore file containing the certificate used for signing the SAML tokens.

KEYSTORE_PASSWORD The keystore password.

AUTHENTICATION_CERTIFICATE_ALIAS Alias of the keystore entry which is used for signing the SAML tokens.

AUTHENTICATION_CERTIFICATE_PASSWORD Password corresponding to the `AUTHENTICATION_CERTIFICATE_ALIAS`

CLIENT_CERTIFICATE_ALIASES Comma separated list with keystore aliases of trusted clients.

SAML_TOKEN_EXPIRY_PERIOD Defines how long a SAML token is valid.

SAML_ASSERTION_ISSUER SAML Token issuer.

SAML_ASSERTION_ID_PREFIX SAML Token prefix.

SAML_ASSERTION_NODE_NAMESPACE Namespace for attribute assertions.

ENCRYPTION_ENABLE Enables or disables encryption of SAML tokens.

INCLUDE_CERTIFICATE Enables or disables inclusion of SAML tokens.

LOG4J_CONFIG_LOCATION Path to the Log4J configuration file.

Policy Enforcement Point Service

Before installing the Policy Enforcement Point Service, refer to the *General Configuration for CHARON services* (page 85).

The Policy Enforcement Point enforces the authorisation decisions made by the Authorisation Service.

The next step is deploying the PEP Service, therefore extract the ZIP archive into the directory of your `${AXIS2_HOME}`.

Now you have to configure the service. The configuration files are in the `${AXIS2_HOME}/WEB-INF/classes` folder. Open the `PEPConfiguration.xml` to configure the service. The configuration file already contains documentation of the single elements.

SOAP Security Proxy

Before installing the SOAP Security Proxy, refer to the *General Configuration for CHARON services* (page 85). If you want to secure a Web Coverage Service, you can use the provided WCS Security Proxy. In this case, jump to *Installing the Proxy* (page 100).

Generating the Proxy

The SOAP Proxy is used as a proxy for a secured service. This means a user client does not communicate directly with a secured service, instead it sends all requests to the proxy service.

First, you have to generate the proxy service. In order to do this, open a shell and navigate to the `${ProxyCodeGen_HOME}/bin` directory. Run the script to generate the proxy service:

- Linux, Unices:

```
./ProxyGen.sh -wsdl path/to/wsdl
```

- Windows:

```
.\ProxyGen.bat -wsdl path\to\wsdl
```

The parameter `-wsdl` points to a file with the WSDL of the secured service.

After a successful service generation, the folder `${ProxyCodeGen_HOME}/tmp/ dist` contains the new proxy service.

Installing the Proxy

Take the service zip and deploy it by unpacking its content to the `${AXIS2_HOME}` folder. For MTOM support, please make sure that the parameter `enableMTOM` in the file `${AXIS2_HOME}/axis2.xml` is enabled.

Edit the `ProxyConfiguration_${SERVICE_NAME}.xml` to configure the service. The configuration file already contains documentation of the single elements.

SOAP Proxy

Table of Contents

- *SOAP Proxy* (page 100)
 - *SOAP Access to WCS* (page 100)
 - *Installation* (page 100)
 - * *Quick installation guide for EOxServer on CentOS* (page 100)
 - * *Old installation guide without rpms* (page 102)

SOAP Access to WCS

SOAP access to services provided by EOxServer is possible if the functionality is installed by the service provider. The protocol is SOAP 1.2 over HTTP.

EOxServer responds to the following WCS-EO requests via its SOAP service interface:

- DescribeCoverage
- DescribeEOCoverageSet
- GetCapabilities
- GetCoverage

To access the EOxServer by means of SOAP requests, you need to obtain the access URL from the service provider. For machine readable configuration the SOAP service exposes the WSDL configuration file: given a service address of `'http://example.org/eo_wcs'` the corresponding WSDL file may be downloaded at the URL `'http://example.org/eo_wcs?wsdl'`.

Installation

A quick-intall guide is provided below. For a full installation guide see the `INSTALL` file in the source tree.

Quick installation guide for EOxServer on CentOS

0. Prerequisites:

- *EOxServer* (page 15) installed and configured, including MapServer and Apache HTTP Server

- Add the yum repository as described in the *Installation on CentOS* (page 18) available at <http://packages.eox.at> (recommended) or directly obtain the RPM packages from http://yum.packages.eox.at/el/6/testing/x86_64/.

1. Basic install:

The following standard installation sets up soap_proxy for an installed eoxserver service accessible at <http://127.0.0.1/eoxserver/ows>

Caution: if upgrading an existing installation of soap_proxy, please be sure to make a backup of the directory `/usr/share/axis2c_eo/services/soapProxy`. The `eo_soap_proxy-1.0.1-1` package does not correctly preserve this directory during upgrading.

Via the repository:

```
sudo yum install axis2c_eo eo_soap_proxy
sudo /etc/init.d/httpd restart
```

or the packages:

```
sudo rpm -i axis2c_eo-1.6.0-3.x86_64.rpm
sudo rpm -i eo_soap_proxy-1.0.1-1.x86_64.rpm
sudo /etc/init.d/httpd restart
```

2. Test:

To test open a web browser to the page:

http://<your_server>/sp_eowcs?wsdl

You should see the wsdl.

Further testing may be done via soapui. See the file `soap_proxy/test/README.txt` in the source tree.

3. Add another service:

To add another service to the basic installation, perform the following steps as root:

By way of example let us say our new soap_proxy service shall be available at http://example.org/sp_foo, and the corresponding backend eoxserver is accessible at http://127.0.0.1/eoxs_foo

First, in the directory `/usr/local/share/axis2c/services` recursively copy the subdirectory `soapProxy` to `soapFoo`:

```
cp -r soapProxy soapFoo
cd soapFoo
```

In `soapFoo` rename `libsoapProxy.so` and `soapProxy.wsdl`:

```
mv libsoapProxy.so libsoapFoo.so
mv soapProxy.wsdl soapFoo.wsdl
```

Note that if selinux is enabled you may need adjust the object type of `libsoapFoo.so`.

edit `soapFoo.wsdl` - at the bottom of the file change `soap:address location` to the new endpoint:

```
<soap:address location="http://example.org/sp_foo"/>
```

edit `services.xml` - change `ServiceClass`, `BackendURL`, and `SOAPOperationsURL`:

```
<parameter name="ServiceClass" locked="xsd:false">soapFoo</parameter>
<parameter name="BackendURL">http://127.0.0.1/eoxs_foo/ows</parameter>
<parameter name="SOAPOperationsURL">http://example.org/sp_foo</parameter>
```

Optionally, you may consider updating the `<description>`.

Edit the file `/etc/httpd/conf.d/030_axis2c.conf`: In the block `<IfModule mod_proxy.c>`, add `'ProxyPass'` and `'ProxyPassReverse'` lines corresponding to your new service:

```
ProxyPass      /sp_foo http://127.0.0.1/sp_axis/services/soapFoo
ProxyPassReverse /sp_foo http://127.0.0.1/sp_axis/services/soapFoo
```

Old installation guide without rpms

0. Prerequisites:

The following is required before you can proceed with installing `soap_proxy`:

- `mapserver` installed & configured.
- Apache `httpd` server(`httpd2` on some systems) installed and running
- `eoxserver` is optional

1. Old Non-rpm installation

This is suitable for general installation e.g. if you are not using `eoxserver` but wish to use `mapserver` directly.

Warning: some of the configuration details are out of date, but the changes are not structural.

Also see the `INSTALL` file in the source tree.

Download from <http://ws.apache.org/axis2/c/download.cgi>

Make a directory for the code:

```
cd someplace
mkdir axis2c
setenv AXIS2C_HOME /path/to/someplace/axis2c
```

Follow the instructions in `'doc'` to compile, and use something like the following configure line to get `mod_axis2` configured for compiling at the same time:

```
./configure --with-apache2="/usr/include/apache2" \
--with-apr="/usr/include/apr-1" --prefix=${AXIS2C_HOME}
```

Execute the standard sequence:

```
make
make install
```

Copy `lib/libmod_axis2.so.0.6.0` to `<apache2 modules directory>` as `mod_axis2.so`.

Edit the file `${AXIS2C_HOME}/axis2.xml` and ensure that the parameter `enableMTOM` has the value `true`.

Check that the following directory exists, if not create it: `${AXIS2C_HOME}/services`

2. Deploy axis2 via your webserver

Configure `mod_axis2` in the apache server config file. On Suse Linux one might edit the file `/etc/apache2/default-server.conf`.

Set up a proxy:

```
<IfModule mod_proxy.c>
  ProxyRequests Off
  ProxyPass /sp_wcs http://127.0.0.1/o3s_axis/services/soapProxy
  ProxyPassReverse /sp_wcs http://127.0.0.1/o3s_axis/services/soapProxy
  ...
  <Proxy *>
    Order deny,allow
    Deny from all
    ...
  </Proxy>
</IfModule>
```

and deploy axis2:

```
LoadModule axis2_module /usr/lib64/apache2/mod_axis2.so
Axis2RepoPath /path/to/AXIS2C_HOME
Axis2LogFile /tmp/ax2logs
Axis2MaxLogFileSize 204800
Axis2LogLevel info
<Location /o3s_axis>
  SetHandler axis2_module
</Location>
```

3. Verify the deployment of axis2

Restart the webserver (`httpd2`) and open the following page:

```
http://127.0.0.1/o3s_axis/services
```

You should get a page that displays the text “Deployed Services” and is otherwise blank.

4. Configure and Compile Soap Proxy.

Change your working directory to the service directory in the `soap_proxy` source code:

```
cd <...>/soap_proxy/service
```

In `soapProxy.wsdl` set `<soap:address location=.../>`. Copy `TEMLATE_services.xml` to `services.xml`. In `services.xml` set `BackendURL` to the address of `eoXserver`.

Now change to the `src` directory:

```
cd src
```

In your environment or in the `Makefile` set `AXIS2C_HOME` appropriately, and execute:

```
make inst
```

Restart you `httpd` server and check that `http://127.0.0.1/o3s_axis/services` shows the `soapProxy` service offering the four EO-WCS operations.

Further testing may be done via `soapui`. See the file `soap_proxy/test/README.txt` in the source tree.

EOxServer Presentations

Table of Contents

- *EOxServer Presentations* (page 104)
 - *FOSS4G 2011, Denver* (page 104)
 - *AGIT 2011, Salzburg* (page 104)
 - *HMA-AWG February 2012, ESA ESRIN* (page 104)
 - *FOSSGIS 2012, Dessau* (page 105)
 - *Linuxwochen Wien 2012* (page 105)
 - *FOSS4G-CEE 2012, Prague* (page 105)
 - *HMA-AWG June 2012, ESA ESRIN* (page 105)
 - *Sentinel-3 OLCI/SLSTR and MERIS/(A)ATSR workshop 2012, ESA ESRIN* (page 105)
 - *SOMAP 2012, Vienna* (page 106)

This sections holds some links to presentations related to EOxServer.

FOSS4G 2011, Denver

WCS in MapServer 6.0¹³⁴

Download the presentation

The **FOSS4G**¹³⁵ is a global conference focused on Free and Open Source Software for Geospatial, organized by OSGeo¹³⁶.

AGIT 2011, Salzburg

Introducing WCS 2.0, EO-WCS, and Open Source implementations (MapServer, rasdaman, and EOxServer) enabling the Online Data Access to Heterogeneous Multidimensional Satellite Data¹³⁷

Download the presentation

The **Angewandte Geoinformatik (AGIT)**¹³⁸ is a conference for applied geo-informatics held annually in Salzburg, Austria. Since 5 years it includes the OSGeo Day where the presentation was given.

HMA-AWG February 2012, ESA ESRIN

WCS Standardization & Reference Implementation¹³⁹

Download the presentation

The **Heterogeneous Missions Access Architecture Working Group**¹⁴⁰ has been defined by the European Space Agency together with other relevant EO data owners (national agencies, European institutions and industry) for the management of the evolution of the interoperability interface standards defined within the HMA project and in follow on activities.

¹³⁴ <http://2011.foss4g.org/sessions/enhanced-support-ogcs-web-coverage-service-wcs-mapserver-60>

¹³⁵ <http://2011.foss4g.org/>

¹³⁶ <http://osgeo.org>

¹³⁷ http://www.agit.at/index.php?option=com_content&task=view&id=132&Itemid=72

¹³⁸ <http://agit.at>

¹³⁹ <https://wiki.services.eoportal.org/tiki-index.php?page=HMA%20AWG%20Meeting%231%202012%2015%20February%202012>

¹⁴⁰ <https://wiki.services.eoportal.org/tiki-index.php?page=HMA+AWG>

FOSSGIS 2012, Dessau

EOxServer, GDAL, MapServer - Zugang zu großen Archiven von Erdbeobachtungsdaten¹⁴¹

Download the presentation

Freie und Open Source Software für Geoinformationssysteme (FOSSGIS)¹⁴² is the German speaking annual OS-Geo conference

Linuxwochen Wien 2012

EOxServer & Mapserver - Open Source Lösungen für Erdbeobachtungsdaten¹⁴³

Download the presentation

Linuxwochen¹⁴⁴ is Austria's biggest event series dedicated to Open Source and Free Software.

FOSS4G-CEE 2012, Prague

EOxServer: A Solution for Online Access to Large Collections of Earth Observation Data¹⁴⁵

Download the presentation

FOSS4G-CEE¹⁴⁶ & Geoinformatics 2012 is the first local conference focused on Free and Open Source Software for Geospatial in Central and Eastern Europe. This year, it is organized together with the traditional Geoinformatics FCE CTU conference in Prague.

HMA-AWG June 2012, ESA ESRIN

Web Coverage Service 2.0 MapServer Implementation¹⁴⁷

Download the presentation

Description: See *HMA-AWG February 2012, ESA ESRIN* (page 104) above

Sentinel-3 OLCI/SLSTR and MERIS/(A)ATSR workshop 2012, ESA ESRIN

EOxServer - An Open Source Solution for Standardized Online Access to Earth Observation Data¹⁴⁸

Download the poster

The *Sentinel-3 OLCI/SLSTR and MERIS/(A)ATSR workshop*¹⁴⁹ is organized by the European Space Agency, together with Eumetsat, and hosted in ESA-ESRIN, Frascati, Italy. The workshop is open to ESA Principle Investigators and co-investigators, scientists and students using MERIS/(A)ATSR data, future follow-on Sentinel-3 OLCI/SLSTR data users, representatives from GMES services, national, European and international space agencies and value adding industries.

¹⁴¹ <http://www.fossGIS.de/konferenz/2012/programm/events/379.de.html>

¹⁴² <http://www.fossGIS.de/konferenz.html>

¹⁴³ http://linuxwochen.at/index.php?option=com_content&view=article&id=331&Itemid=83

¹⁴⁴ <http://linuxwochen.at/>

¹⁴⁵ <http://foss4g-cee.org/program/presentations/eoxserver-a-solution-for-online-access-to-large-collections-of-earth-observation-data/>

¹⁴⁶ <http://foss4g-cee.org/>

¹⁴⁷ <https://wiki.services.esoportal.org/tiki-index.php?page=HMA%20AWG%20Meeting%20no2%202012%208%20June%202012>

¹⁴⁸ <http://congrexprojects.com/sen3symposium/poster-sessions>

¹⁴⁹ <http://www.sen3symposium.org/>

SOMAP 2012, Vienna

EOxServer - Accessing Large Archives of Earth Observation Data Online¹⁵⁰ (photo¹⁵¹)

Download the presentation

The [Symposium on Service-Oriented Mapping](#)¹⁵² aims to be a multidisciplinary event, spanning from computer science to geobusiness. The aim is to bring together various stakeholders in the area of Service-Oriented mapping (data producers, mapping agencies and companies, infrastructure providers, software developers, cartographers, artists, ...) in order to discuss the influence of this new production environment (the networked spatial infrastructure and its service-oriented distribution) on the map production and the perspectives of the new paradigm for research and development in cartography.

Configuration Options

In this section, all valid configuration options and their interpretations are listed.

[core.system]

```
instance_id
```

Mandatory. The ID (name) of your instance. This is used on several locations throughout EOxServer and is inserted into a number of service responses.

[processing.gdal.reftools]

```
VRT_tmp_dir
```

A path to a directory for temporary files created during the orthorectification of referential coverages. This configuration option defaults to the [systems standard](#)¹⁵³.

[backends.cache]

In future, options in this section will influence the behavior of caching of FTP and rasdaman data.

[resources.coverages.coverage_id]

```
reservation_time
```

Determines the time a coverage ID is reserved when inserting a coverage into the system. Needs to be in the following form: <days>:<hours>:<minutes>:<seconds> and defaults to 0:0:30:0.

[services.owscommon]

```
http_service_url
```

Mandatory. This parameter is the actual domain and path URL to the OWS services served with the EOxServer instance. This parameter is used in various contexts and is also included in several OWS service responses.

¹⁵⁰ http://somap.cartography.at/?SOMAP_2012:Program:November_23rd_2012

¹⁵¹ <http://somap.cartography.at/plugins/gallery/includes/image.php?pic=L2hvbWUvLnNpdGVzLzEyL3NpdGUyNDMvd2ViL3NvbWFWmJAxMi9nYWxsZXJ5LW1060&w=1600>

¹⁵² http://somap.cartography.at/?SOMAP_2012

¹⁵³ <http://docs.python.org/library/tempfile.html#tempfile.mkstemp>

[services.ows]

This section entails various service metadata settings which are embedded in W*S GetCapabilities documents.

```
update_sequence=20131219T132000Z
```

```
name=EOxServer EO-WCS
```

```
title=Test configuration of MapServer used to demonstrate EOxServer
```

```
abstract=Test configuration of MapServer used to demonstrate EOxServer
```

```
onlineresource=http://eoxserver.org
```

```
keywords=<KEYWORDLIST>
```

```
fees=None
```

```
access_constraints=None
```

```
provider_name=<CONTACTORGANIZATION>
```

```
provider_site=<URL>
```

```
individual_name=<CONTACTPERSON>
```

```
position_name=<CONTACTPOSITION>
```

```
phone_voice=<CONTACTVOICETELEPHONE>
```

```
phone_facsimile=<CONTACTFACSIMILETELEPHONE>
```

```
delivery_point=<ADDRESS>
```

```
city=<CITY>
```

```
administrative_area=<STATEORPROVINCE>
```

```
postal_code=<POSTCODE>
```

```
country=<COUNTRY>
```

```
electronic_mail_address=<CONTACTELECTRONICMAILADDRESS>
```

```
hours_of_service=<HOURSOFSERVICE>
```

```
contact_instructions=<CONTACTINSTRUCTIONS>
```

```
role=Service provider
```

[services.ows.wms]

```
supported_formats=<MIME type>[, <MIME type>[, <MIME type> ... ]]
```

A comma-separated list of MIME-types defining the raster file format supported by the WMS `getMap()` operation. The MIME-types used for this option must be defined in the *Format Registry* (see “*Supported Raster File Formats and Their Configuration* (page 111)”).

```
supported_crs= <EPSG-code>[, <EPSG-code>[, <EPSG-code> ... ]]
```

List of common CRSes supported by the WMS `getMap()` operation (see also “*Supported CRSs and Their Configuration* (page 110)”).

[services.ows.wcs]

```
supported_formats=<MIME type>[, <MIME type>[, <MIME type> ... ]]
```

A comma-separated list of MIME-types defining the raster file format supported by the WCS `getCoverage()` operation. The MIME-types used for this option must be defined in the *Format Registry* (see “*Supported Raster File Formats and Their Configuration* (page 111)”).

```
supported_crs= <EPSG-code>[, <EPSG-code>[, <EPSG-code> ... ]]
```

List of common CRSes supported by the WCS `getMap()` operation. (see also “*Supported CRSs and Their Configuration* (page 110)”).

[services.ows.wcs20]

```
paging_count_default
```

The maximum number of `wcs:coverageDescription` elements returned in a WCS 2.0 *EOCoverageSetDescription*. This also limits the *count parameter* (page 47). Defaults to 10.

```
default_native_format=<MIME-type>
```

The default *native format* cases when the source format cannot be used (read-only GDAL driver) and there is no explicit source-to-native format mapping. This option must be always set to a valid format (GeoTIFF by default). The MIME-type used for this option must be defined in the *Format Registry* (see “*Supported Raster File Formats and Their Configuration* (page 111)”).

```
source_to_native_format_map=[<src.MIME-type, native-MIME-type>[, <src.MIME-type,  
↪ native-MIME-type> ... ]]
```

The explicit source to native format mapping. As the name suggests, it defines mapping of the (zero, one, or more) source formats to a non-defaults native formats. The source formats are not restricted to the read-only ones. This option accepts comma-separated list of MIME-type pairs. The MIME-types used for this option must be defined in the *Format Registry* (see “*Supported Raster File Formats and Their Configuration* (page 111)”).

```
maxsize = 2048
```

The maximum size for each dimension in WCS `GetCoverage` responses. All sizes above will result in exception reports.

[services.ows.wcst11]

```
allow_multiple_actions
```

This flag enables/disables mutiple actions per WCSt request. Defaults to *False*.

NOTE: It is safer to keep this feature disabled. In case of a failure of one of the multiple actions, an OWS exception is returned without any notification which of the actions were actually performed, and which have not been performed at all. Therefore, we recomend to use only one action per request.

```
allowed_actions
```

Comma-separated list of allowed actions. Each item is one of *Add*, *Delete*, *UpdateAll*, *UpdateMetadata* and *UpdateDataPart*. By default no action is allowed and each needs to be explicitly activated. Currently, only the *Add* and *Delete* actions are implemented by the EOxServer.

```
path_wcst_temp
```

Mandatory. A path to an existing directory for temporary data storage during the WCS-T request processing. This should be a directory which is not used in any other context, since it might be cleared under certain circumstances.

```
path_wcst_perm
```

Mandatory. A path to a directory for permanent storage of transacted data. This is the final location where transacted datasets will be stored. It is also a place where the *Delete* action (when enabled) is allowed to remove the stored data.

[services.auth.base]

For detailed information about authorization refer to the documentation of the *Identity Management System* (page 74).

```
pdb_type
```

Determine the Policy Decision Point type; defaults to 'none' which deactivates authorization.

```
authz_service
```

URL of the Authorization Service.

```
attribute_mapping
```

Path to an attribute dictionary for user attributes.

```
serviceID
```

Sets a custom service identifier

```
allowLocal
```

Allows full local access to the EOxServer. Use with care!

[webclient]

The following configuration options affect the behavior of the *Webclient interface* (page 71).

```
preview_service
outline_service
```

The service type for the outline and the preview layer in the webclient map. One of *wms* (default) or *wmts*.

```
preview_url
outline_url
```

The URL of the preview and outline service. Defaults to the value of the *services.owscommon.http_service_url* configuration option.

[testing]

These configuration options are used within the context of the *Autotest instance* (page 129).

```
binary_raster_comparison_enabled
```

Enable/disable the binary comparison of rasters in test runs. If disabled these tests will be skipped. By default this feature is activated but might be turned off in order to prevent test failures originating on platform differences.

```
rasdaman_enabled
```

Enable/disable rasdaman test cases. If disabled these tests will be skipped. Defaults to *false*.

Supported CRSs and Their Configuration

Table of Contents

- *Supported CRSs and Their Configuration* (page 110)
 - *Coordinate Reference Systems* (page 110)
 - *Web Map Service* (page 110)
 - *Web Coverage Service* (page 111)

This section describes configuration of Coordinate Reference Systems for both WMS and WCS services.

Coordinate Reference Systems

The Coordinate Reference System (CRS) denotes the projection of coordinates to an actual position on Earth. EOxServer allows the configuration of supported CRSes for WMS and WCS services. The CRSes used by EOxServer are specified exclusively by means of *EPSG numerical codes*¹⁵⁴.

Web Map Service

EOxServer allows the specification of the overall list of CRSes supported by all published map layers (listed at the top layer of the WMS *Capabilities* XML document). In case of no common CRS the list can be empty. In addition to the list of common CRSes each individual layer has its *native* CRS which need not to be necessarily listed among the common CRSes. The meaning of the *native* CRS changes based on the EO dataset:

- Rectified Datasets - the actual CRS of the source geo-rectified raster data,
- Rectified Stitched Mosaic - the actual CRS of the source geo-rectified raster data,
- Referenceable Dataset - the CRS of the geo-location grid tie-points.
- Time Series - always set to WGS 84 (may be subject to change in future).

¹⁵⁴ <http://www.epsg-registry.org>

This *native* CRS is also used as the CRS in which the geographic extent (bounding-box) is published.

The list of WMS common CRSes is specified as a comma separated list of EPSG codes in the EOxServer's configuration (`<instance path>/conf/eoxserver.conf`) in section `services.ows.wms`:

```
[services.ows.wms]
supported_crs= <EPSG-code>[,<EPSG-code>[,<EPSG-code> ... ]]
```

Web Coverage Service

EOxServer allows the specification of a list of CRCSes to be used by the WCS. These CRSes can be used to select subsets of the desired coverage or, in case of rectified datasets (including rectified stitched mosaics) to specify the CRS of the output image data. The latter case is not applicable to referenceable datasets as these are always returned in the original image geometry.

The list of WCS supported CRSes is specified as a comma-separated list of EPSG codes in the EOxServer configuration (`<instance path>/conf/eoxserver.conf`) in section `services.ows.wcs`:

```
[services.ows.wcs]
supported_crs= <EPSG-code>[,<EPSG-code>[,<EPSG-code> ... ]]
```

Supported Raster File Formats and Their Configuration

Table of Contents

- *Supported Raster File Formats and Their Configuration* (page 111)
 - *Format Registry* (page 111)
 - *Format Configuration* (page 112)
 - *Web Coverage Service - Format Configuration* (page 112)
 - *Web Coverage Service - Native Format Configuration* (page 112)
 - *Web Map Service - Format Configuration* (page 113)
 - *References* (page 113)

In this section, the EOxServer's handling of raster file formats and OWS service specific format configuration is described.

Format Registry

The format registry is the list of raster file formats recognised by EOxServer. It holds definitions of both input and output formats. Each format record defines the MIME-type (unique, primary key), library, driver, and the default file extension.

Currently, EOxServer handles the raster data exclusively by means of the [GDAL](http://www.gdal.org)¹⁵⁵ library. Thus, in principle, any raster file format supported by the [GDAL](http://www.gdal.org/formats_list.html)¹⁵⁶ library is supported by EOxServer. In particular, any raster file format readable by the GDAL library (provided that the file structure can be decomposed to one single-type, single- or multi-band image) can be used as the input and, vice versa, any raster file format writeable by the GDAL library can be used as the output produced by WCS and WMS services.

Any raster file format intended to be used by EOxServer must be defined in the format registry. The format registry then provides unique mappings from MIME-type to the (GDAL) format driver.

¹⁵⁵ <http://www.gdal.org>

¹⁵⁶ http://www.gdal.org/formats_list.html

Format Configuration

The format registry configuration is split in two parts (files):

- per-installation (mandatory) format configuration (set up automatically during the EOxServer installation) defining the default baseline set of formats (`<instal.path>/eoxserver/conf/default_formats.conf`).
- per-instance (optional) format configuration allowing customization of the format registry (`<instance path>/conf/formats.conf`).

In case of conflicting format definitions, the per-instance configuration takes precedence. Both formats' configuration files share the same text file format.

The formats' configuration is a simple text file containing a simple list of format definitions. One format definition (record) per line. Each record is then a comma separated list of the following text fields:

```
<MIME-type>, <driver>, <file extension>
```

The mime type is used as the primary key and thus any repeated MIME-type will rewrite the previous format definition(s) using this MIME-type. The driver field should be in format `GDAL/<GDAL driver name>`. To list available drivers provided by your GDAL installation use the following command:

```
gdalinfo --formats
```

The GDAL prefix is used as place-holder to allow future use of additional library back-ends. The file extension shall be written including the separating dot `..`. Any leading or trailing white-characters as well as empty lines are ignored. The `#` character is used as line-comment and any content between this character and the end of the line is ignored.

An example format definition:

```
image/tiff,GDAL/GTiff,.tif # GeoTIFF raster file format
```

Since the list of supported drivers may vary for different installations of the back-end (GDAL) library, the library drivers are checked by EOxServer ignoring any format definitions requiring non-supported library drivers. Any invalid format record is reported to the EOxServer log. Further, EOxServer checks automatically which of the library drivers are 'read-only', i.e., which cannot be used to produce output images, and restricts these to be used for data input only.

Web Coverage Service - Format Configuration

The list of the file formats supported by the *Web Coverage Service* (WCS) is specified in the EOxServer configuration (`<instance path>/conf/eoxserver.conf`) in the section `services.ows.wcs`:

```
[services.ows.wcs]
supported_formats=<MIME type>[,<MIME type>[,<MIME type> ... ]]
```

The supported WCS formats are specified as a comma-separated list of MIME-types. The listed MIME-types must be defined in the format registry otherwise they will be ignored. Read-only file formats will also be ignored.

The supported formats are announced through the `WCS Capabilities` and `CoverageDescription` (the output may vary based on the WCS version used). The use of invalid MIME-types (not listed among the supported formats) in `getCoverage()` requests will lead to errors (OWS Exceptions).

Web Coverage Service - Native Format Configuration

The *native format* (as defined by [WCS 2.0.1 \[OGC 09-110r4\]¹⁵⁷](http://www.opengis.org/standards/wcs)) is the default raster file format returned by the `getCoverage()` operation in case of a missing explicit format specification. By default, EOxServer sets

¹⁵⁷ <http://www.opengis.org/standards/wcs>

the *native format* to the format of the stored source data (source format), however, in cases when the source format cannot be used ('read-only' source format) and/or another default format is desired, EOxServer allows the configuration of WCS *native formats* (<instance path>/conf/eoxserver.conf, section services.ows.wcs20):

```
[services.ows.wcs20]
default_native_format=<MIME-type>
source_to_native_format_map=[<src.MIME-type, native-MIME-type>[, <src.MIME-type,
↳native-MIME-type> ... ]]
```

The default *native format* option is used in cases when the source format cannot be used (read-only) and no source to native format mapping is present. This option must always be set to a valid format (GeoTIFF by default). The source to native format mapping, as the name suggests, maps the (zero, one, or more) source format(s) to non-default native formats. The source formats are not restricted to the read-only ones. This option accepts a comma-separated list of MIME-type pairs.

Web Map Service - Format Configuration

The list of the file formats supported by the *Web Map Service's* (WMS) `getMap()` operation is specified in the EOxServer configuration (<instance path>/conf/eoxserver.conf) in section `services.ows.wms`:

```
[services.ows.wms]
supported_formats=<MIME type>[, <MIME type>[, <MIME type>... ]]
```

The supported WMS formats are specified as a comma-separated list of MIME-types. The listed MIME-types must be defined in the format registry otherwise they will be ignored. The read-only file formats will be ignored.

The supported formats are announced through the WMS *Capabilities* (the output may vary based on the WMS version used).

References

[OGC 09-110r4] <http://www.opengeospatial.org/standards/wcs>

Asynchronous Task Processing

Table of Contents

- *Asynchronous Task Processing* (page 113)
 - *Introduction* (page 114)
 - *Tasks* (page 114)
 - * *Introduction* (page 114)
 - * *Life-cycle* (page 114)
 - *ATP Installation and Configuration* (page 115)
 - *ATP Operation* (page 115)
 - *ATP Demo Application* (page 116)
 - *Performance considerations* (page 117)
 - *Further reading* (page 117)

Introduction

The *Asynchronous Task Processing* (ATP) subsystem, as the name suggests, extends the *EOxServer* functionality by the ability to process tasks asynchronously, i.e., in background independently of the default *EOxServer*'s synchronous client request processing.

Although the ATP subsystem is primarily designed to support asynchronous request processing of OGC Web Services such as the Web Coverage Service transaction extension (WCS-T) and/or the Web Processing Service (WPS), it is not limited to these and other parts of *EOxServer* may use it as well.

The ATP subsystem employs the model of a single shared task queue and one or more *Asynchronous Task Processing Daemons* (ATPD) executing the pending tasks pulled from the task queue. A single ATPD is not restricted to a single processed task at time and it can internally process multiple tasks concurrently, e.g., by employing a pool of parallel worker threads assigned to multiple CPU cores.

The ATP subsystem is implemented as Django application using a DB model as the task queue. Although the underlying DB storage may be seen as suboptimal in terms of performance and latency it assures tolerance of the subsystem to possible failures or maintenance shut-downs of both *EOxServer* and/or ATPDs.

Tasks

Introduction

For the correct operation of the ATP subsystem it is essential to understand the concept of a *task* and its life-cycle.

A *task* is an atomic and isolated action (amount of work) to be performed by *EOxServer*. When created, each task has a handler subroutine (python code to be executed) and a set of task specific input parameters to be processed by the handler subroutine. When finished, the tasks produce outputs.

The tasks may be created by different applications (*EOxServer*'s apps and services). The tasks sharing the same handler subroutine and generic parameters belong to the same *task type*.

The ATP is expected to be shared by multiple applications. ATPDs pull the tasks from the shared queue in First-In-First-Out fashion (regardless of the task type) and execute the given handler subroutines. Significant benefit of this shared nature of the ATP subsystem is the control over the processing resources (pool of workers) and isolation of the execution details from the application (isolated from details such as the number of ATPD and working threads).

Life-cycle

The life-cycle of an asynchronous task, i.e., its possible states and state transitions are displayed in Fig.3.

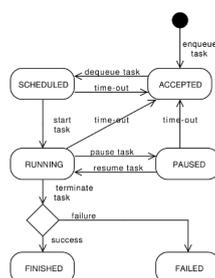


Fig. 1.15: Fig.1: ATP Task State Diagram

Any existing task can be in one of the following states:

- ACCEPTED - a new enqueued task waiting to be pulled by an ATPD (initial state)
- SCHEDULED - a task pulled (dequeued) by an ATPD but not yet started
- RUNNING - a task being processed by an ATPD

- PAUSED - a task which has been put on hold and which is waiting to be resumed
- FINISHED - a task which has been finished successfully (terminal state)
- FAILED - a task which has been finished by a failure (terminal state)

When a task is created and enqueued for processing (ACCEPTED) it is stored in the DB task queue waiting for an ATPD to pull the task out. In this state, it is safely stored and protected against failures and shut-downs of both of the producer (ATPD can access the DB) and of the ATPD (producer can access the DB).

When a task is in one of the intermediate states (SCHEDULED, RUNNING, or PAUSED) it is being processed by an ATPD and it is vulnerable to possible failures. In these states, any unexpected crash of the ATPD could leave a task in an intermediate state forever. Therefore each task type has assigned a security time-out after which the task is considered to be abandoned and shall be re-enqueued for new processing (ACCEPTED). A task, however, can be re-enqueued for limited times (3 times by default). After the number of restarts has been exceeded the task will be rejected (FAILED). This mechanism ensures that no task would be abandoned unfinished after an occasional ATPD crash but also that a defective task would get stacked in the time-out loop.

When a task is in one of the terminal states (FINISHED or FAILED) it is safely stored in the DB. By default a terminated task will be stored forever, however, it is possible to specify an task type specific time-out after which the terminated tasks will be removed automatically.

ATP Installation and Configuration

There are no specific steps to install and configure the ATP subsystem except the basic *EOxServer* installation and configuration. The ATP is tightly coupled with *EOxServer* and works right out of box.

To track the status of the executed tasks and view the stored outputs auxiliary ATP HTML views can be enabled by adding following lines to the URL patterns ('url.py' configuration file) of the actual *EOxServer* instance:

```
urlpatterns = patterns('',
    ...
    (r'^process/status$', procViews.status ),
    (r'^process/status/(?P<requestType>[/]{,64})/(?P<requestID>[/]{,64})$',
    ↪procViews.status ),
    (r'^process/task$', procViews.task ),
    (r'^process/response/(?P<requestType>[/]{,64})/(?P<requestID>[/]{,64})$',
    ↪procViews.response ),
    ...
)
```

ATP Operation

The ATP operation requires at least one ATPD to be running. Currently, there is only one ATPD implemented in *EOxServer*. This ATPD uses multiple sub-processes to process the tasks concurrently. By default, the numbers of sub-processes equals the number of available CPU cores. This ATPD can be executed as follows:

```
$ export PYTHONPATH=<EOxServer install.path>:<EOxServer instance path>
$ export DJANGO_SETTINGS_MODULE=autotest.settings
$ <EOxServer install.path>/tools/asyncProcServer.py

[0x504DD5AE614D562C] INFO: Default number of working threads: 4
[0x504DD5AE614D562C] INFO: 'autotest.settings' ... is set as the Django settings_
↪module
Spatialite version ...: 2.4.0    Supported Extensions:
  - 'VirtualShape'    [direct Shapefile access]
  - 'VirtualDbf'     [direct Dbf access]
  - 'VirtualText'    [direct CSV/TXT access]
```

```

- 'VirtualNetwork' [Dijkstra shortest path]
- 'RTree'          [Spatial Index - R*Tree]
- 'MbrCache'      [Spatial Index - MBR cache]
- 'VirtualFDO'    [FDO-OGR interoperability]
- 'SpatiaLite'    [Spatial SQL - OGC]
PROJ.4 Rel. 4.7.1, 23 September 2009
GEOS version 3.2.2-CAPI-1.6.2
[0x504DD5AE614D562C] INFO: ATPD Asynchronous Task Processing Daemon has just been
↳started!
[0x504DD5AE614D562C] INFO: ATPD: id=0x504DD5AE614D562C (5786516041174439468)
[0x504DD5AE614D562C] INFO: ATPD: hostname=localhost
[0x504DD5AE614D562C] INFO: ATPD: pid=3295

```

The PYTHONPATH and DJANGO_SETTINGS_MODULE values can be passed as command line arguments by the '-p' and '-s' options, respectively. The default number of worker sub-processes can be overridden by the '-n' option:

```

$ <EOxServer install.path>/tools/asyncProcServer.py -n 6 -s "autotest.settings" -p
↳"<EOxServer install.path>" -p "<EOxServer instance path>"

[0xADDB15DB482ED425] INFO: Default number of working threads: 4
[0xADDB15DB482ED425] INFO: Setting number of working threads to: 6
[0xADDB15DB482ED425] INFO: 'autotest.settings' ... is set as the Django settings
↳module
SpatiaLite version ..: 2.4.0    Supported Extensions:
- 'VirtualShape' [direct Shapefile access]
- 'VirtualDbf'   [direct Dbf access]
- 'VirtualText' [direct CSV/TXT access]
- 'VirtualNetwork' [Dijkstra shortest path]
- 'RTree'        [Spatial Index - R*Tree]
- 'MbrCache'     [Spatial Index - MBR cache]
- 'VirtualFDO'   [FDO-OGR interoperability]
- 'SpatiaLite'   [Spatial SQL - OGC]
PROJ.4 Rel. 4.7.1, 23 September 2009
GEOS version 3.2.2-CAPI-1.6.2
[0xADDB15DB482ED425] INFO: ATPD Asynchronous Task Processing Daemon has just been
↳started!
[0xADDB15DB482ED425] INFO: ATPD: id=0xADDB15DB482ED425 (-5919113253695335387)
[0xADDB15DB482ED425] INFO: ATPD: hostname=holly3
[0xADDB15DB482ED425] INFO: ATPD: pid=3345

```

The server can be gracefully terminated by using 'Ctrl-C' or the TERM signal.

ATP Demo Application

There is a demo application showing the running of the ATPD and the ATP as such available in the default EOxServer installation. This demo application can be executed as follows:

```

$ export PYTHONPATH=<EOxServer install.path>:<EOxServer instance path>
$ export DJANGO_SETTINGS_MODULE=autotest.settings
$ <EOxServer install.path>/atp_test.py
SpatiaLite version ..: 2.4.0    Supported Extensions:
- 'VirtualShape' [direct Shapefile access]
- 'VirtualDbf'   [direct Dbf access]
- 'VirtualText' [direct CSV/TXT access]
- 'VirtualNetwork' [Dijkstra shortest path]
- 'RTree'        [Spatial Index - R*Tree]
- 'MbrCache'     [Spatial Index - MBR cache]
- 'VirtualFDO'   [FDO-OGR interoperability]
- 'SpatiaLite'   [Spatial SQL - OGC]
PROJ.4 Rel. 4.7.1, 23 September 2009

```

```

GEOS version 3.2.2-CAPI-1.6.2
ENQUEUE: test_5710ffb4189c4345aebde828d2bbc640 000000
ENQUEUE: test_47e161ec633b4105a1d174759f4a933d 000001
ENQUEUE: test_e53cf3ae654a447191e1308d805d8777 000002
ENQUEUE: test_fb71659cb9274383a8820e0110c86e15 000003
ENQUEUE: test_0e6e5edcdf8244d9b25a932cbd8c6112 000004
ENQUEUE: test_be5fa7af84444c47aba731c8e816f99b 000005
ENQUEUE: test_aae3faa14b5e4f48b8cabae7a0b01a3b 000006
ENQUEUE: test_6be7ea23f0984efbb09181503aa1a974 000007

```

Performance considerations

The ATP is designed for resource demanding longer running tasks (10 seconds and more) which in case of a synchronous operation could clog the system or lead to connection time-outs. On contrary, *light* tasks (less than 1 sec.) should preferably be executed synchronously.

Further reading

The database model used in the ATP subsystem is described in the *Task Tracker Data Model* (page 125) section. The developers' guide, helping with the creation of ATP based applications, can be found in the *Asynchronous Task Processing - Developers Guide* (page 138) section. The complete API reference can be found in `eoxserver.resources.processes.tracker`.

Web Coverage Service - Transaction Extension

Table of Contents

- *Web Coverage Service - Transaction Extension* (page 117)
 - *Introduction* (page 117)
 - *Implementation Details* (page 118)
 - * *Configuration* (page 118)
 - * *Adding New Coverages* (page 118)
 - * *Deleting Existing Coverages* (page 119)
 - * *Asynchronous Operation* (page 119)
 - *References* (page 120)

Introduction

This section describes the *Web Coverage Service - Transaction* (WCS-T) extension as implemented in *EOxServer*. The WCS-T interface is specified by the *Open Geospatial Consortium* (OGC) *Web Coverage Service - Transaction operation extension* (WCS-T) [OGC 07-068r4]¹⁵⁸ standard which describes the invocation of the service in detail. The WCS-T functionality is closely related to the data model of the WCS 2.0 *Earth Observation Application Profile* (EO-WCS) employed by *EOxServer* and allows the specification of EO-WCS metadata for newly inserted EO datasets.

¹⁵⁸ http://portal.opengeospatial.org/files/?artifact_id=28506

Implementation Details

EOxServer provides to option to insert (*Add* action) and delete (*Delete*) coverages (datasets in EO-WCS jargon) via the WCS-T service.

Configuration

For details on the WCS-T configuration see [*services.ows.wcst11*] (page 109).

Adding New Coverages

Currently, it is possible to insert only *Rectified* and *Referenceable* datasets. It is beyond the capabilities of the WCS-T service to assign datasets to container coverage types such as the *Rectified Stitched Mosaic* or *Dataset Series*. Neither is it possible to create plain (non-EO-WCS) coverages.

The input image data must be in valid GeoTIFF file format. No other file format is currently supported. The input is passed to the WCS-T service as a reference (URL, e.g., a *GetCoverage* KVP encoded request). It is not possible to embed the input image data in the WCS-T request.

The creation of a new EO-WCS dataset requires the specification of EO metadata. These metadata can be either passed by the user (recommended way) as a reference using the `ows:metadatat` XML element, or generated automatically by the WCS-T service guessing some of the parameters from the GeoTIFF annotation.

The user provided EO-WCS metadata can be either in form of an EO-O&M XML document or arbitrary XML document with embedded EO-O&M XML fragment (such as the *DescribeCoverage* response of a WCS service).

The following is an example of a valid request to add a coverage:

```
<?xml version="1.0" encoding="UTF-8"?>
<wcst:Transaction service="WCS" version="1.1"
  xmlns:wcst="http://www.opengis.net/wcs/1.1/wcst"
  xmlns:ows="http://www.opengis.net/ows/1.1"
  xmlns:xlink="http://www.w3.org/1999/xlink"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.opengis.net/wcs/1.1/wcst http://schemas.opengis.
  net/wcst/1.1/wcstTransaction.xsd">
  <wcst:InputCoverages>
    <wcst:Coverage>
      <!-- optional coverage identifier -->
      <ows:Identifier>CoverageId</ows:Identifier>
      <!-- reference to image data -->
      <ows:Reference
        xlink:href="http://foo.eox.at/ows?service=WCS&version=2.0.0&
        request=getCoverage&format=image/tiff&coverageid=CoverageId"
        xlink:role="urn:ogc:def:role:WCS:1.1:Pixels"/>
      <!-- optional reference to EO metadata -->
      <ows:Metadata
        xlink:href="http://foo.eox.at/ows?service=WCS&version=2.0.0&
        request=describeCoverage&coverageid=CoverageId"
        xlink:role="http://www.opengis.net/eop/2.0/EarthObservation"/>
      <wcst:Action codeSpace="http://schemas.opengis.net/wcs/1.1.0/actions.
    </wcst:Action>
    </wcst:Coverage>
  </wcst:InputCoverages>
</wcst:Transaction>
```

The coverage identifier specified by the `ows:Identifier` element is optional. When not specified or not usable (most likely because it is already in use by another coverage) a new, unique identifier is generated automatically. Thus the WCS-T service is not bound to the user provided identifier and the actual identifier shall always be read from the transaction response:

```
<?xml version="1.0" encoding="utf-8"?>
<TransactionResponse
  xmlns="http://www.opengis.net/wcs/1.1/wcst"
  xmlns:ows="http://www.opengis.net/ows/1.1"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.opengis.net/wcs/1.1/wcst http://schemas.opengis.
  ↪net/wcst/1.1/wcstTransaction.xsd">
  <RequestId>wcstReq_btjiFfo4a0vT1BQL-ki5</RequestId>
  <ows:Identifier>wcstCov_LoEYNGm3d10ZhUUGdlmm</ows:Identifier>
</TransactionResponse>
```

Unless there is a need for a specific coverage identifier we recommend to leave the identifier selection to be performed by the WCS-T service and omit the `ows:Identifier` element in case of WCS-T coverage inserts.

Deleting Existing Coverages

The coverages inserted via the WCS-T *Add* action can be removed by means of the WCS-T *Delete* action. For security reasons, only the coverages inserted via WCS-T can be actually removed via WCS-T. The only parameter required in the removal request is the coverage (dataset) identifier (`wcst:InputCoverages` XML element):

```
<?xml version="1.0" encoding="UTF-8"?>
<wcst:Transaction service="WCS" version="1.1"
  xmlns:wcst="http://www.opengis.net/wcs/1.1/wcst"
  xmlns:ows="http://www.opengis.net/ows/1.1"
  xmlns:xlink="http://www.w3.org/1999/xlink"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.opengis.net/wcs/1.1/wcst http://schemas.opengis.
  ↪net/wcst/1.1/wcstTransaction.xsd">
  <wcst:InputCoverages>
    <wcst:Coverage>
      <!-- required coverage identifier -->
      <ows:Identifier>wcstCov_LoEYNGm3d10ZhUUGdlmm</ows:Identifier>
      <wcst:Action codeSpace="http://schemas.opengis.net/wcs/1.1.0/actions.
  ↪xml">Delete</wcst:Action>
    </wcst:Coverage>
  </wcst:InputCoverages>
</wcst:Transaction>
```

Asynchronous Operation

EOxServer supports asynchronous WCS-T requests as specified by the [OGC 07-068r4]¹⁵⁹ standard. Asynchronous request processing can be invoked by any WCS-T request including the `wcst:ResponseHandler` element. This element shall contain an URL of the remote response handler where the response shall be sent once the asynchronous processing is finished:

```
<?xml version="1.0" encoding="UTF-8"?>
<wcst:Transaction service="WCS" version="1.1"
  xmlns:wcst="http://www.opengis.net/wcs/1.1/wcst"
  xmlns:ows="http://www.opengis.net/ows/1.1"
  xmlns:xlink="http://www.w3.org/1999/xlink"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.opengis.net/wcs/1.1/wcst http://schemas.opengis.
  ↪net/wcst/1.1/wcstTransaction.xsd">
  <wcst:InputCoverages>
    ...
  </wcst:InputCoverages>
  <wcst:RequestId>RequestId</wcst:RequestId>
  <!-- XML element enabling the asynchronous WCS-T processing -->
```

¹⁵⁹ http://portal.opengeospatial.org/files/?artifact_id=28506

```
<wcst:ResponseHandler>http://foo.eox.at/WCSTResponseHandler</  
↔wcst:ResponseHandler>  
</wcst:Transaction>
```

Currently, the WCS-T implementation supports HTTP and FTP URL schemas for the response handler. In the first case the response is delivered using HTTP/POST. In the latter case, the response is uploaded to a remote FTP server. In case of FTP, the user may specify a full file-name of the delivered file or target directory. If the FTP target is a directory the file-name of the stored response is generated from the request ID returned by the acknowledgement response:

```
<?xml version="1.0" encoding="utf-8"?>  
<Acknowledgement  
  xmlns="http://www.opengis.net/wcs/1.1/wcst"  
  xmlns:ows="http://www.opengis.net/ows/1.1"  
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
  xsi:schemaLocation="http://www.opengis.net/wcs/1.1/wcst http://schemas.opengis.  
↔net/wcst/1.1/wcstTransaction.xsd">  
  <TimeStamp>2012-04-13T16:00:07Z</TimeStamp>  
  <RequestId>wcstReq_6syhsJb02TtYwVxFHOur</RequestId>  
</Acknowledgement>
```

It is worth to mention that request identifiers can be specified in WCS-T requests, however this identifier provides only a hint to the WCS-T server and the server may change it to another value. Thus it is recommended to rely on the request identifier written in the WCS-T response and better omit the optional `wcst:RequestId` XML element in the WCS-T request.

It is possible to specify user/password for the response handler for both HTTP and FTP using the typical URL structure:

```
<schema>://[<username>@<password>]<host>/<path>
```

No other authentication is currently supported.

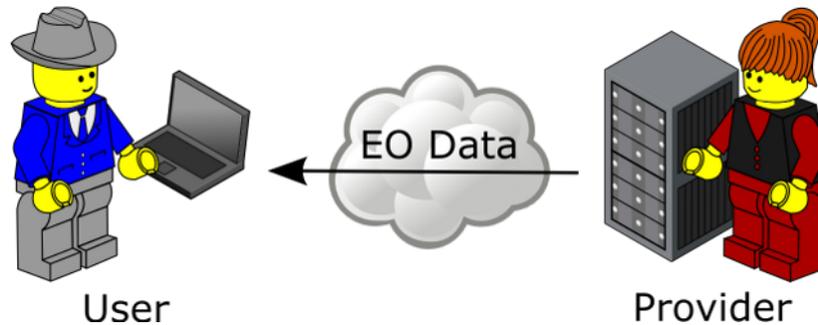
The asynchronous WCS-T operation requires the ATP (Asynchronous Task Processing) subsystem and, in particular, an ATPD (ATP Daemon) running. For more info on the ATP subsystem see the *Asynchronous Task Processing* (page 113) section.

References

[OGC 07-068r4] http://portal.opengeospatial.org/files/?artifact_id=28506

The Developers' Guide is intended for people who want to use EOxServer as a development framework for geospatial services, or do have to extend EOxServer's functionality to implement specific data and metadata formats for instance.

Users of the EOxServer software stack please refer to the *Users' Guide* (page 1). Users range from administrators installing and configuring the software stack and operators registering the available *EO Data* on the *Provider* side to end users consuming the registered *EO Data* on the *User* side.



Basics

Table of Contents

- *Basics* (page 121)
 - *Architectural Layout* (page 122)
 - * *Django* (page 122)
 - * *Database* (page 122)
 - * *MapServer* (page 122)
 - * *GDAL/OGR* (page 122)

This is a short description of the basic elements of the EOxServer software architecture.

Architectural Layout

EOxServer is Python software that builds on a handful of external packages. Most of the description in the following sections is related to the structure of the Python code, but in this section we present the building blocks used for EOxServer.

For further information on the dependencies please refer to the *Installation* (page 15) document in the *Users' Guide* (page 1).

Django

EOxServer is designed as a series of Django apps. It reuses the object-relational mapping Django provides as an abstraction layer for database access. Therefore, it is not bound to a specific database application, but can be run with different backends.

Database

Metadata and part of the EOxServer configuration is stored in a database. A handful of geospatially enabled database systems is supported, though we recommend either PostGIS or SpatiaLite.

MapServer

Many built-in functionalities rely on [MapServer](http://www.mapserver.org)¹⁶⁰ which EOxServer uses through its Python bindings to handle certain OGC Web Service requests.

GDAL/OGR

In some cases EOxServer uses the [GDAL/OGR](http://www.gdal.org)¹⁶¹ library for access to geospatial data directly (rather than through MapServer).

Core

Data Model

The core resources in EOxServer are coverages, more precisely GridCoverages. The EOxServer data model adopts and strongly relates to the data model from EO-WCS (OGC 10-140) as shown below in Figure: “*EO-WCS Data Model from OGC 10-140* (page 123)”.

Data Integration Layer

Figure: “*EOxServer Data Model for Coverage Resources* (page 123)” below shows the data model of the coverage resources. Note the correlation with the EO-WCS data model as shown above.

Data Access Layer

Figure: “*EOxServer Data Model for Back-ends* (page 124)” below shows the data model of the back-ends layer.

¹⁶⁰ <http://www.mapserver.org>

¹⁶¹ <http://www.gdal.org>

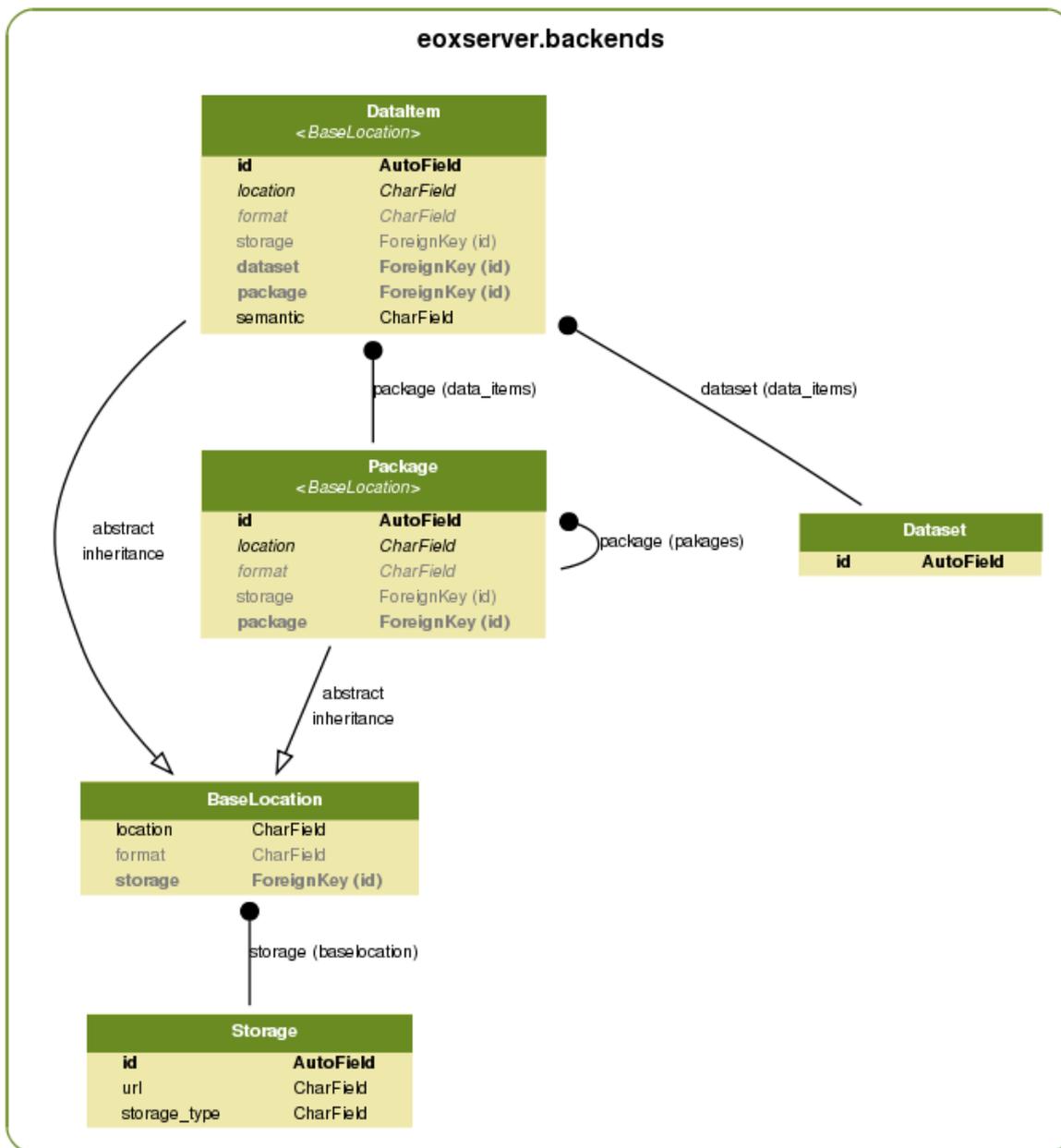


Fig. 2.3: EOxServer Data Model for Back-ends

Task Tracker Data Model

Asynchronous Task Processing (ATP) uses its own DB model displayed in Figure: “*EOxServer Data Model of ATP Task Tracker* (page 125)” to implement the task queue, store the task inputs and outputs and track the tasks’ status. (For more detail on ATP subsystem see “*Asynchronous Task Processing* (page 113)”).

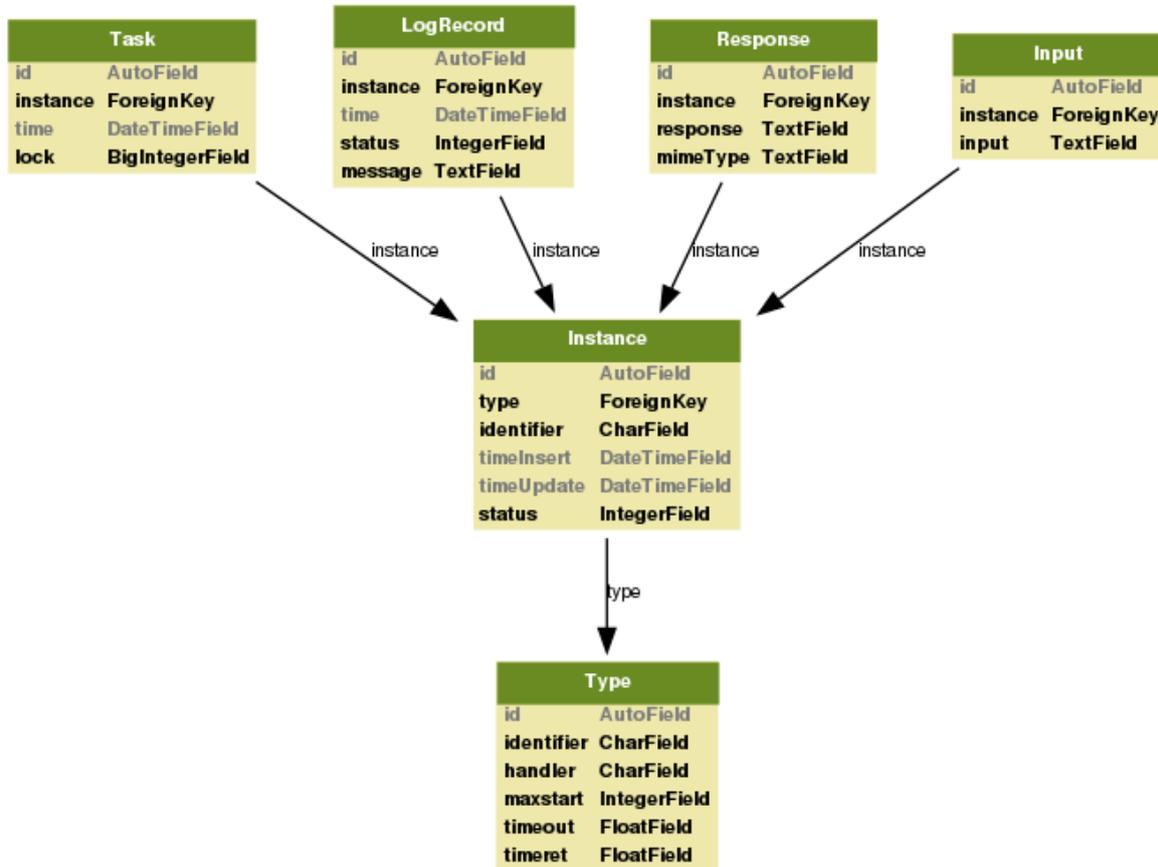


Fig. 2.4: EOxServer Data Model of ATP Task Tracker

Data Migrations

Over the time, the data models and thus the underlying database schema is changing to adapt new features or resolve bugs. Unfortunately Django cannot automatically detect and resolve those changes and upgrade existing instances for us.

To solve this problem, EOxServer uses [South](http://south.aeracode.org/)¹⁶² for schema and data migration management.

What are migrations?

For the uninitiated, migrations (also known as ‘schema evolution’ or ‘mutations’) are a way of changing your database schema from one version into another. Django by itself can only do this by adding new models, but nearly all projects will find themselves changing other aspects of models - be it adding a new field to a model, or changing a database column to have `null=True`.

—from the [South](http://south.aeracode.org/) documentation¹⁶³

¹⁶² <http://south.aeracode.org/>

¹⁶³ <http://south.readthedocs.org/en/latest/whataremigrations.html>

Setup

South needs to be initialized in every instance that wants to make use of the migration features.

Setting up *South* is quite easy, as all you need to do is install *South* (most easily via `pip` or `easy_install`), add it to the `INSTALLED_APPS` setting in `settings.py` and run `python manage.py syncdb`:

```
INSTALLED_APPS = (
    ...
    'eoxserver.testing',
    'eoxserver.webclient',
    'south'
)
```

A complete guide on all installation and configuration options can be found [here](#)¹⁶⁴.

Creating Migrations

To benefit from *South* it is important that *every* change in the data models concerning the actual database structure is tracked by a migration definition. Fortunately, for most of the small changes these can be created automatically by using *South*'s command `python manage.py schemamigration` and passing the app names which have changes in their models.

A very good tutorial for *South* can be found [here](#)¹⁶⁵.

Performing a Migration

To use *South* for data migrations only one command needs to be executed: `python manage.py migrate`. This applies all necessary database schema changes to your database and converts all included data from the original schema to the new one. This command effectively replaces `syncdb` (apart from the initial call to setup *South*).

Plugins

EOxServer uses a plugin framework to extend or alter the built-in functionality. The plugin system is based on *trac*'s [Component Architecture](#)¹⁶⁶. We copied the relevant file as `eoxserver.core.component` to not add the full *trac* framework as a dependency.

EOxServer plugins are classes that inherit from `eoxserver.core.component.Component`. Each component can implement any number of interfaces, which are usually skeleton classes to provide documentation of what methods and fields each implementation shall provide. In this architecture, interfaces are just informative and allow the runtime binding via `eoxserver.core.component.ExtensionPoint`.

All plugins are self-registering, which means the module containing the component just needs to be imported through any kind of import mechanism and, voilà, the component is registered and ready for use.

Important

Components should not be created manually, but only be retrieved via an `eoxserver.core.component.ExtensionPoint`. This further implies that the `__init__()` method shall not take any arguments, as instance creation is out of the reach.

Additionally, `Component` instances are never destroyed and shared among different threads, so it is highly advised to not store any data in the `Component` itself.

¹⁶⁴ <http://south.readthedocs.org/en/latest/installation.html>

¹⁶⁵ <http://south.readthedocs.org/en/latest/tutorial/part1.html>

¹⁶⁶ <http://trac.edgewall.org/wiki/TracDev/ComponentArchitecture>

Loading modules

EOxServer provides mechanisms to conveniently load modules and thus registering all entailed plugins. This is done via the `COMPONENTS` setting in your instances `settings.py`.

This setting must be an iterable of strings which follow the dotted python module path notation, with two exceptions:

- Module paths ending with `.*` will import all modules of a package.
- Paths ending with `.*.*` will do the same, with the exception of doing so recursively.

E.g: `"eoxserver.services.ows.*"` will load all subpackages and modules of the `eoxserver.services.ows` (page 256) package. (This is an easy way to enable all OWS services, by the way).

To only enable WMS in version 1.3 you could use the following import line: `"eoxserver.services.ows.wms.v13.*"`. If you only want to only enable specific requests (for whatever reason) you'd have to list their modules seperately.

The EOxServer instance `settings.py` template is already preconfigured with the most common components modules.

Example

The following demonstrates the use of the component architecture in a simplified manner:

In `myapp/interfaces.py`:

```
class DataReaderInterface(object):
    "Interface for reading data from a file."
    def read_data(self, filename, n):
        "Read 'n' bytes from the file 'filename'."
```

In `myapp/components.py`:

```
from eoxserver.core.component import Component, implements
from myapp.interfaces import DataReaderInterface

class BasicDataReader(Component):
    "Reads data from the file with the built-in Python functionality."

    implements(DataReaderInterface)

    def read_data(self, filename, n):
        with open(filename) as f:
            return f.read(n)
```

We can now use this component the following way in `myapp/main.py`:

```
from myapp.interfaces import DataReaderInterface

class App(object):
    data_readers = ExtensionPoint(DataReaderInterface)

    def run(self, filename):
        if not self.data_readers:
            raise Exception("No data reader implementation found.")

        print(data_readers[0].read_data(filename))
```

In the `"myapp/interfaces.py"` we declare an interface for `"data readers"`. The only method implementations of this interface shall provide is the `read_data()` method. In the `"myapp/components.py"` we provide a simple

implementation of this interface that uses built-in functionality to open a file and read a data. Please not the *implements(DataReaderInterface)* which declares that this component implements a specific interface.

In the “myapp/main.py” we declare a class that actually tries to find an implementation of the *DataReaderInterface* and invoke its `read_data()` method. In this case we only use the first available implementation of the interface, in other cases it might make sense to loop over all, or search for a specific one that satisfies a condition.

Services

This section deals with the creation of new Services handlers that allow to process OGC web service requests and are easily exposed via the `ows` view.

Service Handlers are Components that at least implement the *ServiceHandlerInterface* (page 255). For a Service Handler to be fully accessible it is also necessary to implement either or both of *GetServiceHandlerInterface* (page 255) and *PostServiceHandlerInterface* (page 255). For general information about Plugins/Components please refer to the *Plugins* (page 126) documentation.

Initial Setup

Each service handler must provide the following:

- The `service` the handler will contribute to
- The `versions` of the `service` the handler is capable of responding to
- The `request` of the `service` the handler is able to respond
- a `handle` method that takes a `django.http.HttpRequest`¹⁶⁷ as parameter

A service handler *can* provide an `index`, which allows the sorting of the handlers in a “GetCapabilities” response.

The following is an example handler for the “GetCapabilities” handler of the fictional WES (Web Example Service):

```
from eoxserver.core import Component, implements, ExtensionPoint
from eoxserver.services.ows.interfaces import (
    ServiceHandlerInterface, GetServiceHandlerInterface,
    PostServiceHandlerInterface
)

class WESGetCapabilitiesHandler(Component):
    implements(ServiceHandlerInterface)
    implements(GetServiceHandlerInterface)
    implements(PostServiceHandlerInterface)

    service = "WES"
    request = "GetCapabilities"
    versions = ["1.0"]

    def handle(self, request):
        ...
```

Note: A word about versions: in EOxServer they are represented by the *Version* class. It follows OGC conventions on treating versions. So for example the versions “1.0” and “1.0.1” are considered equal. For our example this means that our handler will be able to respond to any request with a version “1.0.x”.

¹⁶⁷ <https://docs.djangoproject.com/en/1.8/ref/request-response/#django.http.HttpRequest>

Data Formats

Metadata Formats

The *autotest* instance

Table of Contents

- *The autotest instance* (page 129)
 - *Installation* (page 129)
 - *Fixtures* (page 129)
 - *Deployment* (page 130)
 - *Run tests* (page 130)
 - * *Testing Configuration* (page 130)
 - * *XML Schemas* (page 130)

The *autotest* instance is a preconfigured EOxServer instance used for integration testing. It provides test data and accompanying fixtures, integration test procedures and expected results for test comparison.

Technically it is a whole EOxServer instance with an additional Django app that provides the test code.

The instance is preconfigured, and fixtures can be loaded

Installation

To use the *autotest* instance, make sure that EOxServer was installed. You can obtain it via git:

```
git clone git@github.com:EOxServer/autotest.git
cd autotest
```

or from the projects release page:

```
wget https://github.com/EOxServer/autotest/archive/release-<version>.tar.gz tar -xzvf release-
<version>.tar.gz cd autotest
```

If you just want to run the tests with the default settings you should be fine now and *can start testing* (page 130). If you want to run the instance, you have create the database first:

```
python manage.py syncdb
```

Note: You can run the `syncdb` command with the `--no-input` option and run `python manage.py loaddata auth_data.json` to load the default admin fixtures. This adds an administrator account for the admin app. The username and password is both `admin`. This account is, of course, **not** recommended for productive use.

Fixtures

In order to load the actual data fixtures, run the following commands:

For MERIS UInt16 images:

```
python manage.py loaddata meris_range_type.json meris_coverages_uint16.json
```

For MERIS RGB images:

```
python manage.py loaddata range_types.json meris_coverages_rgb.json
```

For referenceable ASAR images:

```
python manage.py loaddata asar_range_type.json asar_coverages.json
```

To load all available fixtures type:

```
python manage.py loaddata autotest/data/fixtures/*.json
```

Deployment

The autotest instance can be deployed *like any other EOxServer instance* (page 23). The fastest way to actually access the data just run:

```
python manage.py runserver 0.0.0.0:8000
```

Run tests

Running tests does not require any deployment or even a database synchronization. To run all autotest testcases just call:

```
python manage.py test autotest_services -v2
```

If you only want to run a specific test case or only a specific test method run this:

```
python manage.py test autotest_services.WCS20GetCapabilitiesValidTestCase.testValid
```

Testing Configuration

Our basic environment to test EOxServer on is a CentOS 6.5 operating system. On other systems some tests might produce slightly different results, which is due to slight variations of dependency software or 64 to 32 bit architecture differences. For this reason, the following setting can be adjusted to skip binary image comparisons:

```
[testing]
binary_raster_comparison_enabled=false
```

XML Schemas

Many tests of the autotest suite perform XML Schema validation. By default, the schemas will be fetched dynamically, but this really slows down the the tests. Because of this, we prepared a schemas repository that can be downloaded and used instead.

```
wget https://github.com/EOxServer/schemas/archive/<version>.tar.gz
tar -xzvf <version>.tar.gz
export XML_CATALOG_FILES=`pwd`"/schemas-<version>/catalog.xml"
```

SOAP Proxy

Table of Contents

- *SOAP Proxy* (page 131)
 - *Architecture* (page 131)
 - * *Supported Interfaces* (page 131)
 - * *Overview* (page 131)
 - *Implementation* (page 131)

Architecture

Soap_proxy is an adapter proxy which accepts POST request in XML encoded in SOAP 1.2 messages, and passes these on to EOxServer. The proxy may also be configured to pass the messages as POST requests to a suitable mapserver executable instead of an EOxServer, for example for testing purposes.

Supported Interfaces

Soap_proxy uses SOAP 1.2 over HTTP.

EOxServer responds to the following WCS-EO requests through SOAP service interface:

- DescribeCoverage
- DescribeEOCoverageSet
- GetCapabilities
- GetCoverage

Overview

Soap_proxy uses the axis2/C framework. An important feature of axis2/C is that it correctly handles SOAP 1.2 MTOM Attachments.

The overall deployment context is shown in the figure below. Soap_proxy is implemented as an axis2/c service, running within the apache2 httpd server as a mod_axis2 module.

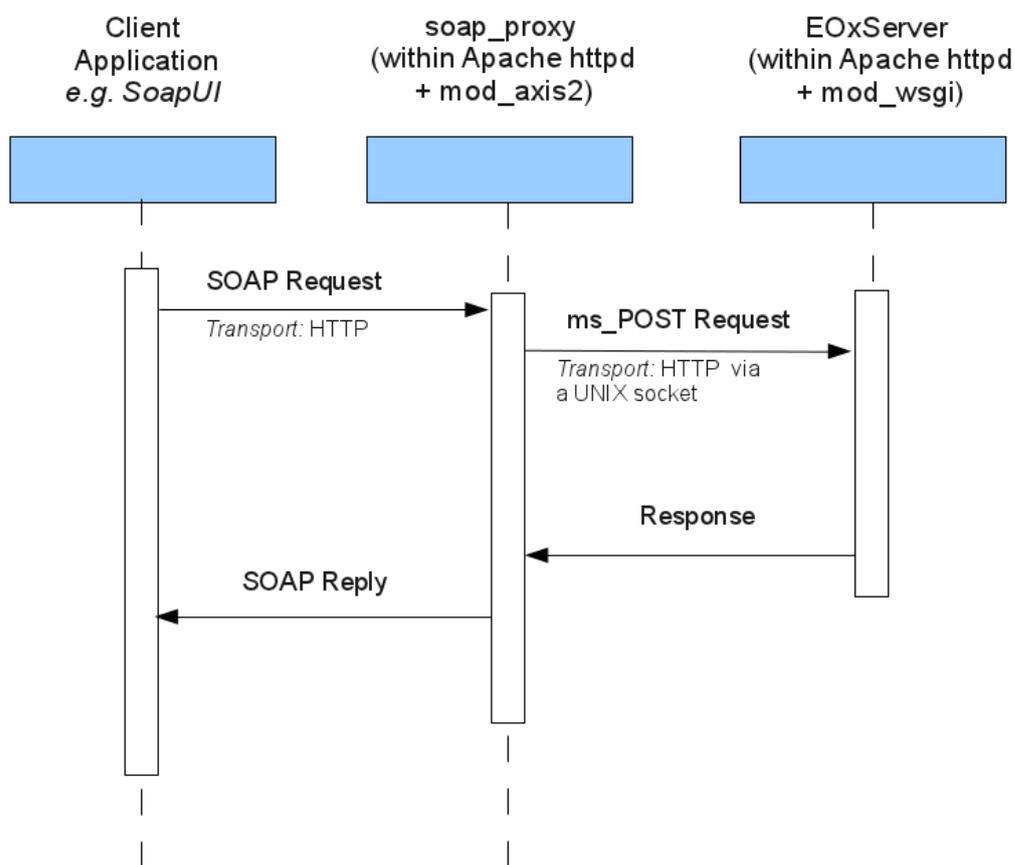
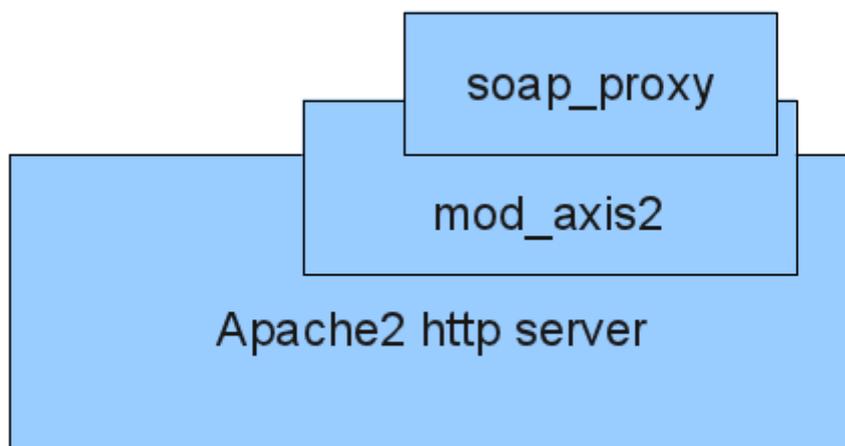
The next figure shows a sequence diagram for a typical request-response message exchange from a client through the soap_proxy to an instance of EOxServer.

Implementation

The implementation is provided in the src directory. The file sp_svc.c is the entry point where the Axis2/c framework calls the soap_proxy implementation code via rpSvc_invoke(), which calls rp_dispatch_op() to do most of the work.

Handling Coverages

This document will explain the basic principles of handling the most important EOxServer data models: coverages. The layout of the data models is explained in its own chapter.



Since all data models in EOxServer are based upon the `django.db.models.Model`¹⁶⁸ class all associated documentation is also applicable to all EOxServer models. Highly recommendable is also the [Django QuerySet documentation](#)¹⁶⁹,

Creating Coverages

As we already mentioned, coverages are basically Django models and are also created as such.

The following example creates a Rectified Dataset.

```
from eoxserver.core.util.timetools import parse_iso8601
from django.contrib.gis import geos
from eoxserver.resources.coverages import models

dataset = models.RectifiedDataset(
    identifier="SomeIdentifier",
    size_x=1024, size_y=1024,
    min_x=0, min_y=0, max_x=90, max_y=90, srid=4326,
    begin_time=parse_iso8601("2014-05-10"),
    end_time=parse_iso8601("2014-05-12"),
    footprint=geos.MultiPolygon(geos.Polygon.from_bbox((0, 0, 90, 90)))
)

dataset.full_clean()
dataset.save()
```

Of course, in a productive environment, all of the above values would come from a actual data and metadata files and would be parsed by *metadata readers* (page 238).

Also, our dataset is currently not linked to any actual raster files. To do this, we need to create at least one `DataItem` and add it to our Dataset.

```
from eoxserver.backends import models as backends

data_item = backends.DataItem(
    dataset=dataset, location="/path/to/your/data.tif", format="image/tiff",
    semantic="bands"
)

data_item.full_clean()
data_item.save()
```

This would link the dataset to a local file with the path `/path/to/your/data.tif`.

Note: Be cautious with relative paths! Depending on the deployment of the server instance the actual meaning of the paths might differ! If you are using `Storages` or `Packages` relative paths are of course okay and unambiguous since they are relative to the package or storage base location.

If you want to set up a data item that resides in a package (such as a .zip or .tar file) or on a storage (like an HTTP or FTP server) you would need to set up the `Packages` or `Storages`:

```
http_storage = backends.Storage(
    url="http://example.com/base_path/",
    storage_type="HTTP"
)
http_storage.full_clean()
```

¹⁶⁸ <https://docs.djangoproject.com/en/1.8/ref/models/instances/#django.db.models.Model>

¹⁶⁹ <https://docs.djangoproject.com/en/dev/ref/models/querysets/>

```
http_storage.save()

data_item.storage = http_storage
data_item.full_clean()
data_item.save()

# *or* in case of a package

zip_package = backends.Package(
    location="/path/to/package.zip",
    format="ZIP"
)
zip_package.full_clean()
zip_package.save()

data_item.package = zip_package
data_item.full_clean()
data_item.save()
```

Note: A `DataItem` can only be in either a `storage` *or* a `package`. If it has defined both a `storage` and a `package`, the `storage` has precedence. If you want to have a `Package` that resides on a `Storage` you must use the `storage` of the `Package`.

Creating Collections

Collections are also created like `Coverages`, but usually require less initial information (because the metadata is usually collected from all entailed datasets).

The following creates a `DatasetSeries`, a collection that can entail almost any object of any subtype of `EOObject`.

```
dataset_series = models.DatasetSeries(identifier="CollectionIdentifier")
dataset_series.full_clean()
dataset_series.save()
```

The handling of collections is fairly simple: you use `insert()` to add a dataset or subcollection to a collection and use `remove()` to remove them. Whenever either of the action is performed, the EO metadata of the collection is updated according to the entailed datasets.

```
dataset_series.insert(dataset)
dataset_series.footprint # is now exactly the same as dataset.footprint
dataset_series.begin_time # is now exactly the same as dataset.begin_time
dataset_series.end_time # is now exactly the same as dataset.end_time

dataset_series.remove(dataset)
dataset_series.footprint # is now None
dataset_series.begin_time # is now None
dataset_series.end_time # is now None
```

Accessing Coverages

The simplest way to retrieve a coverage is by its ID:

```
from eoxserver.resources.coverages import models

dataset = models.Coverage.objects.get(identifier="SomeIdentifier")
```

This always returns an object of type `Coverage`, to “cast” it to the actual type:

```
dataset = dataset.cast()
```

Note: the `cast()` method only makes a database lookup if the actual type and the current type do not match. Otherwise (and only in this case), the object itself is returned and no lookup is performed.

If you know the exact type of the coverage you want to look up you can also make the query with the desired type:

```
dataset = models.RectifiedDataset.objects.get(identifier="SomeIdentifier")
```

If the `get()` query did not match any object (or possible more than one) an exception is raised.

If you want to query more than one coverage at one (e.g: all coverages in a certain time period) the `filter()` method is what you want:

```
from eoxserver.core.util.timetools import parse_iso8601

start = parse_iso8601("2014-05-10")
stop = parse_iso8601("2014-05-12")
coverages_qs = models.Coverage.objects.filter(
    begin_time__gte=start, end_time__lte=stop
)
for coverage in coverages_qs:
    ... # Do whatever you like with the coverage
```

Note: `filter()` returns a Django `QuerySet`¹⁷⁰ which can be chained to further refine the actual query. There is a lot of [documentation on the topic](#)¹⁷¹ I highly recommend.

Usually coverages are organized in collections. If you want to iterate over a collection simply do so:

```
dataset_series = models.DatasetSeries.objects.get(
    identifier="CollectionIdentifier"
)
for eo_object in dataset_series:
    ...
```

It is important to note that such an iteration *does not* yield coverages, but `EOObjects`. This is due to the fact that collections might also contain other collections that don’t necessarily have to inherit from `Coverage`. If you just want to explicitly get all `Coverages` from a collection you can do it like this:

```
coverages_qs = models.Coverage.objects.filter(
    collections__in=[dataset_series.pk]
)
```

You can also combine the filters for searches within a collection:

```
coverages_qs = dataset_series.eo_objects.filter(
    begin_time__gte=start, end_time__lte=stop
)
# append an additional geometry search
coverages_qs = coverages_qs.filter(
    footprint__intersects=geos.Polygon.from_bbox((30, 30, 40, 40))
)
```

¹⁷⁰ <https://docs.djangoproject.com/en/1.8/ref/models/querysets/#django.db.models.query.QuerySet>

¹⁷¹ <https://docs.djangoproject.com/en/dev/topics/db/queries/>

Note: There is no intrinsic order of EOObjects in a Collection, but the EOObjects can be sorted when they are retrieved from a collection. (e.g: by identifier, begin_time or end_time) using the QuerySets `order_by()` method.

Accessing Coverage Data

As already discussed, the actual data and metadata files of a coverage are referenced via its associated DataItems. First, it is necessary to select the DataItems that are actually relevant. This depends on the current situation: for example in a metadata oriented request (such as the WCS DescribeCoverage operation) only metadata items will be accessed (and only if they are of relevance):

```
metadata_items = dataset.data_items.filter(
    semantic="metadata", format="eogml"
)
```

The above example selected only metadata items with the format “eogml”.

In some cases the bands of a coverage are separated into multiple files that have a semantic like this: “bands[x:y]”. To select only those, we can use the [startswith field lookup](#)¹⁷²:

```
band_items = dataset.data_items.filter(
    semantic__startswith="bands"
)
for band_item in band_items:
    # TODO: parse the band index or start/stop indices
    ...
```

Now that we have our relevant DataItems we can start using them.

We also explained that the DataItems can reside on a Storage or inside a Package. Each storage has a specific storage type and each package has a specific format. What types and formats are available depends on your instance configuration, since the formats are implemented as Components. EOxServer ships with support of local, HTTP, FTP and Rasdaman storages and with ZIP and TAR packages. This list of both storages and packages can be easily extended by creating plugin Components implementing either the *FileStorageInterface* (page 230), *ConnectedStorageInterface* (page 230) or the *PackageInterface* (page 231). See the *documentation for writing Plugins* (page 126) for further info.

To ease the actual data access, there are two main methods: `retrieve()` and `connect()`.

Both functions have in common, that they operate on DataItems which are passed as the first parameter to the function.

The function `retrieve()` returns a path to the local file: for already local files, the path is simply passed, in other cases the file is downloaded, unpacked, retrieved or whatever is necessary to make the file locally accessible.

```
data_item = dataset.data_items.get(semantic="metadata")
local_path = retrieve(data_item)
```

You do not have to care for cleanup afterwards, since this is handled by EOxServers cache layer.

The function `connect()` works similarly, apart from the fact that it takes also storages into account that do not provide files, but streams of data. Currently this only includes the Rasdaman Storage. If this function does not deal with a *Connected Storages* (page 230) it behaves like the `retrieve()` function.

Processes

This section deals with the creation of new Processes to be exposed via the WPS interface.

¹⁷² <https://docs.djangoproject.com/en/dev/ref/models/querysets/#std:fieldlookup-startswith>

Processes are simply Components that implement the *ProcessInterface* (page 253). For general information about Plugins/Components please refer to the *Plugins* (page 126) documentation.

Creating a new Process

As we already mentioned, Processes are Components:

```
from eoxserver.core import Component, implements
from eoxserver.services.ows.wps.interfaces import ProcessInterface

class ExampleProcess (Component):
    implements (ProcessInterface)
    ...
```

Apart from some optional metadata and a mandatory identifier, each Process has specific input parameters and output items. Those can be of various formats and complexity. Each input and output must be declared in the processes section. Let's start with a simple example, using LiteralData inputs and outputs:

```
from eoxserver.services.ows.wps.parameters import LiteralData

class ExampleProcess (Component):
    implements (ProcessInterface)

    identifier = "ExampleProcess"
    title = "Example Title."
    metadata = {"example-metadata": "http://www.metadata.com/example-metadata"}
    profiles = ["example_profile"]

    inputs = [
        ("example_input", LiteralData(
            'example_input', str, optional=True,
            abstract="Example string input.",
        ))
    ]

    outputs = [
        ("example_output", LiteralData(
            'example_output', str,
            abstract="Example string output.", default="n/a"
        )),
    ]

    ...
```

LiteralData inputs will always try to parse the input to the defined type. E.g: if you defined your input type to float, an error will be raised if the supplied parameters could not be passed. On the other hand, all your outputs will be properly encoded and even translated to a specific unit if requested. Your execute function will not need to hassle with type conversions of any kind for your inputs/outputs.

Now that we have defined a Process with metadata, inputs and outputs we can start writing the `execute` method of our Process. Each input parsed before it is passed to our `execute` method where it is mapped to a named parameter

Our `execute` method is expected to return either a normal Python object if we only declared a single output, or a `dict`¹⁷³ of outputs where the keys are the names of our declared outputs:

```
class ExampleProcess (Component):
    implements (ProcessInterface)

    ...
```

¹⁷³ <https://docs.python.org/2.7/library/stdtypes.html#dict>

```

inputs = [
    ("example_input", LiteralData(
        'example_input', str, optional=True,
        abstract="Example string input.",
    ))
]

outputs = [
    ("example_output", LiteralData(
        'example_output', str,
        abstract="Example string output.", default="n/a"
    )),
]

def execute(self, **inputs):
    outputs = {}
    outputs["example_output"] = "Echo '%s'" % inputs["example_input"]
    return outputs

```

Another often used type for Processes are BoundingBoxes. They are declared as follows:

```

from eoxserver.core import Component, implements
from eoxserver.services.ows.wps.interfaces import ProcessInterface
from eoxserver.services.ows.wps.parameters import (
    BoundingBoxData, BoundingBox
)

class ExampleProcess(Component):
    implements(ProcessInterface)

    ...

    inputs = [
        ("example_bbox_input", BoundingBoxData(
            "example_bbox_input", crss=(4326, 3857),
            default=BoundingBox([[-90, -180], [+90, +180]]),
        )),
    ]
    outputs = [
        ("example_bbox_output", BoundingBoxData(
            "example_bbox_output", crss=(4326, 0)
        )),
    ]
    ...

```

The third kind of input and output is ComplexData which can come in various formats, binary or textual representation and either raw or base64 encoding.

Asynchronous Task Processing - Developers Guide

Table of Contents

- *Asynchronous Task Processing - Developers Guide* (page 138)
 - *Introduction* (page 139)
 - *Simple ATP Application* (page 139)

- * [Step 1 - Handler Subroutine](#) (page 139)
- * [Step 2 - New Task Type Registration](#) (page 140)
- * [Step 3 - Creating New Task](#) (page 140)
- * [Step 4 - Polling the task status](#) (page 140)
- * [Step 5 - Getting the logged task history](#) (page 141)
- * [Step 6 - Getting the task results](#) (page 141)
- * [Step 7 - Removing the task](#) (page 141)
- [Executing ATP Task](#) (page 141)
 - * [Pulling a task from queue](#) (page 141)
 - * [Task Execution](#) (page 142)
 - * [DB Cleanup](#) (page 142)

Introduction

This guide is intended to help with the creation of applications using the *Asynchronous Task Processing* subsystem of EOxServer.

The first part is guiding creation of the simple task producer, i.e., an application needing the asynchronous processing capabilities.

The second part helps with creation of a task consumer, i.e., the part of code pulling tasks from the work queue and executing them. The task consumer is part of Asynchronous Task Processing Daemon.

An overview of the ATP capabilities is presented in “*Asynchronous Task Processing* (page 113)”. The database model used in by the ATP subsystem is described in “*Task Tracker Data Model* (page 125)”. The complete API reference can be found in “`eoxserver.resources.processes.tracker`”.

Simple ATP Application

Here in this section we will prepare step-by-step a simple demo application making use of the ATP subsystem. The complete application is available at location:

```
<EOxServer instal.dir.>/tools/atp_demo.py
```

The prerequisite of starting the application is that the correct path to the *EOxServer* installation and instance is set together with the correct *Django* settings module.

Initially the application must import the right python objects from the `tracker()` module:

```
from eoxserver.resources.processes.tracker import \
    registerTaskType, enqueueTask, QueueFull, \
    getTaskStatusByIdentifier, getTaskResponse, deleteTaskByIdentifier
```

By this command we imported following objects: i) task type registration function, ii) the task creation (`enqueue`) subroutine, iii) an exception class risen in case of full task queue unable to accept (most likely temporarily) new tasks, iv) task’s status polling subroutine, v) the response getter function and finally vi) the subroutine deleting an existing task. These are the ATP Python objects needed by our little demo application.

Step 1 - Handler Subroutine

Let’s start with preparation of an example of subroutine to be executed - handler subroutine. The example handler below sums sequence of numbers and stores the result:

```
def handler( taskStatus , input ) :
    """ example ATP handler subroutine """
    sum = 0
    # sum the values
    for val in input :
        try :
            sum += float( val )
        except ValueError:
            # stop in case on ivalid input
            taskStatus.setFailure("Input must be a sequence of numbers!")
            return
    # store the response and terminate
    taskStatus.storeResponse( str(sum) )
```

Any handler subroutine (see also `dummyHandler()`) receives two parameters: i) an instance of the `TaskStatus` class and an input parameter. The input parameter is set during the task creation and can be any Python object serialisable by the `pickle` module.

Step 2 - New Task Type Registration

Once we have prepared the handler subroutine we can register the task type to be performed by this subroutine:

```
registerTaskType( "SequenceSum" , "tools.atp_demo.handler" , 60 , 600 , 3 )
```

The `registerTaskType()` subroutine registers a new task type named "SequenceSum". Any task instance of this task type will be processed by the handler subroutine. The handler subroutine is specified as importable module path. Any task instance not processed by an ATPD within 60 seconds (measured from the moment the ATPD pulls a task from the queue) is considered to be abandoned and it is automatically re-enqueued for new processing. The number of the re-enqueue attempts is limited to 3. Once a task instance is finished it will be stored for min. 10 minutes (600 seconds) before it gets removed.

Step 3 - Creating New Task

Once the task handler has been registered as a new task type we can create a task's instance:

```
while True :
    try:
        enqueueTask( "SequenceSum" , "Task001" , (1,2,3,4,5) )
        break
    except QueueFull : # retry if queue full
        print "QueueFull!"
        time.sleep( 5 )
```

The `enqueueTask()` creates a new task instance "Task001" of task type "SequenceSum". The tuple `(1, 2, 3, 4, 5)` is the input to the handler subroutine. In case of full task queue new task cannot be accepted and the `QueueFull()` is risen. Since we want the task to be enqueued a simple re-try loop must be employed.

Step 4 - Polling the task status

After task has been created enqueued for processing its status can be polled:

```
while True :
    status = getTaskStatusByIdentifier( "SequenceSum" , "Task001" )
    print time.asctime() , "Status: " , status[1]
    if status[1] in ( "FINISHED" , "FAILED" ) : break
    time.sleep( 5 )
```

The task status is polled until the final status (FINISHED or FAILED) is reached. The task must be identified by unique pair of task type and task instance identifiers.

NOTE: The task instance is guaranteed to be unique for given task type identifier, i.e., there might be two task with the same instance identifier but different type identifier.

Step 5 - Getting the logged task history

The history of the task processing is logged and the log messages can be extracted by `getTaskLog()` function:

```
print "Processing history:"
for rec in getTaskLog( "SequenceSum" , "Task001" ) :
    print "-" , rec[0] , "Status: " , rec[1][1] , "\t" , rec[2]
```

This function returns list of log records sorted by time (older first).

Step 6 - Getting the task results

Once the task has been finished the task response can be retrieved:

```
if status[1] == "FINISHED" :
    print "Result: " , getTaskResponse( "SequenceSum" , "Task001" )
```

Step 7 - Removing the task

Finally, the result task is not needed any more and can be removed from DB:

```
deleteTaskByIdentifier( "SequenceSum" , "Task001" )
```

Executing ATP Task

In this section we will briefly describe all the steps necessary to pull and execute task instance from the queue. As working example we encourage you the source Python code of the ATPD located at:

```
<EOxServer instal.dir.>/tools/asyncProcServer.py
```

The invocation of the ATP server is described in “*Asynchronous Task Processing* (page 113)”.

Initially the application must import the python objects from the `tracker` module:

```
from eoxserver.resources.processes.tracker import *
```

For convenience we have made available whole content of the module.

Pulling a task from queue

The ATPD is expected to pull task from the queue repeatedly. For simplicity we avoid the loop definition and we will rather focus on the loop body. Following command pulls a list of tasks from queue:

```
try:
    # get a pending task from the queue
    taskIds = dequeueTask( SERVER_ID )
except QueueEmpty : # no task to be processed
    # wait some ammount of time
    time.sleep( QUEUE_EMPTY_QUERY_DELAY )
    continue
```

This command tries to pull exactly one task at time from the DB queue but the applied mechanism of pulling does not guaranties that none or more than one task would be return. Thus the dequeuing function returns a list of tasks and the implementation must take this fact into account. Further, the dequeue function requires unique ATPD identifier (SERVER_ID).

The `dequeueTask()` function changes automatically the status from `ENQUEUED` to `SCHEDULED` and log the state transition. The optional logging message can be provided.

Task Execution

In case we have picked one of the pulled tasks and stored it to `taskId` variable we can proceed with the task execution:

```
# create instance of TaskStatus class
pStatus = TaskStatus( taskId )
try:
    # get task parameters and change status to STARTED
    requestType , requestID , requestHandler , inputs = startTask( taskId )
    # load the handler
    module , _ , funct = requestHandler.rpartition(".")
    handler = getattr( __import__(module,fromlist=[funct]) , funct )
    # execute handler
    handler( pStatus , inputs )
    # if no terminating status has been set do it right now
    stopTaskSuccessIfNotFinished( taskId )
except Exception as e :
    pStatus.setFailure( unicode(e) )
```

In order to execute the task couple of actions must be performed. First an instance of the `TaskStatus` class must be created.

The parameters of the task (task type identifier, task instance identifier, request handler and task inputs) must be retrieved by the `dequeueTask()` function. The function also changes the status of the task from `SCHEDULED` to `RUNNING` and logs the state transition automatically.

The handler “dot-path” must be split to module and function name and loaded dynamically by the `__import__()` function.

Once imported the handler function is executed passing the `TaskStatus` and inputs as the arguments.

The handler function is allowed but not required to set the successful terminal state of the processing (`FINISHED`) and if not set it is done by the `stopTaskSuccessIfNotFinished()` function.

Obviously, the implementation must catch any possible Python exception and record the failure (`try-except` block).

DB Cleanup

In addition to the normal operation each ATPD implementation is responsible for maintenance of the ATPD subsystem in a consistent state. Namely, i) the ATPD must repeatedly check for the abandoned “zombie” tasks and restart them by calling `reenqueueZombieTasks()` function and ii) the ATPD must remove DB records of the finished “retired” tasks by calling `deleteRetiredTasks()` function.

Testing

TBD

```
eoxserver.testing.core
```

EOxServer code style guide

This document tries to establish a set of rules to help harmonizing the source code written by many contributors. The goal is to improve compatibility and maintainability.

Fundamentals

Above all rules, adhere the rules defined in the [Python PEP 8¹⁷⁴](#). Please try to adhere the mentioned code styles. You can check if you compliant to the style guide with the `pylint` or `pep8` command line utilities.

Then:

```
>>> import this
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
```

Package layout and namespacing

Use Python package structures to enable hierarchical namespaces. Do not encode the namespace in function or class names.

E.g: don't do this:

```
# somemodule.py

def myNS_FunctionA():
    pass

class myNS_ClassB():
    pass
```

Instead, do this:

```
# somemodule/myNS.py

def functionA():
    pass
```

¹⁷⁴ <https://www.python.org/dev/peps/pep-0008/>

```
class ClassB():
    pass
```

A developer using these functions can choose to use the namespace explicitly:

```
from somepackage import myNS

myNS.functionA()
c = myNS.ClassB()
```

Import rules

As defined in Python PEP 8, place all imports in the top of the file. This makes it easier to trace dependencies and allows to see and resolve importing issues.

Try to use the following importing order:

1. Standard library imports or libraries that can be seen as industry standard (like numpy).
2. Third party libraries (or libraries that are not directly associated with the current project). E.g: GDAL, Django, etc.
3. Imports that are directly associated with the current project. In case of EOxServer, everything that is under the *eoxserver* (page 261) package root.

Use single empty lines to separate these import groups.

Coding guidelines

Minimizing pitfalls

Don't use mutable types as default arguments

As default arguments are evaluated at the time the module is imported and not when the function/method is called, default arguments are a sort of global variable and calling the function can have unintended side effects. Consider the following example:

```
def add_one(arg=[]):
    arg.append(1)
    print arg
```

When called with no arguments, the function will print different results every time it is invoked. Also, since the list will never be released, this is also a memory leak, as the list grows with the time.

Don't put code in the `__init__.py` of a package

When importing a package or a module from a package, the packages `__init__.py` will first be imported. If there is production code included (which will likely be accompanied by imports) this can lead to unintended circular imports. Try to put all production code in modules instead, and use the `__init__.py` only for really necessary stuff.

Use abbreviations sparingly

Try not to use abbreviations, unless the meaning is commonly known. Examples are HTTP, URL, WCS, BBox or the like.

Don't use leading double underscores to specify 'private' fields or methods or module functions, unless *really* necessary (which it isn't, usually). Using double underscores makes it unnecessarily hard to debug methods/fields and is still not really private, as compared to other languages like C++ or Java. Use single leading underscores instead. The meaning is clear to any programmer and it does not impose any unnecessary complications during debugging.

Improving tests

General rules

Implementing new features shall *always* incorporate writing new tests! Try to find corner/special cases and also try to find cases that shall provoke exceptions.

Where to add the tests?

Try to let tests *fail* by calling the correct assertion or the `fail` functions. Don't use exceptions (apart from `AssertionError`), because when running the tests, this will be visible as "Error" and not a simple failure. Test errors should indicate that something completely unexpected happened that broke the testing code.

Requests for Comments

EOxServer Requests for Comments (RFCs) are a means for EOxServer developers to share their ideas and feature requests, propose enhancements, and discuss high-level issues concerning the further development of the software.

RFC Procedures

See the *RFC Policies* (page 148) for details.

Writing RFCs

If you want to write a Request for Comments, please read the *Guidelines for Requests for Comments* (page 150) first.

RFCs

Table: “*List of accepted EOxServer RFCs* (page 148)” below lists all accepted EOxServer RFCs and their implementation status¹.

¹ Note that this list might not be fully up to date although we try hard.

Table 3.1: List of accepted EOxServer RFCs

No.	Title	Status
0	<i>RFC 0: Project Steering Committee Guidelines</i> (page 153)	Effective
1	<i>RFC 1: An Extensible Software Architecture for EOxServer</i> (page 156)	Implemented in version 0.2
2	<i>RFC 2: Extension Mechanism for EOxServer</i> (page 163)	Implemented in version 0.2
6	<i>RFC 6: Directory Structure</i> (page 173)	Implemented in version 0.2
7	<i>RFC 7: Release Guidelines</i> (page 178)	Effective
8	<i>RFC 8: SVN Commit Management</i> (page 180)	Effective
9	<i>RFC 9: SOAP Binding of WCS GetCoverage Response</i> (page 182)	Implemented in SOAP Proxy
10	<i>RFC 10: SOAP Proxy</i> (page 184)	Implemented in SOAP Proxy
12	<i>RFC 12: Backends for the Data Access Layer</i> (page 188)	Implemented in version 0.2
13	<i>RFC 13: WCS-T 1.1 Interface Prototype</i> (page 192)	Implemented in version 0.2
14	<i>RFC 14: Asynchronous Task Processing (ATP)</i> (page 196)	Implemented in version 0.2
15	<i>RFC 15: Access Control Support</i> (page 199)	Implemented in version 0.2
16	<i>RFC 16: Referenceable Grid Coverages</i> (page 201)	Implemented in version 0.2
17	<i>RFC 17: Configuration of Supported Output Formats and CRSes</i> (page 204)	Implemented in version 0.3
19	<i>RFC 19: Migrate project repository from svn to git</i> (page 221)	Effective

The list below provides links to all EOxServer RFCs available:

RFC Policies

Author Stephan Krause, Stephan Meißl

Date 2011-05-13

This document contains the policies that govern the life cycle of Requests for Comments (RFCs). It may be changed by submitting an RFC for discussion and vote following the provisions of this document.

In this document the terms *shall*, *should* and *may* have a normative meaning, that is well known from software engineering and standards definition:

- *shall*: indicates an absolute requirement to be strictly followed
- *should*: indicates a recommendation
- *may*: indicates an option

Status of RFCs

Every RFC has a status. That status may be one of:

- **IN PREPARATION**: Some text for the RFC has been posted, but that is not the version to be submitted for discussion and voting. An RFC that has this status is still work in progress.
- **PENDING**: The text of the RFC has been submitted for discussion. It may still be altered by the RFC authors in order to reflect the state of the discussion.
- **WITHDRAWN**: The text of the RFC has been withdrawn.
- **VOTING ACTIVE**: The text of the RFC has been frozen and voting is going on.
- **ACCEPTED**: A vote has been held on the RFC and it has been accepted. Implementation has started.
- **REJECTED**: A vote has been held on the RFC and it has been rejected. The RFC is not going to be implemented and the discussion is closed.
- **POSTPONED**: A vote has been held on the RFC and it has been postponed to a later stage of development. The RFC may be reopened any time.

- **OBSOLETE:** A vote has been held on the RFC and it has been declared obsolete. It has been superseded by another RFC or it is not considered applicable any more.

The status IN PREPARATION may be declared by the authors of the RFC. They may move it to PENDING once they consider it ready for discussion and submission to a vote. Any further status changes shall be declared according to the results of the discussion and the voting (see *RFC 0: Project Steering Committee Guidelines* (page 153)).

The following status changes are possible:

- from IN PREPARATION to PENDING, WITHDRAWN
- from PENDING to WITHDRAWN or via VOTING ACTIVE to ACCEPTED, REJECTED, POSTPONED
- from ACCEPTED via VOTING ACTIVE to PENDING, POSTPONED, OBSOLETE
- from POSTPONED to PENDING or via VOTING ACTIVE to ACCEPTED, REJECTED, OBSOLETE

Creation of RFCs

Any one who has write access to the EOxServer SVN may submit an RFC. It shall obey the rules of the *Guidelines for Requests for Comments* (page 150). The initial status of the RFC is IN PREPARATION, lest the authors deem it to be mature for discussion from the start, in which case they may submit it as PENDING. The RFC shall be assigned the next possible consecutive number.

When beginning work on an RFC the authors shall inform the PSC chair.

As long as the RFC is IN PREPARATION or PENDING, only the authors of the RFC shall edit it. Anyone else who wants to contribute to the document shall submit his or her text to the discussion page. The authors may also decide to let him or her become a co-author who has all the rights of an author.

Authors may choose to support their RFC by implementing the needed changes and committing them to a subdirectory of the sandbox directory for review.

Discussion Pages

Any RFC, especially those still IN PREPARATION, shall have a discussion page on the EOxServer Trac Wiki (<http://eoxserver.org/wiki>). The design and the location of the discussion page is detailed in the *Guidelines for Requests for Comments* (page 150).

The discussion page may include links to preliminary implementations which have been committed to a sandbox subdirectory.

Pending RFCs

PENDING RFCs are submitted for discussion. They may still be edited to reflect the state of the discussion or to correct errors. They should not be altered in a radical manner though, changing the proposed solution completely. In this case the authors may withdraw the RFC and propose another one.

An RFC shall be PENDING for at least two business days (in Austria) till a vote can be held on it (see *RFC 0: Project Steering Committee Guidelines* (page 153)).

Withdrawal of RFCs

The authors may withdraw an RFC at any time as long as it is IN PREPARATION or PENDING. The RFC status will change to WITHDRAWN. The authors may decide to leave the text as is or remove everything except for the basic information as defined in the *Guidelines for Requests for Comments* (page 150).

Voting on RFCs

The voting on RFCs is defined in the first RFC: *RFC 0: Project Steering Committee Guidelines* (page 153).

Guidelines for Requests for Comments

Author Stephan Krause

Date 2011-02-19

Last Edit \$Date\$

Discussion <http://eoxserver.org/wiki/DiscussionRfcTemplate>

This document contains instructions for writing RFCs as well as a template for RFCs. Please read it carefully before submitting your own requests.

In this document the terms *shall*, *should* and *may* have a normative meaning that is well known from software engineering and standards definition:

- *shall*: indicates an absolute requirement to be strictly followed
- *should*: indicates a recommended item
- *may*: indicates an optional item

Location of an RFC

The text of an RFC shall be located in the EOxServer SVN Trunk in the directory `docs/en/rfc` under the file name `rfc<number>.rst`. It will be published automatically on the Request For Comments site once the documentation has been built anew.

Discussion Page

Once the RFC status has been moved to PENDING, it is required that the authors create a discussion page for the RFC on the EOxServer Trac Wiki. A *Template for RFC Discussion Pages* (page 152) is included below.

Structure of an RFC

Heading

The page heading shall be in the format “RFC <number>: <title>”.

Basic Information

The RFC shall start with a block containing the author(s) of the request, the creation date, the date of the last edit and its status, like in the following example:

Author John Doe

Created 2011-02-18

Last Edit \$Date\$

Status PENDING

Discussion <http://eoxserver.org/wiki/DiscussionRfcTemplate>

Description of the RFC

The first one or two paragraphs shall contain a short description of the RFC. They should give a high-level overview of the propositions of the request.

Introduction

The first section of the RFC shall be called “Introduction”. It should contain a motivation for the RFC, describe the problem(s) the RFC addresses and give an overview of the proposed solution. It should contain forward references to the sections where specific items are discussed further where applicable.

Keep the introduction short and simple! It is not the place to go into the details, this should be done in the sections of the body of the RFC.

Body of the RFC

The body of the RFC starts right after the introduction. It may start with a more in-depth description of the motivation for the RFC and the problems to address if this cannot be discussed exhaustively in the introduction. Following that the proposed solution should be described in detail and as vividly as possible.

Use examples, tables and pictures where appropriate! Use references to external resources, to the documentation, to other RFCs, to the EOxServer Trac or to the source code.

The body of the RFC may be contained in one section or structured in sections, subsections and subsubsections or even further.

Voting History

The penultimate section of the RFC shall be called “Voting History”. It shall contain the records of the votes held on subject of the RFC. As long as the RFC is in preparation or pending, the section body shall be “N/A”. Example of a voting record:

Motion To accept RFC 1
Voting Start 2011-03-01
Voting End 2011-03-02
Result 3 ACCEPTED, 0 PENDING

Traceability

The last section of the RFC shall be called “Traceability”. It shall contain references to the requirements that have motivated the request if applicable. Furthermore, if the request was accepted, it shall contain references to the tickets in the EOxServer Trac system that concern its implementation. Example:

Requirements O3S_CAP_100
Tickets #1

Where possible, the requirements and tickets shall be hyperlinked to the respective resources (e.g. requirements document, requirement tracing system, EOxServer Trac).

Template for RFCs

Here is a template you should use for your RFCs. Please replace the items in brackets <> by the appropriate text:

```

.. _rfc_<number>:

RFC <number>: <title>
=====

:Author: <author name>
:Created: <date when RFC was created: YYYY-MM-DD>
:Last Edit: <date of last edit: YYYY-MM-DD, please use subversion keyword "Date">
>Status: <one of: IN PREPARATION, PENDING, WITHDRAWN, VOTING ACTIVE,
        ACCEPTED, REJECTED, POSTPONED, OBSOLETE>
:Discussion: <external link to discussion page on EOxServer Trac>

<short description of the RFC>

Introduction
-----

<Mandatory. Overview of motivation, addressed problems and proposed
solution>

<Section title>
-----

<Any number of sections may follow.>

<Subsection title>
~~~~~

<They may have any number of subsections.>

<Subsubsection title>
^^^^^^

<And even subsubsections.>

Voting History
-----

<Voting Records or "N/A">

:Motion: <Text of the motion>
:Voting Start: <YYYY-MM-DD>
:Voting End: <YYYY-MM-DD>
:Result: <Result>

Traceability
-----

:Requirements: <links to requirements or "N/A">
:Tickets: <links to tickets or "N/A">

```

Template for RFC Discussion Pages

RFC Discussion pages shall have the URL <http://eoxserver.org/wiki/DiscussionRfc<number>>. They shall be referenced on the page <http://eoxserver.org/wiki/RfcDiscussions>.

```

= Discussion Page RFC <number>: <title> =

'''RFC <number>:''' [<link>]

== Template Comment ==

```

```
<comment text>

''Author: <author name> | Created: <date and time of creation: YYYY-MM-DD HH:MM:SS>
↔ ''
-----

== Discussion ==
```

RFC 0: Project Steering Committee Guidelines

Author Stephan Meißl

Created 2011-03-02

Last Edit 2011-05-17

Status ACCEPTED

Discussion <http://www.eoxserver.org/wiki/DiscussionRfc0>

Overview

This RFC documents the EOxServer Project Steering Committee Guidelines.

(Credit: Inspired by the MapServer PSC guidelines at: <http://mapserver.org/development/rfc/ms-rfc-23.html>)

Introduction

This RFC describes how the EOxServer Project Steering Committee (PSC) handles membership and makes decisions on all aspects, technical and non-technical, of the EOxServer project.

The PSC duties include:

- defining and deciding on the overall development road map
- defining and deciding on technical standards and policies like file naming conventions, coding standards, etc.
- establishing a regular release cycle
- reviewing and voting on RFCs

The PSC members vote on proposals, RFCs, etc. via e-mail on the dev mailing list. Proposals shall be available for review for at least two days where a single veto delays the progress but at the end a majority of members may adopt a proposal.

Voting

Voting Procedure

The following steps shall be followed in any voting:

- Any interested person may submit a proposal to the dev mailing list for discussion. Note that this is explicitly not limited to PSC members.
- Voting on proposals shall not be closed earlier than two business days after the proposal has been submitted.
- The following voting options shall be used:
 - “+1” .. support willingness to support implementation

- "+0" .. low support
 - "0" .. no opinion
 - "-0" .. low disagreement
 - "-1" .. veto
- A veto shall include clear reasoning and alternative approaches to solve the problem at hand.
 - Any interested person may comment on proposals but only votes from PSC members will be counted.
 - A proposal may be declared accepted if it receives at least +2 and not vetoes (-1).
 - Vetoes proposals that cannot be revised to satisfy all PSC members may be submitted for an override vote. The proposal may be declared accepted if a simple majority of eligible voters votes in favor (+1). Eligible voters are all PSC members that have not been declared inactive. However, it is intended that in normal circumstances vetoers are convinced to withdraw their veto. We are trying to reach consensus.
 - Any eligible voter who has not cast a vote in the last two votes shall be considered inactive. Casting a vote immediately turns the status to active.
 - Upon completion of discussion and voting the author shall announce the new status of the proposal (accepted, withdrawn, rejected, postponed, obsolete).
 - The PSC Chair is responsible for keeping track of who is a member of the PSC Membership.
 - Addition and removal of members from the PSC, as well as selection of a Chair should be handled as a proposal to the PSC.
 - The PSC Chair adjudicates in cases of disputes about voting.

Voting is Required for

- any change to committee membership (adding members, removing inactive members).
- creating and dissolving of sub-committees (e.g. to manage conferences, documentation, or web sites).
- changes to project infrastructure (e.g. tool, location, or substantive configuration).
- anything that could cause backward compatibility issues.
- adding substantial amounts of new code.
- changing inter-subsystem APIs, or objects.
- issues of procedure.
- when releases should take place.
- anything dealing with relationships with external entities such as MapServer or OSGeo.
- anything that might be controversial.

PSC Membership

The PSC is made up of individuals consisting of technical contributors (e.g. developers) and prominent members of the EOxServer user community. There is no fixed number of members for the PSC.

Adding Members

Any member of the dev mailing list may nominate someone for committee membership at any time. Only existing PSC committee members may vote on new members. Nominees must receive a majority vote from existing members to be added to the PSC.

Stepping Down

If, for any reason, a PSC member is not able to fully participate then they certainly are free to step down. If a member is not active (e.g. no voting, no IRC, or e-mail participation) for a period of two months then the committee reserves the right to vote to cease membership. Should that person become active again then they are certainly welcome, but require a nomination.

Membership Responsibilities

Guiding Development

Members should take an active role guiding the development of new features they feel passionate about. Once a change request has been accepted and given a green light to proceed does not mean the members are free of their obligation. PSC members voting “+1” for a change request are expected to stay engaged and ensure the change is implemented and documented in a way that is most beneficial to users. Note that this applies not only to change requests that affect code, but also those that affect the web site, technical infrastructure, policies, and standards.

IRC Meeting Attendance

PSC members are expected to participate in pre-scheduled IRC development meetings. If known in advance that a member cannot attend a meeting, the member should let the meeting organizer know via e-mail.

Mailing List Participation

PSC members are expected to be active on both the users and dev mailing lists, subject to open source mailing list etiquette. Non-developer members of the PSC are not expected to respond to coding level questions on the developer mailing list, however they are expected to provide their thoughts and opinions on user level requirements and compatibility issues when RFC discussions take place.

List of Members

Charter members are (in alphabetical order):

- Arndt Bonitz
- Peter Baumann
- Stephan Krause
- Stephan Meißl
- Milan Novacek
- Martin Paces
- Fabian Schindler

Stephan Meißl is declared initial Chair of the Project Steering Committee.

Voting History

Acceptance All charter members declared their availability via e-mail to the dev mailing list.

Traceability

Requirements N/A

Tickets N/A

RFC 1: An Extensible Software Architecture for EOxServer

Author Stephan Krause

Created 2011-02-18

Last Edit 2011-07-20

Status ACCEPTED

Discussion <http://www.eoxserver.org/wiki/DiscussionRfc1>

This RFC proposes an extensible software architecture for EOxServer that is based on the following ideas:

- Separation of instance and distribution code
- Structuring of the distribution in layers
- Extensibility through a plugin system

Introduction

EOxServer development has been initiated in the course of two ESA projects that aim at providing a harmonized standard interface to access Earth Observation (EO) products, namely:

- Heterogeneous Mission Accessibility - Follow-On Open Data Access (HMA-FO ODA)
- Open-standard Online Observation Service (O3S)

The specification of a software architecture is required by these projects. From a practical point of view, EOxServer has reached a point where a common framework for a rapidly evolving project is needed.

Summarizing the requirements in a nutshell EOxServer has to integrate:

- different OGC Web Services
- different data and processing resources
- heterogeneous data and metadata formats

This leads to the conclusion that an extensible software architecture is needed. The problems to address are discussed in further detail in the *Requirements* (page 157) section.

The *proposed architecture* (page 161) is modular, extensible and flexible and structured in layers. The following separate components are identified:

- Distribution
 - *Distribution Core* (page 175)
 - *Service Layer* (page 175)
 - *Processing Layer* (page 176)
 - *Data Integration Layer* (page 176)
 - *Data Access Layer* (page 177)
- *Instances* (page 177)

In this architecture the Core shall provide the central logic for the extension mechanism while the layers shall contain interface definitions based on the extension model of the Core that can be implemented by extending modules and plugins.

Requirements

The main sources of requirements for EOxServer at the moment of writing this RFC are:

- the HMA-FO Open Data Access Software Requirements Specification (SRS)¹⁷⁵
- the O3S Software System Specification (SSS)
- the feature requests posted on the EOxServer Trac¹⁷⁶

Most of the requirements are related to the features EOxServer shall implement. There is one requirement, however, in the O3S SSS that is directly related to the software architecture; O3S_QUA_004¹⁷⁷ states:

The O3S3 shall sustain maintainability and reusability by using a modular system architecture

This shall facilitate

- isolation and removal of code defects
- integration of new functionality, such as the implementation of new interface standard versions
- extension of the system functionality according to new or modified requirements

Thus, modularity as well as integration and extension of functionality are central issues in the drafting of the EOxServer software architecture. The question remains what considerations shall govern the structuring of the software into modules, what functionality it shall implement and in what way the system shall be able to be extended.

Our approach to this question is to identify different topics of concern for the EOxServer development that shall structure the requirements analysis and give a first hint on the architectural design.

The main goal of EOxServer is to furnish an implementation of OGC¹⁷⁸ Web Services (OWS) intended for use within the Earth Observation (EO) domain. These *services* (page 157) shall provide access to different kinds of *resources* (page 158) and to *processes* (page 158) operating on these resources. The requirements cite different *backends* (page 159) that the software shall implement in order to allow access to local and remote content. Finally, we discuss where and how the software is going to be *deployed* (page 159).

Services

The following OGC Web Services shall be implemented:

Web Coverage Service (WCS)¹⁷⁹ (requirement O3S_CAP_001¹⁸⁰)

The Web Coverage Service shall be able to present Earth Observation data, e.g. optical satellite imagery, SAR data, etc. The following extensions shall be implemented:

Earth Observation Application Profile for WCS (EO-WCS) (requirement O3S_CAP_100¹⁸¹)

This application profile is intended to ease access to large collections of Earth Observation data.

Transactional WCS (WCS-T) (requirement O3S_CAP_150¹⁸²)

This extension of WCS introduces a Transaction operation that supports transfer of data to a WCS server.

Web Map Service (WMS)¹⁸³ (requirement O3S_CAP_220¹⁸⁴)

¹⁷⁵ http://wiki.services.eoportal.org/tiki-download_wiki_attachment.php?attId=957&download=y

¹⁷⁶ <http://www.eoxserver.org>

¹⁷⁷ <https://o3s.eox.at/requirements/ticket/122>

¹⁷⁸ <http://www.opengeospatial.org>

¹⁷⁹ <http://www.opengeospatial.org/standards/wcs>

¹⁸⁰ <https://o3s.eox.at/requirements/ticket/7>

¹⁸¹ <https://o3s.eox.at/requirements/ticket/8>

¹⁸² <https://o3s.eox.at/requirements/ticket/198>

¹⁸³ <http://www.opengeospatial.org/standards/wms>

¹⁸⁴ <https://o3s.eox.at/requirements/ticket/204>

This service shall be used to give to portrayals of the coverages the system presents. The following extension shall be implemented:

WMS Profile for EO Products (EO-WMS) (requirement [O3S_CAP_240](#)¹⁸⁵)

The extension allows access to portrayals of large dataset series.

Web Feature Service (WFS)¹⁸⁶ (requirement [O3S_CAP_260](#)¹⁸⁷)

This service shall be used to present vector data.

Web Processing Service (WPS)¹⁸⁸ (requirement [O3S_CAP_200](#)¹⁸⁹)

This service shall be used to make processing resources accessible online.

Processes

EOxServer shall present various processes to the public using WPS. The processes planned for implementation at the moment of writing this RFC are specific to the use cases to be handled in the course of the O3S project. The capability to publish a variety of processes on the other hand is a general requirement for EOxServer.

Being a project focussing on the EO domain EOxServer concentrates on the processing of EO coverage (raster) data. So, the considerations made for coverages regarding the variety of data and metadata *formats* (page 159) are valid for processes as well.

Resources

EOxServer shall enable public access to different kinds of geo-spatial resources in the Earth Observation domain. These are:

- Coverages
- Vector Data
- Processes

Coverages

Coverages are defined in a very abstract way. What EOxServer focusses on are coverages dealt with by the Earth Observation Application Profile for WCS (EO-WCS) which is a draft OGC Best Practice Paper as of writing this RFC. The main categories of resources defined in that paper are:

Datasets Datasets are the atomic components EO-WCS objects are composed of. They are coverages that are associated with EO Metadata. EO satellite mission scenes are a good example of Datasets. They can be accessed individually even when being part of a Stitched Mosaic or Dataset Series.

Stitched Mosaics Stitched Mosaics are made up from a collection of Datasets that share a common range type and grid. Other than Dataset Series they are not merely a container for Datasets but coverages themselves. The coverage values are generated from the contributing datasets. This process must follow some rule to decide what value to take into account in the areas where the contributing Datasets overlap. The most common rule is “latest-on-top”.

Dataset Series Dataset Series represent collections of Datasets or Stitched Mosaics. They do not impose any constraints on the contained objects, so very heterogeneous data can be included in the same series.

¹⁸⁵ <https://o3s.eox.at/requirements/ticket/210>

¹⁸⁶ <http://www.opengeospatial.org/standards/wfs>

¹⁸⁷ <https://o3s.eox.at/requirements/ticket/214>

¹⁸⁸ <http://www.opengeospatial.org/standards/wps>

¹⁸⁹ <https://o3s.eox.at/requirements/ticket/9>

A major problem for the EOxServer implementation is that raster data coverages originating from EO satellite missions are very heterogeneous. They can use a wide variety of data and metadata formats and are often associated with additional data like bitmasks, etc. that should be presented by EOxServer as well. Furthermore, the data packaging is different for every mission.

Vector Data

Support for Vector Data handling is required by O3S Use Case 2. In that use case road network data shall be generated from Pléiades satellite data using automated feature detection algorithms. The road network data shall be presented using WFS and WMS.

Backends

EOxServer shall implement various backends to access data it presents to the public via the OGC Web Services:

- Backend for local data (requirement [O3S_CAP_013](#)¹⁹⁰)
- Backends for remote data (requirements: HMA-FO SR_ODA_IF_070, [O3S_CAP_014](#)¹⁹¹)
 - using HTTP/HTTPS
 - using FTP
 - using WCS
- Backend for retrieving data from [rasdaman](#)¹⁹² (requirement [O3S_CAP_017](#)¹⁹³)

Deployment

The only requirements originating from the HMA-FO ODA and O3S projects regarding deployment concern the implementation of the O3S Use Cases. Every use case requires one or more instances of EOxServer to be deployed. The instances have different purposes and thus shall present different services and different resources.

The fact that EOxServer shall be deployed many times in different configurations (possibly on the same server) calls for a strict separation of distribution and instance data.

The ability to activate or deactivate various components of the system implies not only that the architecture must be modular but also that it must be configurable to use different combinations of modules.

Summary

The conclusion of the requirements review is that the EOxServer Architecture shall be:

- modular
- extensible
- flexible in the sense that it must be possible to select different combinations of modules to deploy and activate
- able to present resources using different OGC Web Services
- able to access data from different backends
- able to handle different data, metadata and packaging formats
- separating distribution and instance data

The development of the software architecture will be based on these considerations.

¹⁹⁰ <https://o3s.eox.at/requirements/ticket/68>

¹⁹¹ <https://o3s.eox.at/requirements/ticket/69>

¹⁹² <http://www.rasdaman.com>

¹⁹³ <https://o3s.eox.at/requirements/ticket/183>

Architecture Overview

The software architecture development for EOxServer does not start at zero. There are already considerations made in the proposal phase of the O3S project and there is the status quo of version 0.1.0. Taking into account this preparational work and the outcomes of the requirements review, the outlines of the *Proposed Architecture* (page 161) will be developed in the last subsection and the following sections.

Draft Architecture

The O3S draft Architectural Design Document (ADD/SDD) has already proposed a software architecture which is, however, outdated in certain aspects due to changes made in the requirements phase of O3S. Here is an overview of the O3S draft architecture:

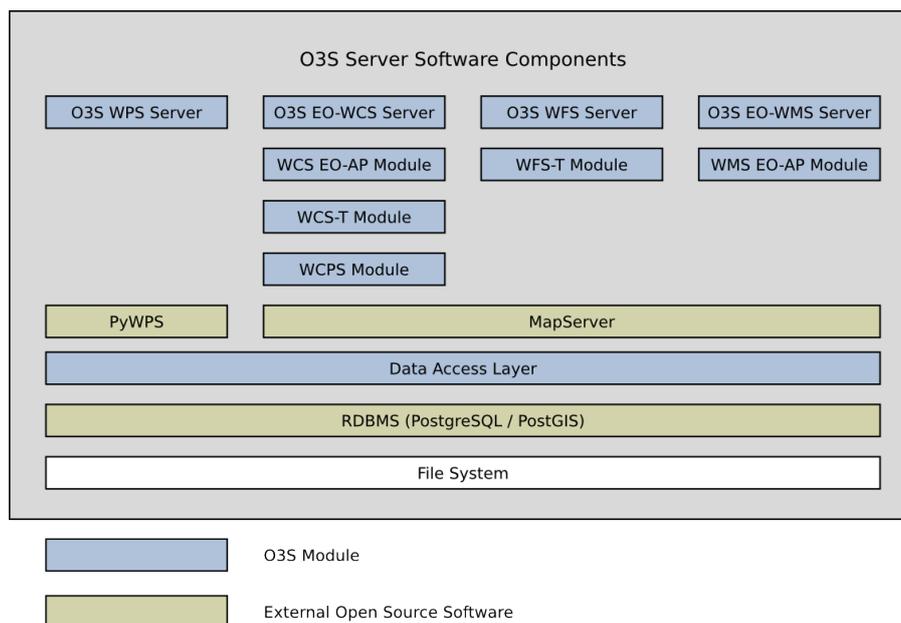


Fig. 3.1: Draft architecture from O3S Proposal

This identifies four servers and extending modules:

- WPS Server
- WCS Server
 - WCS Earth Observation Application Profile Module
 - WCS-T Module
 - WCPS Module (not included in the requirements any more)
- WFS Server
 - WFS-T Module (not included in the requirements any more)
- WMS Server
 - WMS Profile for EO Products Module

Furthermore the architecture proposes to use [PyWPS](http://pywps.wald.intevation.org/)¹⁹⁴ and [MapServer](http://www.mapserver.org)¹⁹⁵ as middleware for handling OGC Web Service requests.

¹⁹⁴ <http://pywps.wald.intevation.org/>

¹⁹⁵ <http://www.mapserver.org>

An additional integrating Data Access Layer is foreseen that shall implement storage patterns such as image pyramids and offer an API to read and write data that hides the internal details of data storage from the service and extension modules using it.

PostgreSQL¹⁹⁶ with its geo-spatial extension PostGIS¹⁹⁷ has been planned as relational database backend. Finally, the system relies on the local filesystem as its only storage backend.

During the requirements phase of O3S and the early development of EOxServer many deviations from this original design have been made necessary. Most importantly:

- Django¹⁹⁸ has been added as dependency
- GDAL¹⁹⁹ has been added as dependency
- the implementation of WCPS has been postponed
- the implementation of WFS-T has been postponed
- Django has made use of different geo-spatial database backends possible
- requirements for remote storage backends have been added

Although the basic concepts of the draft architecture remain valid, an updated version is needed for EOxServer to fulfill its requirements and evolve beyond the project horizon of O3S.

Status Quo of Release 0.1.1

As of release 0.1.1 EOxServer is an integrated Django project including a single Django application and additional modules that support OGC Web Service (OWS) request handling and data integration.

The data model is contained in the `eoxserver.server` application. So is the `ows` view, the central entrance point for OWS requests, and the administration client view as well as tools for automatic data ingestion.

Supporting modules are gathered in the `eoxserver.lib` module. These contain the core application logic for OWS request handling, coverage and metadata manipulation as well as utilities e.g. for XML processing.

EOxServer 0.1.1 includes an extension mechanism already which so far is restricted to services. The `eoxserver.lib.registry` module maintains a central registry for the concrete implementations of OWS interfaces which may be published in the `eoxserver.modules` namespace. At the moment there are implementations for WMS 1.0, 1.1 and 1.3, WCS 1.0, 1.1 and 2.0 as well as a preliminary version of EO-WCS. All these modules use MapServer MapScript for image manipulation and part of the request handling in the backend.

This approach fulfills some of the requirements summarized *above* (page 159) already, but further development of the architecture and the code is necessary to be fully compliant. Most importantly:

- extensibility and flexibility have to be enhanced
- WPS must be implemented
- WFS must be implemented
- support for remote backends is necessary

Proposed Architecture

The proposed architecture for EOxServer shall be based on the following principles:

- **Separation of Instance and Distribution:** instance applications shall be separated from EOxServer distribution code in order to facilitate deployment of multiple services on the same machine and to support flexible configurations of services

¹⁹⁶ <http://www.postgresql.org>

¹⁹⁷ <http://postgis.refrations.net>

¹⁹⁸ <http://www.djangoproject.com>

¹⁹⁹ <http://www.gdal.org>

- **Layered Architecture of the Distribution:** The software architecture shall be structured in layers and a core that contains basic common functionality; each layer builds on the capabilities of the underlying ones to fulfill its tasks
- **Extensibility:** the EOxServer distribution shall be extensible by additional modules and plugins; the distribution core shall provide functionality to enable dynamic binding to extending modules

The identification of different layers is performed based on the structuring of the system components underlying the requirements analysis.

Dependencies

The implementation of EOxServer shall use the following dependencies:

- **Python:** [Python](http://www.python.org)²⁰⁰ shall serve as the implementation language; it has been chosen because
 - it facilitates rapid development
 - the geospatial libraries used all have Python bindings
- **Django:** [Django](http://www.djangoproject.com)²⁰¹ has been selected as development framework because
 - it provides an object-relational mapper that supports various database backends
 - it supports geospatial databases and integrates vector data handling functionality in the GeoDjango extension
 - it allows for rapid web application development
- **Spatial Database Backend:** using GeoDjango, EOxServer shall support at least the [SpatialLite](http://www.gaia-gis.it/spatialite/)²⁰² and [PostGIS](http://postgis.refrains.net)²⁰³ geospatially enabled RDBMS backends.
- **MapServer:** EOxServer shall build on [MapServer](http://www.mapserver.org)²⁰⁴ MapScript in order to facilitate OGC Web Service handling
- **GDAL/OGR:** For image processing tasks and vector data manipulation the Python binding of the [GDAL/OGR](http://www.gdal.org)²⁰⁵ libraries shall be used

Concerning the software architecture, the use of Django enforces a Model-View-Controller (MVC) substructure of the distribution layers of EOxServer.

Distribution Core and Layers

The breakdown of the distribution into core and layers is as follows:

Core The Core shall contain modules for common use throughout the different components of EOxServer. This includes the global configuration data model, the implementation of the extension mechanism as well as the basic functionality for the EOxServer administration client

Service Layer This layer contains the core request handling logic as well as the implementation of services and service extensions

Processing Layer This layer contains the processing models used internally by EOxServer as well as the data model and the basic handling routines for processes to be published using WPS

Data Integration Layer This layer shall provide data models for resources as well as an abstraction layer for different data formats and data packaging formats

Data Access Layer This layer shall provide backends for local and remote data access

²⁰⁰ <http://www.python.org>

²⁰¹ <http://www.djangoproject.com>

²⁰² <http://www.gaia-gis.it/spatialite/>

²⁰³ <http://postgis.refrains.net>

²⁰⁴ <http://www.mapserver.org>

²⁰⁵ <http://www.gdal.org>

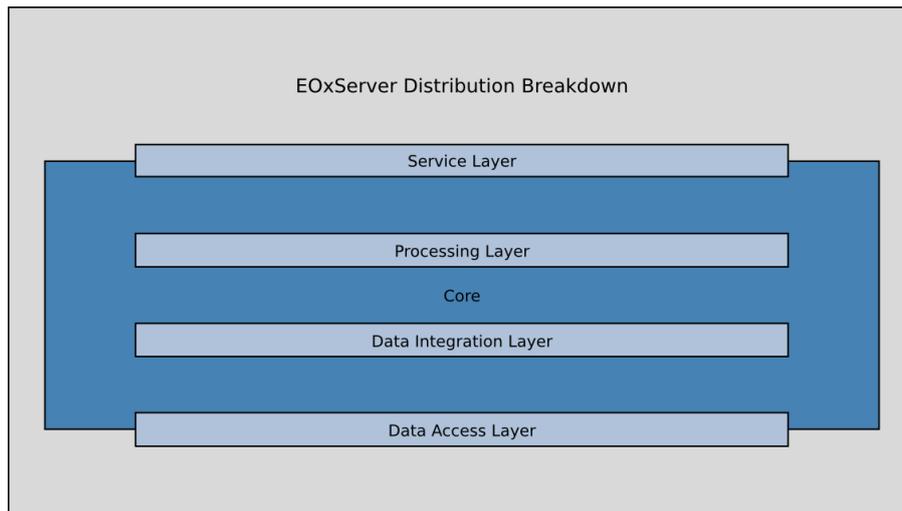


Fig. 3.2: *EOxServer Distribution Breakdown*

Each of the four layers shall be sub-structured in:

- data model
- views
 - for public access (if applicable)
 - for the administration client
- core handling logic
- interface definitions for extensions
- modules implementing the interface definitions

Structure of the Architecture Specification

The further specification of the proposed architecture is subdivided into several sections and separate RFCs. This RFC 1 contains a description of the different architectural layers and of EOxServer instances:

- *Distribution Core* (page 175)
- *Service Layer* (page 175)
- *Processing Layer* (page 176)
- *Data Integration Layer* (page 176)
- *Data Access Layer* (page 177)
- *Instances* (page 177)

The following RFCs discuss different aspects of the architecture in further detail:

RFC 2: Extension Mechanism for EOxServer

Author Stephan Krause

Created 2011-02-20

Last Edit 2011-09-15

Status ACCEPTED

Discussion <http://www.eoxserver.org/wiki/DiscussionRfc2>

This RFC proposes an extension mechanism that allows to integrate extension modules and plugins dynamically into the EOxServer distribution and instances.

Introduction

RFC 1: An Extensible Software Architecture for EOxServer (page 156) proposes an extensible architecture for EOxServer in order to ensure

- modularity
- extensibility
- flexibility

of the design. It establishes the need for an extension mechanism which acts as a sort of “glue” between different parts of the architecture and enables dynamic binding to these components.

This RFC discusses the extension mechanism in further detail and identifies the architectural principles and components needed to implement it.

The constituent components of the extension mechanism design are interface declarations, the respective implementations and a central registry that contains metadata about interfaces and implementations and enables dynamic binding to the latter ones.

Requirements

RFC 1: An Extensible Software Architecture for EOxServer (page 156) proposes an extension mechanism for EOxServer. It shall assure extensibility by additional modules and plugins and provide functionality to enable dynamic binding to extending modules.

In the layered architecture of RFC 1 the *Distribution Core* (page 175) shall be the place where the central logic that enables the dynamic extension of system functionality resides. The layers shall provide interface definitions based on the extension model of the Core that can be implemented by extending modules and plugins.

Now which extensions are needed and which requirements do they impose on the extension mechanisms? Digging deeper we have a look at the four architectural layers of EOxServer and analyze the interfaces and implementations needed by each of them.

The *Service Layer* (page 175) defines a structured approach to OGC Web Service (OWS) request handling that discerns different levels:

- services
- service versions
- service extensions
- service operations

For all of these levels interfaces are defined that are implemented by extending modules for specific OWS and their different versions and extensions.

The *Processing Layer* (page 176) defines interfaces for processes and processing chains (see *RFC 5: Processing Chains* (page 173)). Some of these are used internally and integrated into the distribution, most will be provided by plugins. While the process interface needs to be generic in order to make the implementation of many different processes possible, it must be concise enough to allow binding between processes in a processing chain. So, this must be sustained by the extension mechanism as well.

The *Data Integration Layer* (page 176) shall provide an abstraction layer for different data formats, metadata formats and data packaging formats. This shall be achieved using common interfaces for coverage data, vector data and metadata respectively.

Data and packaging formats are often not known by the system before ingesting a dataset. Thus, some kind of autodetection of formats is necessary. This is provided partly by the underlying libraries such as [GDAL²⁰⁶](#), but shall also be considered for the design of the extension mechanism: it must be possible to dynamically bind to the right data, metadata and data packaging format based on evaluations of the data. These tests should be implemented by format extensions and supported by the extension mechanism.

The *Data Access Layer* (page 177) is built around the interface definitions of backends and data sources stored by them.

In addition to modularity and extensibility RFC 1 states that the system shall be

flexible in the sense that it must be possible to select different combinations of modules to deploy and activate

Modules can be combined to build a specific application. From a user perspective it is essential to be able to activate and deactivate services, service versions and service extensions globally and/or separately for each resource or process. The same applies for other extensible parts of the system such as backends.

The O3S Use Case 2 for instance requires a server setup that consists of:

- local and WCS backends in the Data Access Layer
- a specific combination of coverage, vector data, metadata and packaging formats in the Data Integration Layer
- a feature detection process in the Processing Layer
- WPS and WFS implementations in the Service Layer

All other backends, services and processes shall be disabled.

Summarizing the requirements the extension mechanism shall support:

- extensibility by additional modules and plugins
- dynamic binding
- interface definitions for extensions
- implementations that can be enabled or disabled
 - globally
 - per resource or per process
- modules that can be configured dynamically to build an application
- autodetection of data, metadata and data packaging formats

Extension Mechanism

The basic questions for the design of the extension mechanism are:

- how to declare extensible interfaces
- how to design implementations of these interfaces
- how to advertise them
- how to bind to them

Unlike Java or C++, Python does not have a built-in mechanism to declare interfaces. A method definition always comes with an implementation. With Python 2.6 support for abstract base classes and abstract methods was added, but at the moment it is not an option to use this framework as this would break support for earlier Python versions.

So, two basic design options remain:

- using conventional Python classes and inheritance mechanisms for interfaces and implementations

²⁰⁶ <http://www.gdal.org>

- customize the interface declaration and implementation creation using Python metaclasses

Whereas the first approach is easier, the second one can provide more control and a clear differentiation between interface declaration and implementation. Both design options are discussed in further detail in the *Interfaces and Implementations* (page 166) section below.

The second major topic is how to find and bind to implementations of an interface if not all implementations are known to the system a priori, as is the case with plugins. Some “glue” is needed that holds the system together and allows for dynamic binding. In the case of EOxServer this is implemented by a central registry that keeps track of implementations by automatically scanning Python modules in certain directories that are supposed to contain EOxServer extending modules or plugins. For more details on the basics of *Registry* (page 168) see below.

In most cases an instance of EOxServer will not need all the functionality provided by the distribution or plugins installed on the system. Dynamic binding allows for enabling and disabling certain services, processes, formats, backends and plugins in an interactive way using the administration client. In order to assure this required functionality a configuration data model is needed that allows to store information about what parts of the system are activated and what resources they may operate on. See the section *Data Model* (page 168) for further details.

Implementations of interfaces are not isolated objects. They depend on libraries, functionality provided by the EOxServer core and layers and, last but not least, on other interface implementations. In order to assure that the dynamically configurable system is in a consistent state, the interdependencies between implementations need to be properly advertised and stored in the configuration data model.

After this short overview, we will go more in depth in the following sections.

Interfaces and Implementations

As already discussed before there are two design options for interfaces and implementations:

- interfaces and implementations as conventional Python classes that are linked through inheritance
- interfaces as special Python classes that are linked to implementations by a custom mechanism.

Whereas the first approach is straightforward and easy to implement and handle it has also some serious drawbacks. Most importantly it does not allow for a clear separation between interface declaration and implementation. A method declared in the interface always must contain an implementation, and an implementation may change the signature of the methods it implements in any possible way.

What’s more, as the implementation inherits (mostly generic) method code from the interface there is no way to validate if it actually defines concrete methods to override the “abstract” ones the interface class provides.

So, there are also good reasons for the second approach although it is more challenging for developers. The approach proposed here allows to customize class generation and inheritance enabling validation at “compile time” (i.e. when classes are created) and runtime (i.e. when instance methods are invoked) as well as separation of interface definition from implementation.

How can this be achieved? The proposed mechanism relies on an interface base class called `Interface` that concrete interface declarations can derive from, implementing code contained in a conventional Python class and a method called `implement()` that generates a special implementation class from the interface declaration and the class containing the implementing code.

Interface Declaration

It has already been said that interface declarations shall derive from a common base class called `Interface`. But that is not the end of the story - one big question remains: how to declare actual methods without implementation? The proposed approach is not to declare methods as such at all, but use classes representing them instead.

For this end three classes are to be defined alongside the `Interface` base class.

- instances of the `Constant` class represent constants defined by the interface
- instances of the `Method` class represent methods

- instances of the `Arg` class represent method arguments; subclasses of `Arg` allow for type validation, e.g. instances of `IntArg` represent integer arguments

Let's have a look at a quick example:

```
from eoxserver.core.interfaces import Interface, Method, Arg

class ServiceInterface(Interface):
    handle = Method(
        Arg("req")
    )
```

Note: Code examples in this RFC are merely informational. The actual implementation may deviate a little bit from them. A reference documentation will be prepared for the definitive extension mechanism.

This snippet of Python code represents a simple and complete interface declaration. The `ServiceInterface` class will be used in further examples as well. It shows a method definition that declares the following: the method `handle` shall take one argument of arbitrary type named `req` that stands for an OWS request.

As you can see the declaration is a class variable containing an instance of the `Method` class. It is not a method (it does not even have to be callable). It serves two purposes:

- documentation of the interface
- validation of the implementation

Thinking of these two goals, the writer of the code could have been more rigorous and declare an argument like this:

```
handle = Method(
    ObjectArg("req", arg_class=OWSRequest)
)
```

That way it is documented what kind of argument is expected. When defining the implementation it is enforced that it have a method `handle` which takes exactly one argument besides `self`, otherwise an exception will be raised. When invoking an interface of the implementation it can be validated that the argument is of the right type. More on this later under *Validation of Implementations* (page 168). Now let's have a look at implementations.

Implementations

The proposed design of interface implementation intends to hide all the complexity of this process from the developers of implementations. They just have to write an implementing class which is a normal new-style Python class, and wrap it with the `implement()` method of the interface, such as in the following example:

```
from eoxserver.services.owscommon import ServiceInterface

class WxSService(object):
    def handle(self, req):
        # ...

        return response

WxSServiceImplementation = ServiceInterface.implement(WxSService)
```

The call to `implement()` ensures validation of the interface and produces an implementation class that inherits all the code of the implementing class and contains information about the interface. This is only the basic functionality of the interface implementation process: more is to be revealed in the following sections.

Validation of Implementations

The validation of implementations is performed in two ways:

- at class creation time
- at instance method invocation time

Validation at class creation time checks:

- if all methods declared by the interface are implemented
- if the method arguments of the interface and implementation match

Class creation time validation is performed unconditionally.

Instance method invocation time (“runtime”) validation is optional. It can be triggered by the `runtime_validation_level` setting. There are three possible values for this option:

- `trust`: no runtime validation
- `warn`: argument types are checked against interface declaration; in case of mismatch a warning is written to the log file
- `fail`: argument types are checked against interface declaration; in case of mismatch an exception is raised

The `runtime_validation_level` option can be set

- globally (in configuration file)
- per interface
- per implementation

where stricter settings override weaker ones.

Note: The `warn` and `fail` levels are intended for use throughout the development process. In a production setting `trust` should be used.

Registry

The Registry is the core component for managing the extension mechanism of EOxServer. It is the central entry point for:

- automated detection of registered interfaces and implementations
- dynamical binding to the implementations
- configuration of components and relations between them

Its functionality shall be discussed in further detail in the following subsections:

- *Data Model* (page 168)
- *Detection* (page 170)
- *Binding* (page 170)

Data Model

The data model for the Extension Mechanism including dynamic binding is implemented primarily by the *Registry* (page 168); for persistent information it relies on the configuration files and the database.

As you’d expect, the Registry data model relies on interfaces and implementations. However, not all of them are registered, but only descendants of `RegisteredInterface` and their respective implementations.

RegisteredInterface extends the configuration model for interfaces with information relevant to the registration and dynamic binding processes. This is an example for a valid configuration:

```

from eoxserver.core.registry import RegisteredInterface

class SomeInterface(RegisteredInterface):

    REGISTRY_CONF = {
        "name": "Some Interface",
        "intf_id": "somemodule.SomeInterface",
        "binding_method": "direct"
    }

```

The most important parts are the interface ID `intf_id` and the `binding_method` settings which will be used by the registry to find implementations of the interface and to determine how to bind to them. For more information see the *Binding* (page 170) section below.

The registry model is accompanied by a database model that allows to store persistently which parts of the system (services, plugins, etc.) are enabled and which resources they have access to.

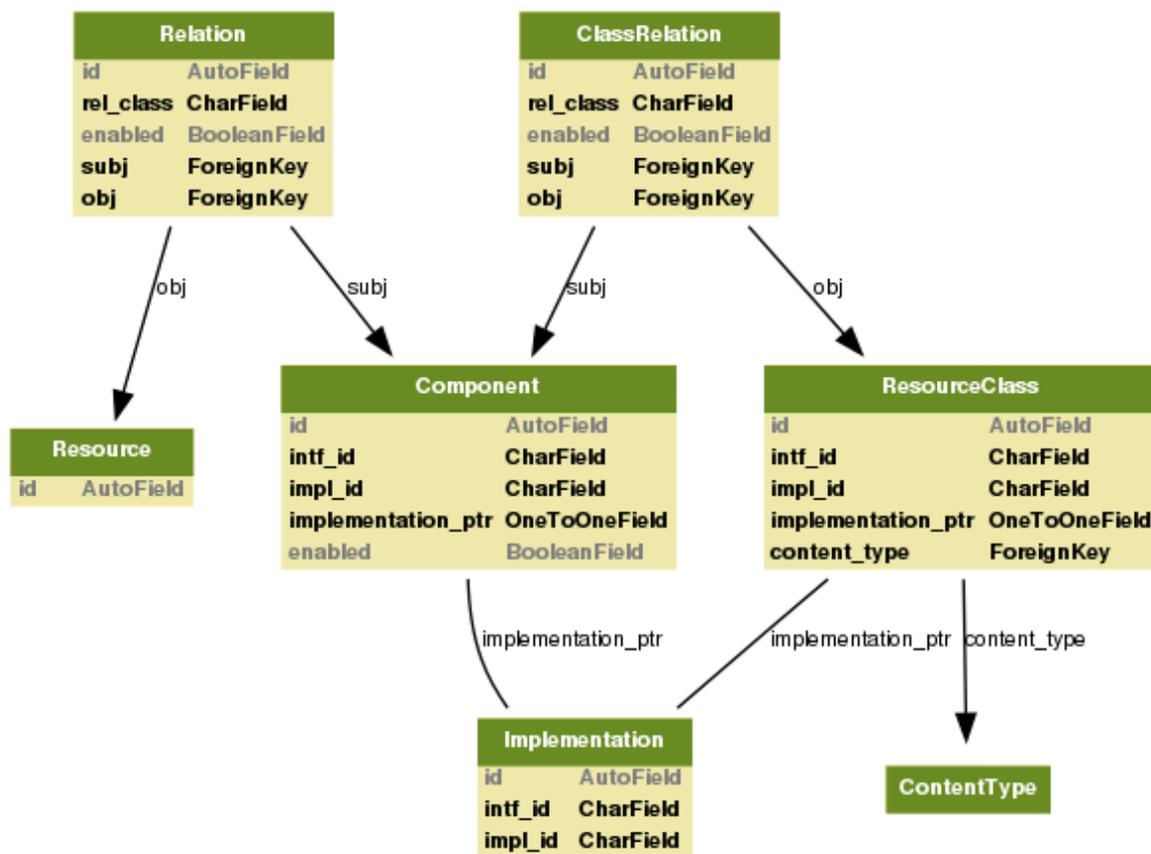


Fig. 3.3: Database Model for the Registry

For every registered implementation an `Implementation` instance and database record are created. Implementations are subdivided into components and resource classes, each with their respective model deriving from `Implementation`. Components stand for the active parts of the system like Service Handlers. They can be enabled or disabled. Resource classes relate to a specific resource wrapper which in turn relate to some specific model derived from `Resource`.

Furthermore, there is the possibility to create, enable and disable relations between components and specific resource instances or resource classes. These relations are used to determine whether a given component has

access to a given resource or resource class. They allow to configure the behaviour e.g. of certain services and protect parts of an EOxServer instance from unwanted access.

As the number of registered components is quite large and as there are many interdependencies between them and to resource classes specific Component Managers shall be introduced in order to:

- group them to larger entities which are more easy to handle
- validate the configuration with respect to these interdependencies
- facilitate relation management
- automatically create the needed relations

These managers shall implement the common `ComponentManagerInterface`.

Detection

The first step in the dynamic binding process provided by the registry is the detection of interfaces and implementations to be registered. For this end the registry loads the modules defined in the configuration files and searches them for descendants of `RegisteredInterface` and their implementations. The metadata of the detected interfaces and implementations (the contents of ‘REGISTRY_CONF’) is ingested into the registry. This metadata is used for binding to the implementations, see the following subsection *Binding* (page 170) for details.

The main EOxServer configuration file `eoxserver.conf` contains options for determining which modules shall be scanned during the detection phase. The user can define single modules and whole directories to be searched for modules there.

Binding

The registry provides four binding methods:

- direct binding
- KVP binding
- test binding
- factory binding

Direct binding means that the implementation to bind to is directly referenced by the caller using its implementation ID:

```
from eoxserver.core.system import System

impl = System.getRegistry().bind(
    "somemodule.SomeImplementation"
)
```

Direct binding is available for every implementation. You can also set the `binding_method` in the `REGISTRY_CONF` of an interface to `direct`, meaning that its implementations are reachable only by this method. This is used e.g. for component managers and factories.

The easiest method for parametrized dynamic binding is key-value-pair matching, or KVP binding. It is used if an interface defines `kvp` as its `binding_method`. The interface must then define in its `REGISTRY_CONF` one or more `registry_keys`, the implementations in turn must define `registry_values` for these keys. When looking up a matching implementation, the parameters given with the request are matched against these key-value-pairs. Finally, the registry returns an instance of the matching implementation:

```
from eoxserver.core.system import System

def dispatch(service_name, req):

    service = System.getRegistry().findAndBind(
```

```

    intf_id = "services.interfaces.ServiceHandler",
    params = {
        "services.interfaces.service": service_name.lower()
    }
)

response = service.handle(req)

return response

```

This binding method is used e.g. for binding to service, version and operation handlers for OGC Web Services based on the parameters sent with the request.

A more flexible way to determine which implementation to bind to is the test binding method ("binding_method": "testing"). In this case, the interface must be derived from TestingInterface. The implementation must provide a test() method which will be invoked by the registry in order to determine if it is suitable for a given set of parameters. This can be used e.g. to determine which format handler to use for a given dataset:

```

from eoxserver.core.system import System

format = System.getRegistry().findAndBind(
    intf_id = "resources.coverages.formats.FormatInterface",
    params = {
        "filename": filename
    }
)

...

```

The fourth binding method is factory binding ("binding_method": "factory"). In this case the registry invokes a factory that returns an instance of the desired implementation. Factories must be implementations of a descendant of FactoryInterface. Implementations and factories are linked together only at runtime, based on the metadata collected during the detection phase. This binding method is used e.g. for binding to instances of a resource wrapper:

```

from eoxserver.core.system import System

resource = System.getRegistry().getFromFactory(
    factory_id = "resources.coverages.wrappers.SomeResourceFactory",
    obj_id = "some_resource_id"
)

```

In order to access other functions of the factory you can bind to it directly. For retrieving all resources that are accessible through a factory you would use code like this:

```

from eoxserver.core.system import System

resource_factory = System.getRegistry().bind(
    "resources.coverages.wrappers.SomeResourceFactory"
)

resources = resource_factory.find()

```

Voting History

Motion To accept RFC 2

Voting Start 2011-07-25

Voting End 2011-09-15

Result +6 for ACCEPTED

Traceability

Requirements N/A

Tickets N/A

RFC 3: OGC Service Extensions

Author Stephan Krause

Created 2011-02-20

Last Edit 2011-02-20

Status IN PREPARATION

Discussion <http://www.eoxserver.org/wiki/DiscussionRfc3>

<short description of the RFC>

Introduction

<Mandatory. Overview of motivation, addressed problems and proposed solution>

Voting History

N/A

Traceability

Requirements N/A

Tickets N/A

RFC 4: Data Packaging

Author Stephan Krause

Created 2011-02-20

Last Edit 2011-02-25

Status IN PREPARATION

Discussion <http://www.eoxserver.org/wiki/DiscussionRfc4>

<short description of the RFC>

Introduction

<Mandatory. Overview of motivation, addressed problems and proposed solution>

Voting History

N/A

Traceability

Requirements N/A

Tickets N/A

RFC 5: Processing Chains

Author Stephan Krause

Created 2011-02-23

Last Edit 2011-03-01

Status IN PREPARATION

Discussion <http://www.eoxserver.org/wiki/DiscussionRfc5>

<short description of the RFC>

Introduction

<Mandatory. Overview of motivation, addressed problems and proposed solution>

Voting History

N/A

Traceability

Requirements N/A

Tickets N/A

RFC 6: Directory Structure

Author Stephan Krause

Created 2011-02-24

Last Edit 2011-09-15

Status ACCEPTED

Discussion <http://www.eoxserver.org/wiki/DiscussionRfc6>

This RFC proposes a directory structure for the EOxServer distribution as well as EOxServer instances.

Introduction

RFC 1: An Extensible Software Architecture for EOxServer (page 156) introduces a layered architecture for EOxServer as well as a separation of EOxServer distribution and instances. This RFC lays out a directory structure that is in line with this architecture.

Directory Structure

Distribution

- `core`: contains the modules of the Core
 - `util`: contains utility modules to be used throughout the project
- `services`: contains the modules of the Service Layer
 - `ows`: contains implementations of OGC Web Services
- `processing`: contains the modules of the Processing Layer
 - `processes`: contains processes
- `resources`: contains the modules of the Data Integration Layer
 - `coverages`: contains the modules related to coverage resources
 - * `formats`: contains the modules related to coverage formats
 - `vector`: contains the modules related to vector data
 - * `formats`: contains the modules related to vector data formats
- `contrib`: contains (links to) third party modules
- `conf`: contains the default configuration

Instance

The instance directory contains the three Django project modules:

- `settings.py`
- `manage.py`
- `urls.py`

And the following subdirectories

- `conf`: configuration files
 - `eoxserver.conf`: the central EOxServer configuration
 - `template.map`: template MapFile for OWS requests
- `data`: database files
 - `config.sqlite`: SQLite database

Voting History

Motion To accept RFC 6

Voting Start 2011-07-25

Voting End 2011-09-15

Result +6 for ACCEPTED

Traceability

Requirements N/A

Tickets N/A

Distribution Core

The Core shall act as a “glue” for EOxServer that links the different parts of the software together and provides functionality used throughout the EOxServer project.

It defines the core of the configuration data model which is extended by the layers and implementing modules. The configuration is partly stored in the database and partly in files. Both parts need to be easily modifiable and extensible.

Therefore the Core also includes an administration client that can be used by system operators to edit the part of the configuration stored in the database. The basic functionality of the administrator, the entry view and its extension mechanisms shall be part of the Core.

The Core includes modules for common use, for instance utilities for the handling of spatio-temporal metadata as well as for decoding and encoding of XML documents.

Most importantly, the Core contains the central logic that enables the dynamic extension of system functionality. The layers shall provide interface definitions based on the extension model of the Core that can be implemented by extending modules and plugins. For more details see *RFC 2: Extension Mechanism for EOxServer* (page 163).

Service Layer

The Service Layer contains the OWS request handling logic as well as the implementation of services and service extensions.

It defines a configuration **data model** for OGC Web Services and for their metadata. The model includes:

- service metadata to be published in the GetCapabilities response
- options to enable or disable a specific service or service extension for a given data source
- options to configure the services themselves, e.g. enabling or disabling certain non-mandatory features

The Service Layer provides **views** for public access, namely the central entrance point for OWS requests. It also contains views for the administration client that allow to configure services and service metadata.

The **core handling logic** for OGC Web Services is part of the Service Layer. It implements the behaviour defined by OWS Common and defines a structured approach to request handling that discerns different levels:

- services
- service versions
- service extensions
- service operations

The way services and service extensions interact is described in further detail in *RFC 3: OGC Service Extensions* (page 172).

The Service Layer defines request handler **interfaces** for each of these levels that are **implemented** by modules for:

- WPS
- WCS
 - EO-WCS
 - WCS-T
- WMS
 - EO-WMS
- WFS

Processing Layer

The Processing Layer contains the processing models used internally by EOxServer as well as the data model and the basic handling routines for processes to be published using WPS.

In its **data model** it defines the configuration options and metadata for processes. The model shall also support processing chains as described in further detail in *RFC 5: Processing Chains* (page 173). The Processing Layer publishes administration client **views** to support the configuration of processes and processing chains.

The Processing Layer defines **interfaces** for processes. It also contains implementations of the processes used internally by EOxServer; these include:

- coverage tiling
- coverage mosaicking

Further processes as required e.g. by the O3S Use Cases will be added as plugins based on the data model and interface definitions of the Processing Layer.

Data Integration Layer

The Data Integration Layer shall provide data models for resources as well as an abstraction layer for different data formats and data packaging formats.

Data packaging formats are explained in greater detail in *RFC 4: Data Packaging* (page 172). Roughly speaking, they represent the way data and metadata for an EO product or derived product are packaged. They shall abstract from the actual substructure of the packaging format in directories and files so these resources can be handled transparently by EOxServer.

Its **data model** shall include items common to all types of data as well as individual models for:

- coverages
- vector data
- metadata

Just as the other layers the Data Integration Layer shall publish administration client **views** that support adding, modifying and removal of resources and their respective metadata.

The **interface definitions** of the Data Integration Layer shall provide an abstraction layer for:

- various data formats
- various metadata formats
- various data packaging formats

The modules **implementing** these interfaces shall support:

- coverage data formats supported by:
 - GDAL²⁰⁷
 - NEST²⁰⁸ (optional)
- vector data formats supported by OGR²⁰⁹
- metadata formats:
 - EO-GML
 - DIMAP (optional)
 - INSPIRE (optional)
 - GSC-DA (optional)

²⁰⁷ <http://www.gdal.org>

²⁰⁸ <http://www.array.ca/nest>

²⁰⁹ <http://www.gdal.org/ogr/>

- data packaging formats:
 - directories
 - ZIP archives
 - TAR archives
 - compressed file formats:
 - * ZIP
 - * GZIP
 - * BZ2

Data Access Layer

The Data Access Layer shall provide transparent access to local and remote data using different backends. It constitutes an abstraction layer for data sources.

Its **data model** therefore provides configuration options for the backends. It contains **views** for the administration client to configure different data sources.

The Data Access Layer is built around the **interface definitions** of backends and data sources stored by them. The following backends need to be **implemented**:

- local backends:
 - file system
 - [rasdaman](http://www.rasdaman.com)²¹⁰ backend
- remote backends:
 - using HTTP/HTTPS
 - using FTP
 - using WCS

Instances

EOxServer instances are Django projects that import different EOxServer modules as Django applications.

Like every Django project they contain a settings file that governs the Django configuration and in addition the most basic parts of EOxServer configuration. Specifically:

- the connection details for the database containing the EOxServer configuration is defined in the settings file
- the Django `INSTALLED_APPS` setting must be used to define the parts of the EOxServer data model that shall be loaded
- some EOxServer configuration settings that are needed in the startup phase will be appended to the Django settings file

Apart from the settings, every Django project has an “urlconf” that defines which URLs shall point to the different views of the project. For using the full EOxServer functionality there have to be URLs pointing to the Service Layer OWS entrance point and the administration client entrance point defined by the EOxServer core.

Furthermore the instance contains the Django configuration files whose content is defined by the configuration data model of the Core and the layers.

Optionally, the instance directory may include subdirectories for the data (if stored locally) and the database (if using the file-based SpatialLite spatial database backend).

²¹⁰ <http://www.rasdaman.com>

Finally, in a production setting, it shall contain the modules needed to deploy the instance. The favourite deployment method is WSGI (see [PEP 333](#)²¹¹). These must be configured as well to include the path to the instance.

The Django project may or may not contain applications itself, which may or may not use EOxServer functionality. Writing an own application is not necessary to use EOxServer, though; placing links to EOxServer views in the urlconf is sufficient.

Voting History

Moved to ACCEPTED by unanimous consent without a formal vote on July 20th, 2011.

Traceability

Requirements	HMA-FO	SR_ODA_IF_070,	O3S_CAP_001 ²¹² ,	O3S_CAP_013 ²¹³ ,
	O3S_CAP_014 ²¹⁴ ,	O3S_CAP_017 ²¹⁵ ,	O3S_CAP_100 ²¹⁶ ,	O3S_CAP_150 ²¹⁷ ,
	O3S_CAP_200 ²¹⁸ ,	O3S_CAP_220 ²¹⁹ ,	O3S_CAP_240 ²²⁰ ,	O3S_CAP_260 ²²¹ ,
	O3S_QUA_004 ²²²			

Tickets N/A

RFC 7: Release Guidelines

Author Stephan Meißl

Created 2011-05-04

Last Edit \$Date\$

Status ACCEPTED

Discussion <http://eoxserver.org/wiki/DiscussionRfc7>

Id \$Id\$

Overview

This RFC documents the EOxServer release manager role and the phases of EOxServer's release process.

(Credit: Inspired by the MapServer release guidelines at: <http://mapserver.org/development/rfc/ms-rfc-34.html>)

The EOxServer Release Manager Role

For every release of EOxServer, the PSC elects a release manager via motion and vote on the dev mailing list.

The overall role of the release manager is to coordinate the efforts of the developers, testers, documentation, and other contributors to lead to a release of the best possible quality within the scheduled timeframe.

The PSC delegates to the release manager the responsibility and authority to make certain final decisions for a release, including:

²¹¹ <https://www.python.org/dev/peps/pep-0333>

²¹² <https://o3s.eox.at/requirements/ticket/7>

²¹³ <https://o3s.eox.at/requirements/ticket/68>

²¹⁴ <https://o3s.eox.at/requirements/ticket/69>

²¹⁵ <https://o3s.eox.at/requirements/ticket/183>

²¹⁶ <https://o3s.eox.at/requirements/ticket/8>

²¹⁷ <https://o3s.eox.at/requirements/ticket/198>

²¹⁸ <https://o3s.eox.at/requirements/ticket/9>

²¹⁹ <https://o3s.eox.at/requirements/ticket/204>

²²⁰ <https://o3s.eox.at/requirements/ticket/210>

²²¹ <https://o3s.eox.at/requirements/ticket/214>

²²² <https://o3s.eox.at/requirements/ticket/122>

- Approving or not the release of each beta, release candidate, and final release
- Approving or rejecting non-trivial bug fixes or changes after the feature freeze
- Maintaining the release schedule (timeline) and making changes as required

When in doubt or for tough decisions (e.g. pushing the release date by several weeks) the release manager is free to ask the PSC to vote in support of some decisions, but this is not a requirement for the areas of responsibility above.

The release manager's role also includes the following tasks:

- Setup and maintain a release plan wiki page for each release
- Coordinate with the developers team
- Coordinate with the QA/testers team
- Coordinate with the docs/website team
- Keep track of progress via Trac milestones and ensure tickets are properly targeted
- Organize IRC meetings if needed (including agenda and minutes)
- Tag source code in SVN for each beta, RC, and release
- Branch source code in SVN after the final release (trunk becomes the next dev version)
- Update version in files for each beta/RC/release
- Package source code distribution for each beta/RC/release
- Update appropriate website/download page for each beta/RC/release
- Make announcements on users and announce mailing lists for each release
- Produce and coordinate bugfix releases as needed after the final release

Any of the above tasks can be delegated but they still remain the ultimate responsibility of the release manager.

The EOxServer Release Process

The normal development process of a EOxServer release consists of various phases.

- Development phase

The development phase usually lasts several months. New features are proposed via RFCs and voted by the EOxServer PSC.

- RFC freeze date

For each release there is a certain date by which all new feature proposals (RFCs) must have been submitted for review. After this date no features will be accepted anymore for this particular release.

- Feature freeze date / Beta releases

By this date all features must have been completed and all code has to be integrated. Only non-invasive changes, user interface work and bug fixes are done now. There are usually 3 to 4 betas and a couple of release candidates before the final release.

- Release Candidate

Ideally, the last beta that is bug free. No changes to the code. Should not require any migration steps apart from the ones required in the betas. If any problems are found and fixed, a new release candidate is issued.

- Final release / Expected release date

Normally the last release candidate that is issued without any show-stopper bugs.

- Bug fix releases

No software is perfect. Once a sufficient large or critical number of bugs have been found for a certain release, the release manager releases a new bug fix release a.k.a. third-dot release.

EOxServer Version Numbering

EOxServer's version numbering scheme is very similar to Linux's. For example, a EOxServer version number of 1.2.5 can be decoded as such:

- 1: Major version number.

The major version number usually changes when significant new features are added or when major architectural changes or backwards incompatibilities are introduced.

- 2: Minor version number.

Increments in minor version number almost always relate to additions in functionality.

- 5: Revision number.

Revisions are bug fixes only. No new functionality is provided in revisions.

Voting History

Motion Adopted on 2011-11-16 with +1 from Stephan Meißl, Milan Novacek, Martin Paces

Traceability

Requirements N/A

Tickets N/A

RFC 8: SVN Commit Management

Author Stephan Meißl

Created 2011-05-04

Last Edit 2011-05-18

Status ACCEPTED

Discussion <http://www.eoxserver.org/wiki/DiscussionRfc8>

Overview

This RFC documents the EOxServer guidelines for SVN commit access and specifies some guidelines for SVN committers.

(Credit: Inspired by the MapServer SVN commit management guidelines at: <http://mapserver.org/development/rfc/ms-rfc-7.1.html>)

Election to SVN Commit Access

Permission for SVN commit access shall be provided to new developers only if accepted by the EOxServer Project Steering Committee (PSC). A proposal should be written to the PSC for new committers and voted on normally. It is not necessary to write an RFC document for these votes. An e-mail to the dev mailing list is sufficient.

Removal of SVN commit access should be handled by the same procedure.

The new committer should have demonstrated commitment to EOxServer and knowledge of the EOxServer source code and processes to the committee's satisfaction, usually by reporting tickets, submitting patches, and/or actively participating in the various EOxServer forums.

The new committer should also be prepared to support any new feature or changes that he/she commits to the EOxServer source tree in future releases, or to find someone to which to delegate responsibility for them if he/she stops being available to support the portions of code that he/she is responsible for.

All committers should also be a member of the dev mailing list so they can stay informed on policies, technical developments, and release preparation.

Committer Tracking

A list of all project committers will be kept in the main eoxserver directory (called COMMITTERS) listing for each SVN committer:

- Userid: the id that will appear in the SVN logs for this person.
- Full name: the users actual name.
- Email address: A current email address at which the committer can be reached. It may be altered in normal ways to make it harder to auto-harvest.
- A brief indication of areas of responsibility.

SVN Administrator

One member of the PSC will be appointed the SVN Administrator. That person is responsible for giving SVN commit access to folks, updating the COMMITTERS file, and other SVN related management. Initially Stephan Meißl will be the SVN Administrator.

SVN Commit Practices

The following are considered good SVN commit practices for the EOxServer project.

- Use meaningful descriptions for SVN commit log entries.
- Add a ticket reference like “(#1232)” at the end of SVN commit log entries when committing changes related to a ticket in Trac.
- Include changeset revision numbers like “r7622” in tickets when discussing relevant changes to the code-base.
- Changes should not be committed in stable branches without a corresponding ticket. Any change worth pushing into a stable version is worth a Trac ticket.
- Never commit new features to a stable branch: only critical fixes. New features can only go in the main development trunk.
- Only ticket defects should be committed to the code during pre-release code freeze.
- Significant changes to the main development version should be discussed on the dev mailing list before making them, and larger changes will require an RFC approved by the PSC.
- Do not create new branches without the approval of the PSC. A Release manager designated under [RFC 7: Release Guidelines](#) (page 178) is automatically granted permission to create a branch, as defined by their role described in [RFC 7: Release Guidelines](#) (page 178).
- All source code in SVN should be in Unix text format as opposed to DOS text mode.
- When committing new features or significant changes to existing source code, the committer should take reasonable measures to insure that the source code continues to work.
- Include the standard EOxServer header in every new file and set the following SVN properties:

- svn propset svn:keywords 'Author Date Id Rev URL' <new_file>
- svn propset svn:eol-style native <new_file>

Legal

Committers are the front line gatekeepers to keep the code base clear of improperly contributed code. It is important to the EOxServer users and developers to avoid contributing any code to the project without it being clearly licensed under the project license.

Generally speaking the key issues are that those providing code to be included in the repository understand that the code will be released under the EOxServer License, and that the person providing the code has the right to contribute the code. For the committer themselves understanding about the license is hopefully clear. For other contributors, the committer should verify the understanding unless the committer is very comfortable that the contributor understands the license (for instance frequent contributors).

If the contribution was developed on behalf of an employer (on work time, as part of a work project, etc) then it is important that an appropriate representative of the employer understand that the code will be contributed under the EOxServer License. The arrangement should be cleared with an authorized supervisor/manager, etc.

The code should be developed by the contributor, or the code should be from a source which can be rightfully contributed such as from the public domain, or from an open source project under a compatible license.

All unusual situations need to be discussed and/or documented.

Committers should adhere to the following guidelines, and may be personally legally liable for improperly contributing code to the source repository:

- Make sure the contributor (and possibly employer) is aware of the contribution terms.
- Code coming from a source other than the contributor (such as adapted from another project) should be clearly marked as to the original source, copyright holders, license terms and so forth. This information can be in the file headers, but should also be added to the project licensing file if not exactly matching normal project licensing (eoxserver/COPYING and eoxserver/README).
- Existing copyright headers and license text should never be stripped from a file. If a copyright holder wishes to give up copyright they must do so in writing to the project before copyright messages are removed. If license terms are changed it has to be by agreement (written in email is ok) of the copyright holders.
- When substantial contributions are added to a file (such as substantial patches) the author/contributor should be added to the list of copyright holders for the file.
- If there is uncertainty about whether a change is proper to contribute to the code base, please seek more information from the PSC.

Voting History

Motion Adopted on 2011-05-17 with +1 from Arndt Bonitz, Stephan Krause, Stephan Meißl, Milan Novacek, Martin Paces, Fabian Schindler

Traceability

Requirements N/A

Tickets N/A

RFC 9: SOAP Binding of WCS GetCoverage Response

Author Milan Novacek

Created 2011-05-17

Last Edit 2011-05-30

Status ACCEPTED

Discussion <http://www.eoxserver.org/wiki/DiscussionRfc9>

Introduction

The current/draft OGC specifications for the SOAP binding for a WCS GetCoverage Response are inconsistent with the SOAP spec if the GetCoverage response includes a binary file. This RFC proposes an update to OGC 09-149r1 to resolve the inconsistencies: Requirements 5 and 6 should be changed to use SOAP MTOM where the entire coverage response comprises the attachment. This coverage attachment is referred to from within a new element ‘Coverage’ which is also defined as part of this RFC.

Problem Description

In OGC 09-149r1, Requirement 5 mandates that a GetCoverage SOAP response shall be encoded as “SOAP with Attachments” as defined in [W3C Note 11], but using SOAP 1.2 rather than SOAP 1.1. Requirement 6 says, rather imprecisely, that in a GetCoverage response, the SOAP Envelope shall contain one Body element which contains the Coverage as its single element.

For binary attachments to SOAP 1.2 messages, W3C recommends the usage of MTOM instead of SwA (see [1] and [2]). According to the guidance in [1], the SOAP 1.2 MTOM standard requires the use of the `xop:Include` element to refer to binary attachments. The difficulty arises because the “`gml:rangeSet`” element, which according to OGC 09-110r is mandated for a GetCoverage response, does not have a provision for using the `xop:Include` element to refer to an attached file. For this reason one cannot include a reference to an MTOM SOAP attachment in the GetCoverage response.

Proposed Changes to OGC 09-149r1

To resolve the problem, we propose to update two requirements of OGC 09-149r1 as follows:

Requirement 5: A GetCoverage SOAP response **shall** be encoded according to the W3C SOAP 1.2 standard [<http://www.w3.org/TR/soap12-part1/>] using MTOM [<http://www.w3.org/TR/soap12-mtom/>].

Requirement 6: In a GetCoverage response, the SOAP Body **shall** contain one element, “Coverage” of type “SoapCoverageType”, defined in the namespace <http://www.opengis.net/wcs/2.0>, according to the schema definition in <http://www.opengis.net/wcs/2.0/wcsSoapCoverage.xsd>.

Schema Location

For discussion purposes of this RFC, the proposed schema *wcsSoapCoverage.xsd* is available in the sandbox [3]. For convenience, *wcsCommon.xsd* in the same directory has been modified to include *wcsSoapCoverage.xsd*.

References

- [1] <http://www.w3.org/TR/soap12-part0/>
- [2] <http://www.w3.org/TR/soap12-mtom/>
- [3] sandbox/sandbox_wcs_soap_proxy/schemas/wcs/2.0/wcsSoapCoverage.xsd

Voting History

Motion Adopted on 2011-05-30 with +1 from Martin Paces, Stephan Meißl, Milan Novacek, Stephan Krause, and +0 from Arndt Bonitz

Traceability

Requirements “N/A”

Tickets “N/A”

RFC 10: SOAP Proxy

Author Milan Novacek

Created 2011-05-18

Last Edit 2011-05-30

Status ACCEPTED

Discussion <http://www.eoxserver.org/wiki/DiscussionRfc10>

Introduction

This RFC proposes the design and implementation of the module *soap_proxy*. Initially *soap_proxy* is for use with WCS services. The intent of *soap_proxy* is to provide a soap processing front end for those WCS services which do not natively accept soap messages. *Soap_proxy* extracts the xml of a request from an incoming SOAP message and invokes mapserver or eoxserver in POST mode with the extracted xml. It then accepts the response from mapserver or eoxserver and repackages it in a SOAP reply.

Description

Soap-proxy should implement OGC 09-149 *Web Coverage Service 2.0 Interface Standard - XML/SOAP Protocol Binding Extension*. See RFC-9 for a proposal to address certain problems with the current revision of this standard (which is OGC 09-149r1).

Initially it is planned that *soap_proxy* supports WCS 2.0. WCS 1.1 is a low priority. The possibility should be investigated to generalize *soap_proxy* to enable support of other protocols such as WPS.

Soap_proxy is implemented as a Web Service using the Axis2/C framework [AXIS], plugged into a standard Apache HTTP server via its `mod_axis2` module.

Governance

Source Code Location

The *soap_proxy* code will be located in the subdirectory ‘*soap_proxy*’ at the main level of the eoxserver repository, i.e. at the same level as the eoxserver directory: `trunk/soap_proxy`.

Initial Code Base

A first prototype implementing parts of the functionality has been developed under the O3S project. The source of this prototype will be copied to the *soap_proxy* repository and form the basis for further development.

RFCs and Decision Process

In the early stages, development surrounding of *soap_proxy* not directly affecting eoxserver will be undertaken in a relaxed manner compared to the RFC based decision taking that prevails for eoxserver.

All non trivial changes to the *soap_proxy* core will be announced for discussion on the eoxserver-dev mailing list, but will not undergo the RFC voting process unless there is a direct impact on any actual eoxserver functionality.

Once the transition phase of the integration has been completed, the development of *soap_proxy* will follow the standard RFC based decision taking.

License

Soap_proxy will use either GPL or a MapServer-style license, this is yet TBD.

Wiki, Trac, Tickets

Soap_proxy will use all of the eooserver support infrastructure.

References

[**AXIS**] <http://axis.apache.org/axis2/c/core/>

Voting History

Motion Adopted on 2011-05-30 with +1 Martin Paces, Stephan Meißl, Fabian Schindler, Milan Novacek

Traceability

Requirements “N/A”

Tickets “N/A”

RFC 11: WPS 1.0.0 Interface Prototype

Author Martin Paces

Created 2011-07-20

Last Edit 2011-07-21

Status DRAFT

Discussion <http://www.eoxserver.org/wiki/DiscussionRfc11>

Introduction

This RFC describes the design and implementation of the OGC WPS 1.0.0 Interface prototype. The WPS (Web Processing Service) interface prototype adds the processing functionality to the EOx-Server and is capable of invocation of both synchronous and asynchronous processes invoked using either XML or KVP encoding as described in OGC 05-007r7 *OpenGIS Web Processing Service* document.

Description

The implementation extends the set of EOx-Server’s OWS service handlers by the WPS specific interface. Namely, it adds following handlers

- WPS service handler
- WPS 1.0.0 version handler
- WPS *GetCapabilities* operation handler

- WPS *DescribeProcess* operation handler
- WPS *Execute* operation handler

The added WPS functionality could be split three (currently separated) logical parts:

- WPS interface and operation logic (subject to this RFC)
- WPS data model and generic process class (loosely based on PyWPS, currently separated from the interface and operation logic)
- WPS process instances – user defined processes, ancestors of the generic service class (completely independent of the EOX-Server, not subject to this RFC)

WPS Interface and Operation Logic

This part implements the actual OWS service handlers and it is tightly coupled with the EOX-Server. It parses and interprets the operation request and generates the operation responses reusing existing parts of the EOX-Server (primarily the XML and KVP request decoders). This interface has access to the installed WPS process instances (implemented as python modules) and it reads their descriptions. In case of the *Execute* operation it fetches the parsed input data to the selected process instance, triggers the actual execution of the process, and generates the status responses and handles output data XML packing and encoding.

In case of a synchronous execution the WPS processes are executed in context of the EOX-Server's OWS request. In case of an asynchronous WPS processes a dedicated OS process is started from the context of the EOX-Server's OWS request.

This part is distributed under the EOX-Server's MapServer-like open source licence.

WPS Data Model and Generic Process Class

This part (not subject to this RFC) is loosely based on the WPS Process API of the :PyWPS: SW. Due to the flaws of the original data model and requirements of the EOX-Server integration the the original :PyWPS: code was substantially modified (practical rewritten) leaving only traces of the generic (parent) WPS Process class.

The work is based on the stable :PyWPS: version 3.1.0. The reason we have replaced the original data model was that it had several design and implantation flaws (e.g., the way how the multiple input and output data occurrences were handled, bounding box data handling and encoding, the way how input sequences were detected). After first initial correcting attempts we gave up and rewrote the model from scratch. The generic Process class was modified: (i) due to the the changes made to the data model, (ii) removing unused parts of code (e.g., useless class reinitialization, Grass integration, internationalization), (iii) and finally due to the needs of the EOX-Server integration.

Despite the only fragments of the original :PyWPS:, this code was derived from the :PyWPS: and it is distributed under the terms of the original GPL licence.

WPS Process Instances

The process instances are not subject to this RFC and should be written by the WPS users to provide the desired functionality. The processes are created as separated python modules each containing a single customized subclass of the generic process class. The unique process identifier is the same as the name of python module (file's base name), the rest of the process description is defined by an implementer in the class definition.

We provide set of sample demo process samples covering from basic to most advanced cases. This part is distributed under the terms of PyWPS GPL licence.

Transition to Operation - Issues to be resolved

The existing prototype has still a couple issues to be resolved before operational deployment.

- licence issues - the WPS Process's data model and parent Process's class shall be merged with the WPS Interface and Operation logic and distributed together under the same licence terms
- resource tracker - there should be a resource tracker looking after the used resources, i.e., stored files and executed asynchronous processes. Each of these resources shall be monitored and released (deleted in case of unused files, properly killed in case of "zombie" processes) once is not usefull anymore.

Governance

Source Code Location

WPS Interface

Currently the Interface code can be downloaded from the WPS sandbox:

http://eoxserver.org/svn/sandbox/sandbox_wps

WPS - Data Model and Generic Process Class

The code derived from the PyWPS (only the parts needed for EOX-Server integration) can be found at:

<http://o3s.eox.at/svn/deliverables/developments/wps/server>

WPS - Demo Processes

The demo services are available at:

https://o3s.eox.at/svn/deliverables/developments/wps/wps_demo_services/

Initial Code Base

A first prototype implementing parts of the functionality has been developed under the O3S project.

RFCs and Decision Process

TBD

License

WPS Interface prototype shall be distributed under the terms of the EOX-Server's MapServer-like licence.

The other parts required by the WPS functionality are available under the terms of the [PyWPS] GPL licence.

Wiki, Trac, Tickets

TBD

References

[PyWPS] <http://pywps.wald.intevation.org/>

Voting History

N/A

Traceability

Requirements “N/A”

Tickets “N/A”

RFC 12: Backends for the Data Access Layer

Author Stephan Krause

Created 2011-08-31

Last Edit \$Date\$

Status ACCEPTED

Discussion <http://eoxserver.org/wiki/DiscussionRfc12>

This RFC proposes the implementation of different backends that provide common interfaces for data stored in different ways. It describes the first version of the Data Access Layer implementation as well as changes to the Data Integration Layer that are caused by the changes to the data model.

Introduction

RFC 1: An Extensible Software Architecture for EOxServer (page 156) introduced the Data Access Layer as an abstraction layer for access to different kinds of data storages. These are most notably:

- data stored on the local file system
- data stored on a remote file system that can be accessed using FTP
- data stored in a rasdaman database

The term *backend* has been coined for the part of the software implementing data access to different storages.

This RFC discusses an architecture for these backends which is based on the extension mechanisms discussed in *RFC 2: Extension Mechanism for EOxServer* (page 163). After the *Requirements* (page 188) section the architecture of the Data Access Layer is presented. It is structured into a section describing the *Data Access Layer Data Model* (page 190) which consists basically of *Storages* (page 190) and *Locations* (page 190).

Furthermore, the necessary changes to the Data Integration Layer are explained. On the one hand these affect the *Data Model* (page 191) which is altered considerably. On the other hand new structures (*Data Sources* (page 191) and *Data Packages* (page 191)) that provide more flexible solutions for data handling by the Data Integration Layer and the layers that build on it.

Requirements

We may refer here to the *Backends Requirements* (page 159) section as well as the description of the *Data Access Layer* (page 177) in *RFC 1: An Extensible Software Architecture for EOxServer* (page 156). These state the need for different backends to access local and remote data in different ways and thus are the incentive for this RFC and the respective implementation.

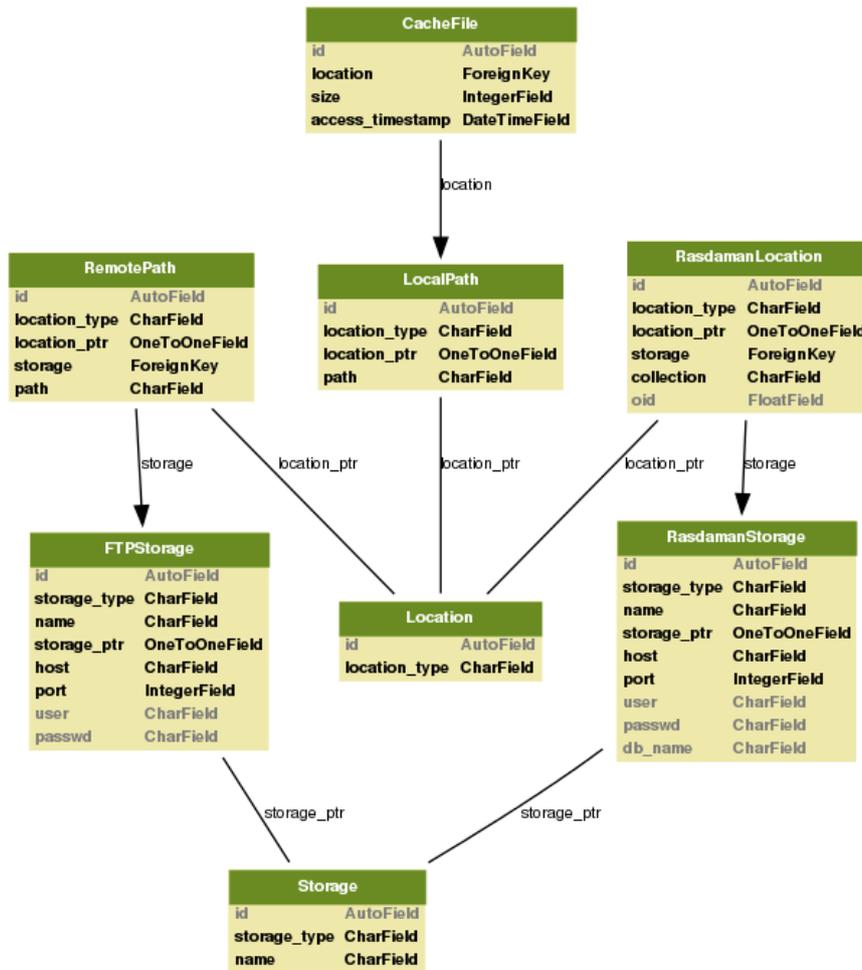


Fig. 3.4: Data Access Layer Database Model

Data Access Layer Data Model

The new database model for the Data Access Layer is shown in the figure below:

The core element of the Data Access Layer data model is the `Location`. A location designates a piece of data or metadata, actually any object that can be stored in one of the `Storage` facilities supported. Each backend defines its own subclasses of `Location` and `Storage` to represent repositories, databases, directories and objects stored therein.

The database model is embedded in wrappers that add logic to the model and provide common interfaces to access the data and metadata of the objects in the backend. Internally, they make use of the extension mechanism of *RFC2* (page 163) to allow to find and get the right model records and wrappers.

Last but not least, there is a *File Cache* (page 190) for storing files retrieved from remote hosts. The locations of the cache files are stored in the database so EOxServer can keep track of them and implement an intelligent cleanup process.

Storages

The `Storage` subclasses represent different types of storage facilities. In the database model, only FTP and rasdaman backends have their own models defined that contain the information how to connect to the server. This is not needed for locally mounted file systems, so the local backend does not have a representation in the database.

The wrapper layer constructed on top of the database model on the other hand knows three classes of storages that provide a common interface to access their data:

- `LocalStorage` which implements access to locally mounted file systems
- `FTPStorage` which implements access to a remote FTP server
- `RasdamanStorage` which implements access to a rasdaman database

Each of these storage classes is associated to a certain type of location.

The common interface for storages allows to retrieve their type and their capabilities. Depending on these capabilities the storage classes also provide methods for getting a local copy of the data and retrieving the size of an object as well as scanning a directory for files. At the moment these three methods are implemented by file-based backends only (`LocalStorage` and `FTPStorage`).

Locations

Locations represent the points where to access single objects on a storage facility. At the moment three types of locations corresponding to the three storage types are implemented:

- `LocalPath` defines a path on the locally mounted file system
- `RemotePath` defines a path on a remote server reachable via FTP
- `RasdamanLocation` defines a collection (database table) and oid corresponding to a single rasdaman array

Locations share a common interface that is closely related to the storage interface. So, given the storage capabilities, it is possible to fetch a local copy, retrieve the size of an object and scan the location for files. The `LocationWrapper` subclasses extend these interfaces to make storage specific location information (e.g. host name for remote storages) accessible.

File Cache

With the `CacheFileWrapper` class the Data Access Layer provides a very simple file cache implementation at the moment that serves to cache remote files retrieved via FTP. The cache keeps track of the files it contains using the `CacheFile` model in the database.

So far, no synchronization for data access is implemented, i.e. threads that are processing requests have no possibility to lock a cache file in order to prevent it from being removed by another thread or process (e.g. periodical cleanup process). This is foreseen for the future.

Changes to Data Integration Layer Data Model

In order to use the new possibilities brought by the implementation of the Data Access Layer, the Data Integration Layer had to be revised and changed considerably. Up until now there has been a strong link between the type of coverage and the way it was stored. Datasets had to be stored as files in the local file system whereas mosaics were stored in tile indexes. This strong link had to be weakened to allow for new combinations.

The solution is a compromise between flexibility and simplicity. Although one can think of many more combinations, we introduce three classes of so-called `DataPackage` objects. A data package combines a data resource with an accompanying metadata resource. Both resources are referred to by `Location` subclass instances. Now the three data package classes are:

- `LocalDataPackage` which combines a local data file with a local metadata file
- `RemoteDataPackage` which combines a remote data file with a remote metadata file (both reachable via FTP); it contains a `CacheFile` reference for data in the local cache
- `RasdamanDataPackage` which combines a rasdaman array with a local metadata file

Furthermore, the concept of data directories where to look up datasets automatically had to be revised in order to use the new capabilities of the Data Access Layer. They were replaced by a concept called data sources which includes local and remote repositories. The `DataSource` model combines a local or remote `Location` with a search pattern for dataset names. Automatic lookup of rasdaman arrays is not foreseen at the moment.

Like most database objects, data packages and data sources are accessible using wrappers that provide a common interface and add application logic to the data model.

Data Packages

The `DataPackageInterface` defines methods for high-level and low-level data access and for metadata extraction from the underlying datasets. It is implemented by wrappers for local, remote and rasdaman data packages (`LocalDataPackageWrapper`, `RemoteDataPackageWrapper` and `RasdamanDataPackageWrapper` respectively).

The implementation of the data package wrappers is based on the `GDAL`²²³ library and its Python binding for data access as well as for geospatial metadata extraction. It contains an `open()` method that returns a `GDAL` dataset providing a uniform interface for raster data from different sources and formats. For low-level data access a `getGDALDatasetIdentifier()` method is provided which allows to retrieve the correct connection string for `GDAL` and thus to configure `MapServer`.

Geospatial metadata is read from the datasets themselves at the moment. Note that this is not possible for rasdaman arrays so far, so automatic detection and ingestion of these is not enabled.

EO Metadata is read from the accompanying metadata file and translated into the internal data model of `EOxServer`. The existing metadata extraction classes have been revised in order to comply with the extensible architecture presented in *RFC 1* (page 156) and *RFC 2* (page 163).

Data Sources

The wrappers for data sources (`DataSourceWrapper`) provide the capability to search a local or remote location for datasets. At the moment only file lookup is implemented whereas automatic rasdaman array lookup has been omitted. This is mostly due to the fact that rasdaman arrays do not contain geospatial metadata and a separate mechanism has to be found to retrieve this vital information.

The wrapper implementations provide a `detect` method that returns a list of `DataPackageWrapper` objects with which coverages are initialized (using the geospatial and EO metadata read from the data package).

²²³ <http://www.gdal.org/>

Ingestion and Synchronization

The `Synchronizer` implementation in `eoxserver.resources.coverages.synchronize` has to be revised according to the changes in the Data Access Layer and Data Integration Layer.

The implementations for containers, i.e. Rectified Stitched Mosaics and Dataset Series, shall retrieve the data sources associated with a coverage and use its `detect` method to obtain the data packages included in it. Rectified or Referenceable Datasets are constructed from these. The interfaces of both should not change.

The interface of `RectifiedDatasetSynchronizer` on the other hand will have to change in order to allow for remote files to be ingested. In detail, the `create()` and `update()` methods will not expect a file name any more, but a location wrapper instance (either `LocalPathWrapper` or `RemotePathWrapper`). These can be generated by a call to the `LocationFactory` like this:

```
from eoxserver.core.system import System

factory = System.getRegistry.bind("backends.factories.LocationFactory")

location = factory.create(
    type = "local",
    path = "<path/to/file>"
)

...
```

Voting History

Motion To accept RFC 12

Voting Start 2011-09-06

Voting End 2011-09-15

Result +5 for ACCEPTED (including 1 +0)

Traceability

Requirements N/A

Tickets N/A

RFC 13: WCS-T 1.1 Interface Prototype

Author Martin Paces

Created 2011-09-13

Last Edit \$Date\$

Status ACCEPTED

Discussion <http://www.eoxserver.org/wiki/DiscussionRfc13>

Introduction

This RFC describes the design and implementation of the interface prototype of the *Open Geospatial Consortium* (OGC) *Web Coverage Service - Transaction operation extension* (WCS-T) [OGC 07-068r4]²²⁴ standard. The WCS-T extends the baseline WCS (allowing download of coverages only) by additionally allowing modifications of the stored coverages, namely, it allows adding, deleting, or updating of the coverages' data and their metadata.

²²⁴ http://portal.opengeospatial.org/files/?artifact_id=28506

WCS Transaction Operation

The WCS-T standard [OGC 07-068r4] defines an additional WCS *transaction* operation to perform modifications of the WCS Coverages. A single *transaction* contains one or more actions to be performed over coverages (coverage actions). The WCS-T standard requires that all WCS-T implementations shall support at least one action per request, multiple actions per request are optional.

The possible coverage actions are:

- Add - inserts new coverage and its metadata (required by all WCS-T implementations)
- Delete - removes an existing coverage and its metadata (optional)
- UpdateAll - replace data and metadata of an existing coverage (optional)
- UpdateMetadata - replace metadata of an existing coverage (optional)
- UpdateDataPart - replace data subset of an existing coverage (optional)

The supported optional features (multiple actions per request or optional coverage actions) shall be announced in the *ServiceIdentification* section of the WCS *Capabilities* XML document using the *Profile* XML element (see [OGC 07-068r4] for a detailed list of the applicable URNs).

Although not explicitly mentioned by the WCS-T standard, we assume the *transaction* operation shall be present in the *OperationMetadata* section of the WCS *Capabilities*.

The WCS-T standard allows XML encoded requests submitted as HTTP/POST requests. The KVP encoding of HTTP/GET requests is not supported by WCS-T since “the KVP encoding appears impractical without significantly restricting Transaction requests” [OGC 07-068r4]. Further, the [OGC 07-068r4] introduction mentions that the exchanged XML documents shall use the SOAP packaging, however, the examples are presented without the SOAP wrapping leaving this requirement in doubts.

The WCS-T requests can be processed synchronously or asynchronously. In the first case, the request is processed immediately and the transaction response is returned once actions have been processed successfully. In the latter case, the request is validated and accepted by the server returning simple acknowledgement XML document. The request is then processed asynchronously possibly much later than the acknowledgement XML document has been returned to the client. The asynchronous operation is triggered by presence of the *responseHandler* element in the WCS-T request. This element contains an URL where the response document should be uploaded.

All the data passed to the server by the WCS-T requests are in form of URL references. The support for direct data passing via MIME/multi-part encoded requests is not considered by the WCS-T standard.

The format of the ingested coverage data is not considered by the WCS-T standard at all. Neither it can be annotated by the WCS-T request nor by the WCS-T *OperationMetadata*. Thus we assume the format selection is left at discretion of the WCS-T implementation.

The WCS-T standard requires that certain metadata shall be provided by the client. These are geo-transformation, coverage description, and coverage summary. Apart from this mandatory metadata application specific metadata may be added by the implementation.

The WCS-T standard allows clients to submit their request and (created) coverages identifiers. These identifiers do not need to be used by the WCS-T server as they may collide with the identifiers of other requests or coverages, respectively, or simply not follow the naming convention of the particular WCS server. Thus the client provided identifiers are not binding for the WCS server and they rather provide a naming hint. As result of this the WCS-T client shall never rely on the identifiers provided to the WCS-T server but it shall always read the identifier returned by the WCS-T XML response.

EOxServer Implementation

The WCS *transaction* operations is implemented using the service handlers API of EOxServer. Since the WCS-T standard requires the version of the *transaction* operation to be ‘1.1’ (rather than the ‘1.1.0’ version used by other WCS operations) a specific WCS 1.1 version handler must have been employed. The operation itself is then implemented as a request handler.

Since the presence of the WCS-T operation needs to be announced by the WCS *Capabilities* the WCS 1.1.x *getCapabilities* operation request handlers have to be modified. Since the *Capabilities* XML response is generated by the MapServer (external library) the only feasible way to introduce the additional information to the *getCapabilities* XML response is to: i) capture the MapServer's response, ii) modify the XML document, and iii) send the modified XML instead of the MapServer's one.

The *transaction* request or response XML documents do not use the (presumably) required SOAP packaging. We have intentionally refused to follow this requirement in our implementation as the SOAP packing and unpacking is duty of EOxServer's *SOAP Proxy* component and our own implementation would rather duplicate the functionality implemented elsewhere.

Our implementation, by default, offers the WCS-T core functionality only. All the optional features such as multiple coverage actions per request or the optional coverage actions shall be explicitly enabled by EOxServer's configuration (see following section for details).

Both synchronous and asynchronous modes of operation are available. While the synchronous request are processed within the context of the WCS-T request handler the asynchronous requests are parsed and validated within the context of the WCS-T request but the processing itself is performed by the Asynchronous Task Processing (ATP) subsystem of EOxServer. Namely, the processing task is enqueued to the task queue and than later executed by one of the employed Asynchronous Task Processing Daemons (ATPD). More details about the ATP can be found in [ATP-RFC].

As it was already mentioned, the asynchronous mode of operation is triggered by presence of the *responseHandler* element in the WCS-T request and this element contains an URL where the response document should be uploaded. Our implementation supports following protocols:

- FTP - using the PUT command; username/password FTP authentication is possible
- HTTP - using POST HTTP request; username/password FTP authentication is possible

Secured (SSL or TLS) versions of the protocols are currently not supported.

The username/password required for authentication can be specified directly by the URL

```
scheme://[username:password@]domain[:port]/path
```

In case of FTP, when the paths point to a directory a new file will be created taking the request ID as the base file-name and adding the '.xml' extension. Otherwise a file given by the path will be created or rewritten.

The WCS-T implementation uses always pairs of identifiers (internal and public) for both request and (created) coverage identifiers. The public identifiers are taken from the WCS-T request, provided they do not collide with identifiers in use. In case of not supplied or colliding identifiers the public identifiers are set from the internal ones. The public identifiers are used in the client/server communication or for naming of the newly created coverages. The internal identifiers are exclusively used for naming of the internal server resources (asynchronous tasks, directory and file names, etc.)

Each WCS-T request, internally, gets a *context*, i.e. set of resources assigned to a particular request instance. These resources are: i) an isolated temporary workspace (a directory to store intermediate files deleted automatically once the request is finished), ii) an isolated permanent storage (a directory where the inserted coverages and their metadata is stored) and iii) in case of asynchronous mode of operation ATP task instance. These resources make use of the internal identifiers only.

EOxServer Configuration

The EOxServer's WCS-T implementation need to be configured prior to the operation. The configuration is set in EOxServer's 'eoxserver.conf' file. The WCS-T specific options are grouped together in the 'services.ows.wcst11' section.

The WCS-T options are:

- `allow_multiple_actions` (False/True) - allow multiple actions per single WCS-T request.
- `allowed_optional_action` (Delete,UpdateAll,UpdateMetadata,UpdateDataPart) - comma separated list of enabled optional WCS-T coverage action. Set empty if none.

- path_wcst_temp (*path*) - directory to use as temporary workspace
- path_wcst_perm (*path*) - directory to use as permanent workspace

Example:

```
...
# WCS-T 1.1 settings
[services.ows.wcst11]

# enable disable multiple actions per request
allow_multiple_actions=False

# list enabled optional actions {Delete,UpdateAll,UpdateMetadata,UpdateDataPart}
allowed_optional_actions=Delete,UpdateAll

# temporary storage
path_wcst_temp=/home/test/o3s/sandbox_wcst_instance/wcst_temp

# permanent data storage
path_wcst_perm=/home/test/o3s/sandbox_wcst_instance/wcst_perm
...
```

Coverages, Data and Metadata

The one and only currently supported format of pixel data is GeoTIFF.

All the necessary meta-data required by the EOxServer are extracted from the GeoTIFF annotation and (optionally) from the provided EO meta-data (see section below).

Due to the limitations of the current Coverage Managers' API of the EOxServer the current WCS-T implementation has following restrictions:

- only rectified grid coverages can be ingested;
- urn:ogc:def:role:WCS:1.1:CoverageDescription metadata are ignored and even not required as this information cannot be inserted to EOxServer anyway;
- urn:ogc:def:role:WCS:1.1:CoverageSummary metadata are ignored as this information cannot be inserted to EOxServer anyway;
- urn:ogc:def:role:WCS:1.1:GeoreferencingTransform metadata are ignored as this information is relevant to referenced data only
- urn:ogc:def:role:WCS:1.1:OtherSource metadata are ignored as this information cannot be inserted to EOxServer anyway.

WCS-T and Earth Observation Application Profile

In order to be able to ingest additional metadata as defined by the *WCS 2.0 - Earth Observation Application Profile* [EO-WCS] we allow the ingestion of client-defined EO-WCS metadata attached to the ingested pixel data. The EO-WCS XML is passed as coverage OWS Metadata XML element with 'xlink:role="http://www.opengis.net/eop/2.0/EarthObservation"'.

Governance

Source Code Location

http://eoxserver.org/svn/sandbox/sandbox_wcst

RFCs and Decision Process

TBD

License

The WCS-T implementation shall be distributed under the terms of *EOxServer's MapServer-like license* (page 263).

Wiki, Trac, Tickets

TBD

References

[OGC 07-068r4] http://portal.opengeospatial.org/files/?artifact_id=28506

[ATP-RFC] <http://eoxserver.org/doc/en/rfc/rfc14.html>

[EO-WCS] *TBD*

Voting History

Motion To accept RFC 13

Voting Start 2011-12-15

Voting End 2011-12-22

Result +3 for ACCEPTED

Traceability

Requirements *N/A*

Tickets *N/A*

RFC 14: Asynchronous Task Processing (ATP)

Author Martin Paces

Created 2011-10-25

Last Edit 2011-12-09

Status ACCEPTED

Discussion <http://www.eoxserver.org/wiki/DiscussionRfc14>

This RFC describes the Asynchronous Task Processing subsystem of the *EOxServer*.

Introduction

The *Asynchronous Task Processing* (ATP) subsystem, as the name suggests, extends the EOxServer functionality by ability to process tasks asynchronously, i.e., on background independently of the default EOxServer's synchronous client requests processing.

Although the ATP is designed primarily to support asynchronous request processing of OGC Web Services such as the Web Coverage Service transaction extension and/or the Web Processing Service, it is not limited to these and other application may use it as well.

The ATP employs the model of a single central task queue and one or more *Asynchronous Task Processing Daemons* (APTD) executing the pending tasks pulled from the task queue. A single ATPD is not restricted to a single processed task at time and can internally process multiple tasks concurrently, e.g., by employing a pool of worker processes assigned to multiple CPU cores.

The ATP subsystem is implemented as Django application using the DB model as the task queue. The underlying DB storage although it may be seen as suboptimal in terms of the performance and latency it assure tolerance of the subsystem to possible failures or maintenance shut-downs of both EOxServer or APTDs.

The ATP can be shared by multiple application at time as each task has its type (application to which it belongs) and each type of the task has a predefined handler subroutine. The shared nature of the APT subsystem allows fine control over the processing resources, e.g., the number of concurrently running task matching number of available CPU cores.

The ATP is primarily designed for resource demanding longer running tasks (10 seconds and more) which in case of synchronous operation could clog the system or lead to connection time-outs. On contrary, *light* tasks (less than 1 sec.) should preferably be executed synchronously as the extra ATP latency might be unfavourable.

Asynchronous Task Processing

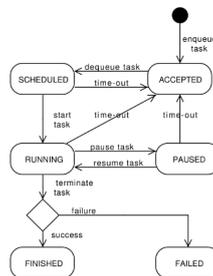


Fig. 3.5: Fig.1: ATP Task State Diagram

The ATP subsystem is capable of tracking of the tasks during their life cycle depicted by the Task state diagram Fig.1. The task can be in one of the following states:

- ACCEPTED - a new enqueued task waiting to be pulled by the processing daemon
- SCHEDULED - a task pulled (dequeued) by the processing daemon but not yet started
- RUNNING - a task being processed by the processing daemon
- PAUSED - a task which has been put on hold by the processing daemon and which is waiting to be resumed
- FINISHED - a task which has been finished successfully (terminal state)
- FAILED - a task which has been finished by a failure (terminal state)

When a task becomes identified as stalled (by exceeding the type specific time-out) it may be re-enqueued, i.e., the processing shall be terminated, enqueued as a new task again changing its status from one of the non-terminal states (SCHEDULED, RUNNING, PAUSED) to ACCEPTED. This procedure is implemented to avoid abandoned “zombie” tasks left, e.g., by an aborted processing daemon. However, this procedure is repeated only limited times (the count is task type specific, three by default), once the allowed restart's count is exceeded the task is marked as FAILED.

The history of the task's state transition is logged in order to provide information to the system operator.

The finished tasks are kept recorded for ever by default, however, this can be changed by a task type (application) specific retention time, which allow automatic removal of out-dated tasks, e.g., one day, week or month after their finish.

To inspect the state of the APT subsystem, a couple of simple Django html views has been created.

ATP DB Model

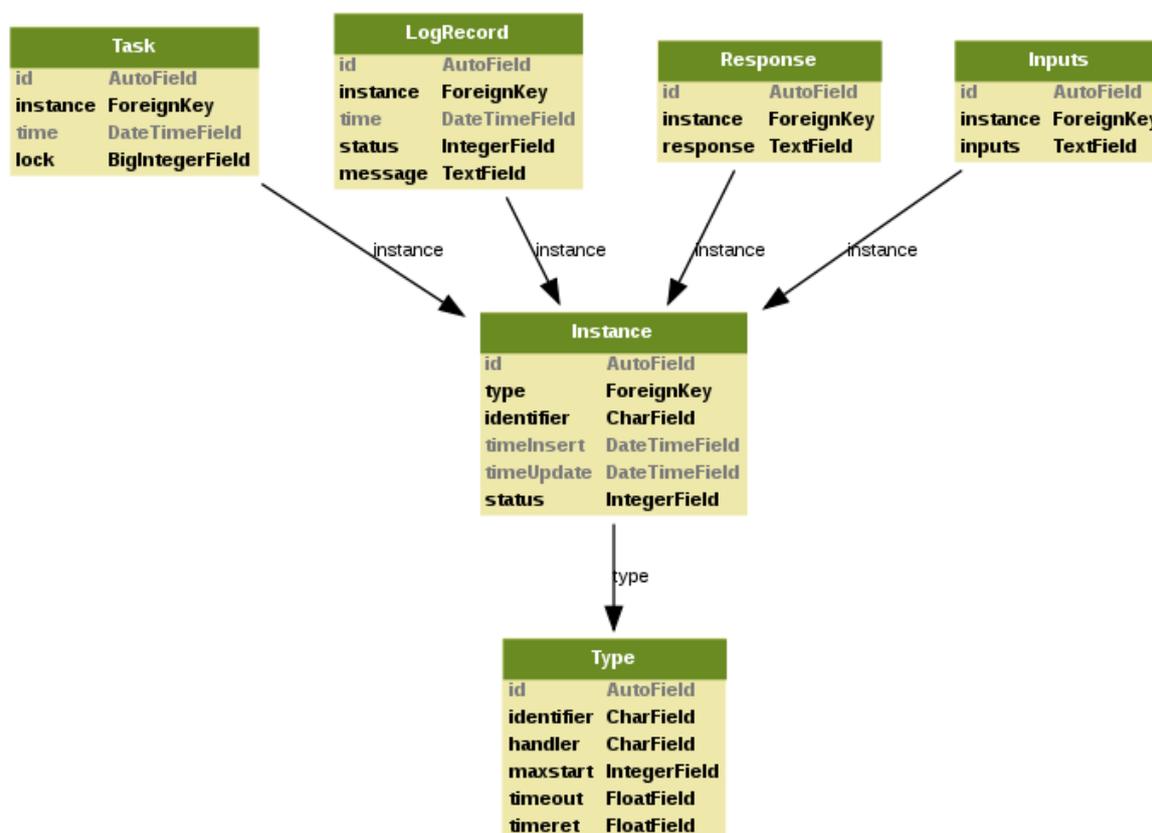


Fig. 3.6: Fig.2: ATP DB model

The APT Django DB model consists of six classes as depicted in Fig.2.

- **Type** - defining the type of task instance, its unique identifier, task handler (python subroutine), and the type specific parameters such as maximum unsuccessful attempts to start the task execution, time-out after the which the task is considered to be abandoned and re-enqueued for processing (e.g., due to ATPD failure), retention time to keep the record of the finished task.
- Instance - defining a single task instance, its identifier and current state.
- Inputs - record holding input parameters stored serialized (pickled) Python object
- Response - record holding the optional tasks output (most likely an XML response document or serialized Python object)
- LogRecord - single log entry. The log keeps history of the task's state transition.
- Task - single task queue record. The task table holds the accepted tasks, their enqueue time, ATPD assignment.

ATP API

The ATP subsystem provides simple API which allows:

- registering of new task type and its parameters (repeated registration updates the parameters)
- removal of unused task types (provided there is no instance of the removed type)
- enqueueing of new task instance and input parameters (implies creation of new task instance)
- dequeuing of enqueued instance (used by APTD)
- removal of finished tasks
- re-enqueueing of a non terminal state task
- changing of the task status
- adding and retrieval of the response (output)

Further the mandatory function prototype to define new handlers is given.

Governance

Source Code Location

http://eoxserver.org/svn/sandbox/sandbox_wcst

RFCs and Decision Process

TBD

License

The APT implementation shall be distributed under the terms of *EOxServer's MapServer-like license* (page 263).

Wiki, Trac, Tickets

TBD

References

Voting History

Motion To accept RFC 14

Voting Start 2011-12-15

Voting End 2011-12-22

Result +4 for ACCEPTED

Traceability

Requirements *N/A*

Tickets *N/A*

RFC 15: Access Control Support

Author Arndt Bonitz

Created 2011-11-14

Last Edit 2011-02-09

Status ACCEPTED

Discussion <http://eoxserver.org/wiki/DiscussionRfc15>

Overview

This RFC describes access control support for the EOxServer. The following figure gives an overview of the proposed access control implementation and its different components:

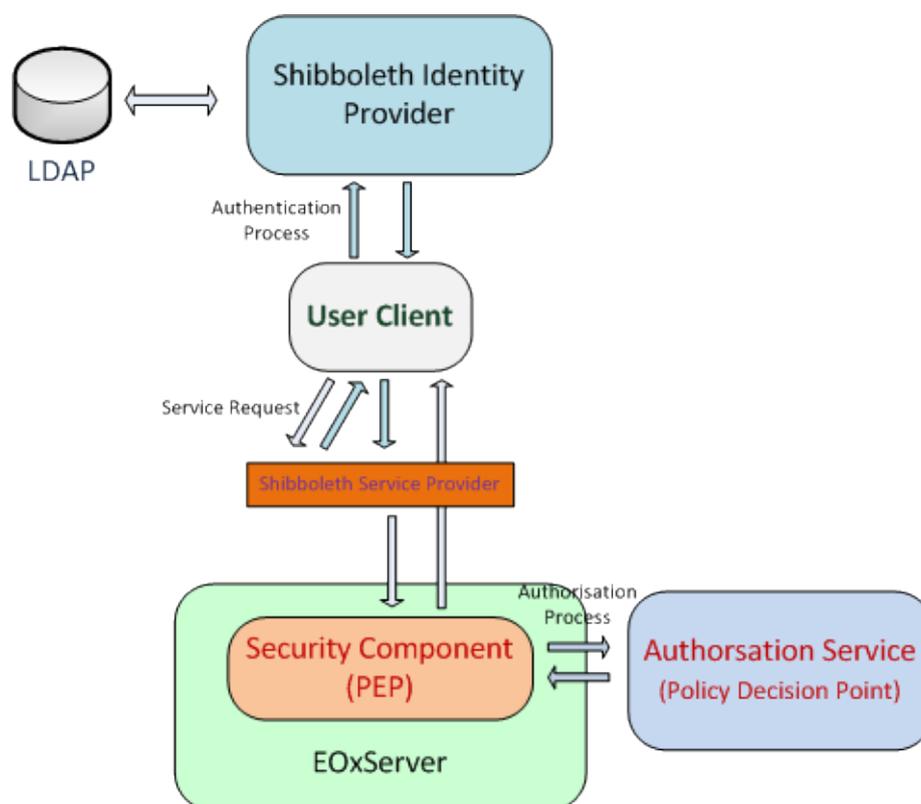


Fig. 3.7: EOxServer Access Control Implementation

The access control implementation relies on the Shibboleth framework²²⁵ and parts of the CHARON framework²²⁶, namely the CHARON Authorisation Service. The components are all released as Open Source. Shibboleth is used for the authentication of users; the CHARON Authentication Service is responsible for making authorisation decisions if a certain request may be performed.

Authentication

Authentication is not handled directly by the EOxServer components, but uses the Shibboleth federated identity management system. In order to do this, two requirements must be met:

- A Shibboleth Identity Provider (IdP) must be available for authentication

²²⁵ <http://shibboleth.internet2.edu/>

²²⁶ <http://www.enviromatics.net/charon/index.html>

- A Shibboleth Service Provider must be installed and configured in an [Apache HTTP Server](#)²²⁷ to protect the EOxServer resource.

A user has to authenticate at an IdP in order to perform requests to an EOxServer with access control enabled. The IdP issues a SAML token which will be validated by the SP.

If the user is valid, the SP adds the user attributes by the IdP to the HTTP Header of the original service requests and conveys it to the protected EOxServer instance. The whole process ensures, that only authenticated users can access the EOxServer.

Authorisation

As noted in the previous section, the Shibboleth system provides the underlying service and all asserted user attributes. These attributes can be used to make a decision if a certain user is allowed to perform an operation on the EOxServer. The authorisation decision is not made directly in the EOxServer, but relies on the CHARON Authorisation Service.

The Authorisation Service is responsible for the authorisation of service requests. It makes use of [XACML](#)²²⁸, a XML based language for access policies. The Authorisation Service is part of the [CHAORN](#)²²⁹ project. The EOxServer security components are only responsible for performing an authorisation decision request on the Authorisation Server and enforcing the authorisation decision.

EOxServer Security Component

The EOxServer security component is located in the package `eoxserver.services.auth.base` in the EOxServer source code directory. The implementation of the `PolicyDecisionPointInterface` for the proposed setup is included in `eoxserver.services.auth.charonpdp.py`, which is a wrapper for the CHARON Authorisation Service client. Every request for authorisation is encoded into a XACML Authorization Query and sent to the Authorisation Service. The decision (permit, deny) of the service is then enforced by the EOxServer.

A first implementation can be found in this [EOxServer sandbox](#)²³⁰ and there's also an [e-mail discussion](#)²³¹ about this in the dev mailing list archives.

Voting History

Motion Adopted on 2011-02-09 with +1 from Arndt Bonitz, Fabian Schindler, Stephan Meißl and +0 from Milan Novacek, Martin Paces

Traceability

Requirements N/A

Tickets N/A

RFC 16: Referenceable Grid Coverages

Authors Stephan Krause, Stephan Meissl, Fabian Schindler

Created 2011-11-24

Last Edit \$Date\$

Status ACCEPTED

²²⁷ <http://httpd.apache.org/>

²²⁸ <http://www.oasis-open.org/committees/xacml/#XACML20>

²²⁹ <http://www.enviromatics.net/charon/index.html>

²³⁰ http://eoxserver.org/browser/sandbox/sandbox_security

²³¹ <http://eoxserver.org/pipermail/dev/2011-October/000295.html>

Discussion <http://www.eoxserver.org/wiki/DiscussionRfc16>

This RFC proposes an implementation for Referenceable Grid Coverages as well as for the WCS 2.0 operations working on them.

The implementation is available in the SVN under http://eoxserver.org/svn/sandbox/sandbox_ref.

Introduction

Referenceable Grid Coverages are coverages whose internal grid structure can be mapped to a coordinate reference system by some general transformation. They differ from rectified grid coverages in that the coordinate transformation is not necessarily affine.

In the context of Earth Observation, raw satellite data can be seen as referenceable grid coverages. They are typically delivered as image files but do not have an affine transformation from the image geometry to a georeferenced coordinate system. Depending on the desired geocoding precision, the referencing transformation can be very complex involving additional data (DEMs) and orbit metadata.

EOxServer shall be able to deliver (subsets of) Earth Observation raw data in its original (referenceable grid) geometry using WCS 2.0 and EO-WCS. Furthermore, it shall implement easily computable approximate referencing algorithms based on ground control points (GCPs) in order to enable coordinate transformations and rectified previews of the original data using WMS.

For the time being, the implementation will focus on SAR image data collected by the ENVISAT-ASAR sensor made available by ESA.

Requirements

The main requirement source for Referenceable Grid Coverage implementation in EOxServer is the ESA O3S project. In the course of this project EOxServer shall be installed in front of a small archive of ENVISAT-ASAR data. In a first step, we will focus on covering the requirements of this use case, adding more generic referenceable grid support in future iterations.

The ENVISAT-ASAR data are available in ENVISAT .N1 original format.

Delivery of the original referenceable grid data shall be supported using WCS 2.0 and EO-WCS. Subsetting shall be supported in pixel coordinates (imageCRS) and in a coordinate reference system. The CRS subsets shall be mapped to pixel coordinates using a simple coordinate transformation based on GCPs.

No support for resampling (`size` and `resolution`) or reprojection (`outputcrs`) parameters is required as these are not applicable to referenceable grid coverages.

At least GeoTIFF shall be supported as output format. GCP and metadata information contained in the .N1 original file shall be preserved.

In order to support (rectified) WMS previews, a simple georeferencing algorithm based on GCPs shall be implemented. This shall be reused to provide rectified versions of referenceable grid coverages using WCS 2.0.

Implementation Details

Input Formats

The implementation for referenceable grid coverages relies on GDAL for input data and metadata (georeferencing information, GCPs). Any format that supports storage of GCPs with the dataset can be used. The two most important formats are the ENVISAT .N1 format and GeoTiff.

Referencing Algorithm and Subsetting

WCS 2.0 allows to define subsets either in the image CRS, i.e. pixel coordinates, or in some geographic or projected coordinate system. For rectified grid coverages geographic coordinates can be easily transformed to pixel coordinates in a straightforward way. This is not the case for referenceable grid coverages, though.

For referenceable grid coverages produced by Earth Observation missions, the “correct” referencing transformation is not known in general. Instead, there are many different algorithms some of them relying on different additional data and metadata (DEMs, orbit information).

For the purposes of the EOxServer Referenceable Grid Coverage implementation, a simple first order interpolation algorithm based on GCPs is used. This algorithm does not use any additional data or metadata. The rationale for this decision is that there is no way to advertise the actual referencing algorithm in WCS or WMS, and therefore the most simple and straightforward algorithm was used.

Subsets given in georeferenced coordinates are transformed to the image CRS using the inverse transformation algorithm based on GCPs. The implementation uses not only the corner coordinates of the subsetting rectangle but also intermediary points to calculate an envelope and thus to guarantee that the requested extent be included in the result.

Genuine Referenceable Grid Coverage Support in WCS 2.0

Referenceable Grid Coverages in their original geometry are available using the EO-WCS extension of WCS 2.0.

The current implementation supports the `subset` parameter and transforms the given subsets as indicated in the previous subsection. The `size` and `resolution` parameters are not supported as they do only apply to rectified grid coverages.

The `format` parameter options are implemented in the same way as for rectified grid coverages.

The `rangesubset` parameter is foreseen for implementation.

In order to be able to serve referenceable grid data, the original `WCS20GetCoverageHandler` was split up into `WCS20GetReferenceableCoverageHandler` and `WCS20GetRectifiedCoverageHandler`. While the latter one still relies on MapServer, the one for referenceable grid data uses the vanilla GDAL Python bindings as well as additional GDAL-based extensions written for the EOxServer project.

Metadata is read from the original dataset and tagged onto the result dataset using the capabilities of the respective GDAL format drivers. Depending on the driver implementation, the way the metadata is stored may be specific to GDAL.

Coverage Metadata Tailoring

The WCS 2.0 standard specifies that the complete referencing transformation be described in the metadata of a referenceable grid coverage. This is a major problem for Earth Observation data as in general there is no predefined transformation; rather there are several different possible algorithms of varying complexity that can be used for georeferencing the image, possibly involving Earth Observation metadata such as orbit information, GCPs and additional data such as DEMs.

Furthermore there is no way to define an algorithm and describe its parameters (e.g. the GCPs) in GML, but only the outcome of the algorithm, i.e. a pixel-by-pixel mapping to geographic coordinates. This would produce a tremendous amount of mostly useless metadata and blow up the XML descriptions of coverage metadata to hundreds of megabytes for typical Earth Observation products.

Therefore the current EOxServer implementation does not deliver any of the `gml:AbstractReferenceableGrid` extensions in its metadata. Instead a non-standard `ReferenceableGrid` element is returned that contains all the elements inherited from `gml:Grid` but no further information. This is only a provisional solution that will be changed as soon as an appropriate way to describe referencing metadata is defined by the WCS 2.0 standard or any of its successors.

Support for Rectified Data in WMS and WCS 2.0

The implementation of the WCS 2.0 (EO-WCS) GetCoverage request as well as the WMS implementation is based on MapServer which supports rectified grid coverages only. It is not possible to use any kind of GCP based referencing algorithm in MapServer directly.

GDAL provides a mechanism to create so-called virtual raster datasets (VRT). These consist of an XML file describing the parameters for transformation, warping and other possible operations on raster data. They can be generated using the GDAL C API and are readable by MapServer (which relies on GDAL as well).

In order to provide referenced versions of referenceable data, EOxServer creates such VRTs on the fly using the EOxServer GDAL extension. The VRT files are deleted after each request.

GDAL Extension

The EOxServer GDAL extension provides a Python binding to some C functions using the GDAL C API that implement utilities for handling referenceable grid coverages. At the moment the Python bindings are implemented using the Python `ctypes`²³² module.

The `eoxserver.processing.gdal.refertools` module contains functions for

- computing the pixel coordinate envelope from a georeferenced subset
- computing the footprint of a referenceable grid coverage
- creating a rectified GDAL VRT from referenceable grid data

All functions use a simple GCP-based referencing algorithm as indicated above.

The GDAL Extension was made necessary because the standard GDAL Python bindings do not support GCP based coordinate transformations.

Voting History

Motion Adopted on 2012-01-03 with +1 from Arndt Bonitz, Martin Paces, Fabian Schindler, Stephan Meißl and +0 from Milan Novacek

Traceability

Requirements “N/A”

Tickets “N/A”

RFC 17: Configuration of Supported Output Formats and CRSes

Author Stephan Krauses, Martin Pačes

Created 2012-05-08

Last Edit \$Date\$

Status ACCEPTED

Discussion n/a

This RFC proposes modifications of the EOxServer allowing configuration of

- the supported output formats for WMS and WCS
- the supported CRSes for WMS and WCS

The RFC presents the rationale and proposes data model changes and new global configuration options.

²³² <http://docs.python.org/library/ctypes.html>

Introduction

The reason for preparation of this RFC is the need to change the way how the supported (file) formats and CRSes (CRS - Coordinate Reference Systems) for raster data are handled by the EOxServer's WCS and WMS services to assure compliance to OGC standards, interoperability and configurability of the services.

In case of WMS, the formats and CRSes shall be listed in the WMS Capabilities.

In case of WCS, the supported formats and CRSes shall be reported by the WCS Capabilities (per service parameters) and in the Coverage Descriptions (per coverage parameters). Compatibility with the WCS 2.0.1 corrigendum and the upcoming WCS 2.0 CRS Extension document shall be assured.

Currently, only the native CRS of a dataset is reported in the metadata and only a small hard-coded set output file format is announced as supported (JPEG2000, HDF4, netCDF and GeoTIFF for WCS). Hence, there is no way to configure these parameters.

Furthermore, the underlying MapServer implementation does not return proper OWS exceptions if an CRS not advertised in the service capabilities or coverage descriptions is requested.

Supported CRSes and Output Formats in OGC Web Services

The table below gives an overview over the support for reporting CRS and output format metadata in different standards implemented by EOxServer.

Table 3.2: Support for CRS and output format metadata

Service and Version	Supported CRS	Supported Formats
WMS 1.1.0	per layer	per service
WMS 1.1.1	per layer	per service
WMS 1.3.0	per layer	per service
WCS 1.1.2	per coverage	per coverage
WCS 2.0.0	n/a	n/a
WCS 2.0.1	per service	per service

All services but the WCS 2.0 CRS extension (listed under WCS 2.0.1) allow for reporting CRSes for each coverage / layer individually; the CRS extension could still be amended, though.

On the other hand, only WCS 1.1.2 allows output format specification on a per coverage basis whereas all others standards allow to report supported formats in the global service metadata only.

The WCS 2.0.1 corrigendum introduces the concept of native CRSes and formats which are reported in the coverage description. The native CRS is the one the domain set uses.

Counterintuitively, the WCS 2.0.1 native file format is not necessarily the same as the file format of the stored data. Since not all source file formats are supported as the output file format (e.g. ENVISAT N1), it is rather the default format delivered when there no specific file format is requested (omitting the `FORMAT` parameter in `GetCoverage` requests).

Supported Output Formats and WCS 2.0.1 Native Format

As most services (all but WCS 1.1.2, see the table above) allow output format configuration only per service instance, we propose that the list of supported formats shall be kept in the global configuration. This can be most easily done by adding new items to the global configuration file `conf/eoxserver.conf`.

Due to the nature of the data transmitted by these services the configuration should be separate for WMS and WCS.

The EOxServer implementation for WCS 2.0 and EO-WCS requires three parameters to be defined for each supported format:

- the MIME type

- the name of the GDAL driver
- the default file extension

The possible format choices are restricted by the capabilities of the underlying SW components (MapServer and GDAL). The list of allowed formats can be found at http://www.gdal.org/formats_list.html.

Although the source format (i.e. the actual format of the stored data) could be determined for each coverage individually at runtime it is preferable to store this information in the database for performance reasons.

The actual native format announced by the WCS 2.0.1 compliant coverage description can differ from the source format as not every source format can be used as output format.

The implementation of the native format reporting for WCS 2.0.1 requires that EOxServer knows the mapping from the source to WCS 2.0.1 native format. As this mapping varies depending on the GDAL version, available external libraries or simply on the preference of the instance administrator the actual mapping shall be configurable, i.e., it shall be a configuration item in `conf/eoxserver.conf`.

For all the proposed configuration items reasonable default shall be provided.

Supported CRSes

All services but WCS 2.0.1 support per-coverage or per-layer reporting of CRSes. The WCS 2.0 CRS extension is not yet finished and it is suggested that it, too, should allow for CRS metadata being reported in the coverage description, although this provision is not included in the current draft of the document.

Currently, the EOxServer implementation of WMS and WCS sets the `ows_srs` MapServer parameter to the original CRS of a coverage. Thus the currently only announced CRS is the native CRS of the dataset.

This RFC proposes to introduce global configuration items for WCS and WMS, respectively, allowing definition of CRSes to be reported in addition to the native CRS. These CRSes shall also be used for EO-WMS layers corresponding to DatasetSeries.

In order to report a native CRS for Referenceable Grid Coverages the data model needs to be changed to include the SRID of the GCP projection of ReferenceableDatasets.

Proposed Implementation

Changes to the Data Model

For implementing the native format reporting in WCS 2.0.1, an additional field `gdal_driver_name` on the `LocalDataPackage` and `RemoteDataPackage` model shall be added. For the `RasdamanDataPackage` model, a dedicated database field is not necessary as the GDAL driver is already known because of the nature of the data package. The driver name should be provided by the `DataPackageWrapper` implementation.

In order to report the native CRS of Referenceable Datasets, a `srid` field shall be added to the `ReferenceableDatasetRecord` model.

Changes to the Configuration Files

The following new configuration settings are needed for output format handling:

- a list of GDAL formats with MIME types and a flag indicating if the format is writable or read-only
- a list of MIME types to be reported as supported formats in WMS
- a list of MIME types to be reported as supported formats in WCS
- a default format MIME type to be used for native format reporting in WCS 2.0.1
- an optional mapping of source format to for native format reporting in WCS 2.0.1

The list of GDAL formats shall be configured in a CSV-like separate configuration file in `conf/formats.conf`. Each line in the file shall correspond to a given format. The syntax is as follows:

```
<GDAL driver name>,<MIME type>,<either "rw" for writable or "ro" for read-only_
↳formats>,<default file extension>
```

e.g.:

```
GTiff,image/tiff,rw,.tiff
```

Empty lines shall be ignored as well as any comments started by single # (hash) character and ended by the end of the line.

A default configuration (`default_formats.conf`) and a template (`TEMPLATE_formats.conf`) shall be included in the `eoXserver/conf` directory. The default configuration shall only be used as a fall-back if no `formats.conf` file is available in the instance `conf` directory.

The other configuration settings shall be defined in `conf/eoXserver.conf`:

```
[services.ows.wcs]
supported_formats=<MIME type>[,<MIME type>,...]

[services.ows.wms]
supported_formats=<MIME type>[,<MIME type>,...]

[services.ows.wcs.wcs20]
default_native_format=<MIME type>
source_to_native_format_map=[<src MIME type>,<dst MIME type>[,<src MIME type>,<dst_
↳MIME type>,...]
```

The following new configuration settings are needed for CRS handling:

- a list of supported CRS IDs (SRIDs) for WMS layers
- a list of supported CRS IDs (SRIDs) for WCS coverages

The respective entries in `conf/eoXserver.conf`:

```
[services.ows.wcs]
supported_crs=<SRID>[,<SRID>,...]

[services.ows.wms]
supported_crs=<SRID>[,<SRID>,...]
```

Default settings shall be defined in `eoXserver/conf/default.conf`.

Module `eoXserver.resources.coverages.formats`

In order to support output format handling a dedicated module shall be implemented that

- reads the list of GDAL formats from the configuration files
- map GDAL driver names to MIME types and vice versa
- map MIME type (i.e., format) to default file extensions
- map source format to WCS 2.0.1 native format

Changes to the Service Implementations

The WMS and WCS modules need to be altered to use the new global settings in the service and layer / coverage configuration.

The hard-coded format settings in WCS 2.0 (`eoxserver.services.ows.wcs.wcs20.getcov` module) shall be removed.

The GDAL driver name obtained from the `DataPackageWrapper` implementation shall be translated at runtime to the respective MIME type using the functionality provided by `eoxserver.resources.coverages.formats` module (including the translation from the source MIME type to the WCS 2.0.1 native MIME type).

Changes to the Administration Tools

The `create_instance` command shall copy the template format configuration file to the `conf` directory of the instance.

The Coverage Managers shall store the GDAL driver name of the native format in the database.

Voting History

Motion To accept RFC 17

Voting Start 2012-05-11

Voting End 2012-05-17

Result +5 for ACCEPTED

Traceability

Requirements N/A

Tickets N/A

RFC 18: Operator Interface Architecture

Author Stephan Krause, Fabian Schindler

Created 2012-05-08

Last Edit \$Date\$

Status PENDING

Discussion n/a

The new Operator Interface of EOxServer shall become the main entrance point for operators who want to administrate an EOxServer instance. The Web UI design shall focus on usability and support for frequent administration tasks.

The architecture of the Operator Interface shall be modular and extensible in order to accomodate for future extension and facilitate the maintenance of the software.

Introduction

At the moment operators have two possibilities to administrate an EOxServer instance:

- Command Line Tools
- Administration Web Client

The current Administration Client implementation is based on the `django.contrib.admin`²³³ package and very tightly coupled with the data model of EOxServer. Whereas this approach has made the development considerably easier it has several severe drawbacks with respect to usability and safety of the system:

- the EOxServer data model is fairly complicated and handling it requires a deep understanding of the EO-WCS standard as well as Django concepts like model inheritance
- certain actions trigger long-running processing tasks on the server side that are so far hidden from the operators
- there is no support for asynchronous requests which would be the preferred method
- error reporting and status monitoring is only minimal
- the current Admin Interface allows to edit database records without checks for consistency; the danger of breaking the system unintentionally is quite high

Therefore a new web-based Operator Interface shall be designed that facilitates the administration tasks. It shall be more usable in the sense that

- the design shall focus on frequent administration tasks rather than the data model
- the interface shall provide guidance for operators
- safety shall be increased by checking the consistency of input data and organizing the operator actions in a way that precludes unintentionally breaking the system
- the operator shall have an overview of the processing tasks going on in the backend

From the software point of view, the design shall focus on

- modularity and extensibility, thus preparing for future extensions of EOxServer and increasing maintainability
- reusing existing administration code like Coverage Managers
- separation of model, view and controller components where model and controller components should be concentrated on the server side and the view on the client side

Requirements

The Operator Interface shall support the most frequent tasks for administration. These include:

- registering a Dataset
- handling the Range Types
- creating a Dataset Series
- creating a Stitched Mosaic
- deleting a Dataset, Dataset Series or Stitched Mosaic
- adding a Dataset to a Dataset Series or Stitched Mosaic
- removing a Dataset from a Dataset Series or Stitched Mosaic
- creating / adding / removing a data source to/from a Dataset Series or Stitched Mosaic
- viewing the logs
- enabling / disabling of components
- user management

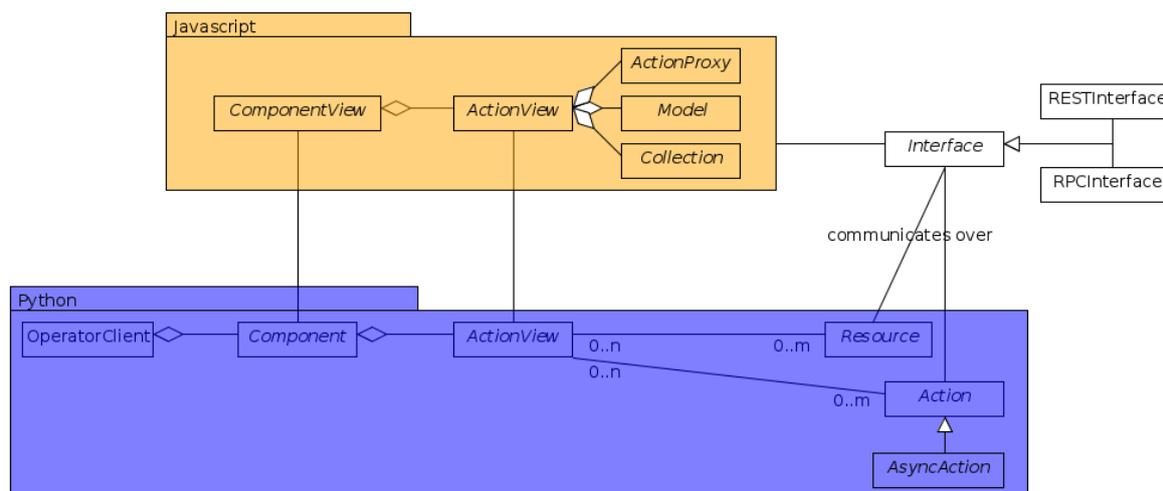


Fig. 3.8: The Operator Interface structure expressed in a UML class diagram.

Basic Concepts

The Operator Interface shall be organized in so called Operator Components. Operator Components correspond to groups of related packages and modules of EOxServer or its extensions. The most important components at the moment are `eoxserver.core` and `eoxserver.resources.coverages` (page 239).

An Operator Component bundles Actions and Views related to the specific EOxServer component in the backend.

Actions provide an interface for operators to edit the system configuration including the data and metadata stored in the database. Most Actions are related to resources, e.g. coverages or Dataset Series.

In order to make the functionality of these Actions available, the Operator Interface shall include Action Views. Action Views shall group actions and information that are closely related to each other.

Each Operator Component may contain several Action Views. They represent a UI for access to the actions in the backend. Several Actions may be attached to a single Action View, and Actions may appear in several Action Views.

For example, an Action View might show a list of Rectified Datasets with basic metadata which allows to create and delete items. Creation and deletion should each be modeled as Actions on the server side. Another Action View may show the whole information for a single Rectified Dataset and include forms and inputs to edit the metadata.

As far as possible, the Action Views should be composed of reusable Widgets. Widgets consist of HTML and/or JavaScript. The aforementioned list of Rectified Datasets would be a typical example. It could be used also in the Dataset Series View.

The core implementation of the Operator Interface shall provide reusable components to build Widgets of (e.g. lists ...).

The communication between the Action Views and the underlying Actions should be done via specific Interfaces. One REST-based interface shall be implemented which shall allow to read data and metadata to be displayed, and one RPC-based interface shall be implemented in order to trigger actions on the server side.

Detailed Concept Description

In this chapter, the introduced concepts will be elaborated in detail.

²³³ <https://docs.djangoproject.com/en/1.8/ref/contrib/admin/#module-django.contrib.admin>

Layout of the Operator Interface

The entry point to the operator interface shall be a dashboard-like page. It is envisaged to present a tab for each Operator Component; this tab shall contain an overview of the Action Views the Operator Component exhibits.

So, on the client side, each Operator Component should provide:

- A name for the Operator Component that will be shown as caption of the tab
- the overview of the Operator Component, which links to the Action Views
- the Action Views
- the Widgets used in the Action Views
- a widget to be displayed on the entry page dashboard (optional)

Each visual representation of the Operator Interface, namely the entry page dashboard, the Operator Component overview and the Action Views consist of:

- A Django HTML template
- A JavaScript View class
- A python class, entailing arbitrary information and “glue” between the other two parts

Only the third part needs to be adjusted when creating a new visual element, for both the template and the JavaScript class defaults shall help with the usage.

Action Views and Operator Component overviews should fit into the same basic layout; customizable CSS should be used for styling. The design of the entry page design (dashboard) may differ from the design of the sub-pages.

Components and Operator Components

Proposed Operator Components:

- User Management
- Configuration Management
- Action Control Center
- Coverages

Action Views

Proposed Action Views:

- User Management
 - add/delete users
 - edit permissions
- Configuration Management
 - enable and disable components
 - edit configuration settings
- Action Control Center
 - overview over running and completed actions
 - detail views for actions, including status and logs
- Coverages
 - For both Rectified and Referenceable datasets:

- * list view including limited update and delete actions
- * detail view including update and delete actions
- * create view to create a new dataset
- For both Rectified Stitched Mosaics and Dataset Series
 - * list view including limited update and delete actions
 - * detail view including update, delete and synchronize actions and a list display of all contained datasets and data sources including actions to insert/remove data sources or datasets
 - * create view to create a new dataset
- list view of Range Types with create, limited update and delete actions
- detail view of Range Types with update and delete actions and a list display of all included Bands with update actions
- list view of Nil Values with create, update and delete actions

Actions

The Actions shall be represented by corresponding Python classes on the server side. Actions shall be reusable in the sense that they can also be invoked using a CLI command.

Most Actions are tied to resources like coverages. Resources in that sense should not be confused with database models. In most cases, a resource will be tied to a higher-level object: coverage resources for instance shall be tied to the wrappers defined in `eoxserver.resources.coverages.wrappers`.

It should be possible to invoke Actions in synchronous and asynchronous mode. For the asynchronous mode, the existing facilities of the *Asynchronous Task Processing* (page 113) (the `eoxserver.resources.processes` (page 239)) shall be adapted and extended. For this purpose, the `eoxserver.resources.processes.models.LogRecord` shall receive an additional field `level`, which specifies the log level the log record was created with. This allows easy filtering for a minimum log level and e.g: only show errors and warnings raised during a process.

Every Action shall expose methods to

- validate the parameters
- start the Action and return the ID of that action
- stop the Action
- check the status of the Action
- check the log messages issued by the Action (maybe this is better implemented using the Resource mechanism)

On the client side, Actions are wrapped with ActionProxy objects that offer an easy API and abstraction for the remote invocation of the Actions methods. For Asynchronous Action the AsyncActionProxy offers a specialization.

Resources

Resources are an interface to the data stored as models in the database but also custom data sources are possible. When applied to models, a resource allows the create, read, update and delete (CRUD) methods, but this may be restricted per resource for certain models where the modification of data requires a more elaborate handling.

On the client side, Resources are wrapped in Models and Collections, which provide a layer of abstraction and handle the communication with and consume the REST interface offered by the server. A Model is the abstraction of a single dataset and a Collection is a set of models in a certain context.

Both Models and Collections offer certain events, to which the client can react in a suitable manner. This may trigger a synchronization of data with the server or a (re-)rendering of data on the client in an associated view. Additionally, models offer validation, which can be used for example to check if all mandatory fields are set, or inputs are syntactically correct.

Interfaces

The following interfaces will be used to exchange data between the server and the client:

RPC Interface

Actions shall be triggered via the RPC Interface. Invocation from the Operator Interface can be synchronous or asynchronous. Incoming requests from the Operator Interface shall be dispatched to the respective Actions using a common mechanism that implements the following workflow:

- validate the parameters conveyed with the request, using the Action interface
- in case they are invalid, return an error code
- in case they are valid, proceed
- queue the Action in the asynchronous processing queue
- return a response that contains the Action ID

Using the Action ID, the Operator Interface can

- check the status of the Action
- view the log messages issued by the Action
- cancel the Action

REST Interface

The REST interface shall be used for resource data retrieval and simple modification. Usually a REST interface is tightly bound to a database model and its fields. Thus modification of data via REST should only be possible in simple situations where there is no dependency to other resources and no other synchronization mechanism necessary.

Where the REST interface is not applicable, the RPC interface shall be used.

Directory Structure

For the server part, the directory structure of the operator interface follows the standard guidelines for Django apps (as created with the *django-admin.py startapp* command):

```
operator/
|-- action.py
|-- common.py
|-- component.py
|-- __init__.py
|-- resource.py
|-- sites.py
|-- static
|   |-- operator
|       |-- actions.js
|       |-- actionviews.js
|       |-- componentviews.js
|       |-- main.js
```

```
|         |-- router.js
|         |-- widgets.js
|-- templates
|   |-- operator
|     |-- base_actionview.html
|     |-- base_component.html
|     |-- operatorsite.html
```

In the templates directory all Django templates are held. It is encouraged to use the same scheme for all components to be implemented.

The static files are placed in the sub-folder “operator” which serves as a namespaces for javascript module retrieval. All components shall use an additional unique subfolder to avoid collision. For example: “operator/coverages”.

Implementation Details

In this chapter, the proposed implementation API of components explained.

Implementing Components

To create a component, one simply shall have to subclass the abstract base class provided by the Operator Interface API. It shall be easily adjustable by using either a custom JavaScript view class or a different django template.

To further improve the handling of components, several default properties within the subclass can be used, like title, name, description or others. Of course default values shall be provided.

Components are registered by the Operator Interface API function `register()`, which shall be sufficient to append it to the visualized components.

Example:

```
import operatorinterface as operator

class MyAComponent (operator.Component) :
    dependencies = [SomeOtherComponent]
    name = "ComponentA"
    javascript_class = "operator/component/MyAComponentView"

operator.site.register(MyAComponent)
```

Implementing Action Views

The implementation of action views is very much like the implementation of components and should follow the same rules concerning JavaScript view classes and django templates.

Additionally it shall have two fields named `actions` and `resources`, each is a list of Action or Resource classes.

Example:

```
class MyTestActionView (operator.ActionView) :
    actions = [MyTestAction]
    resources = [ResourceA, ResourceB]
    name = "mytestactionview"
    javascript_class = "operator/component/MyTestActionView"
```

Implementing Resources

Implementing Resources should be as easy as implementing actions. As with Actions, Resources are implemented by subclassing the according abstract base class and providing several options. The only mandatory arguments shall be the Django model to be externalized, optional are the permissions required for this resource, maybe means to limit the access to read-/write-only (maybe coupled to the provided permissions) and the inc-/exclusion of model fields.

Example:

```
class MyResource(ModelResource):
    model = MyModel
    exclude = ( ... )
    include = ( ... )
    permissions = [ ... ]
```

Implementing Actions

To implement a new Action, it shall be enough to inherit from an abstract base class and implement the required methods. Once registered the operator framework shall handle the URL and method registration.

Example:

```
class ProgressAction(BaseAction):
    name = "progressaction"
    permissions = [ ... ]

    def validate(self, params):
        ...

    def start(self):
        ...

    def status(self, obj_id):
        ...

    def stop(self, obj_id):
        ...

    def view_logs(self, obj_id, timeframe=None):
        ...
```

Access Control

The Operator Interface itself, its Resources and its Actions shall only be accessible for authorized users. Also, the Interface shall distinguish between at least two types of users: administrative users and users that only have reading permissions and are not allowed to alter data. The permissions shall be able to be set fine-grained, on a per-action or per-resource basis.

It is proposed to use the Django built-in auth framework and its integrations in other software frameworks.

Configuration and Registration of Components

On the server side, the Operator Interface is set up similar to the Djangos built-in Admin Interface. To enable the Operator Interface, its app identifier has to be inserted in the *INSTALLED_APPS* list setting and its URLs have to be included in the URLs configuration file.

Also similar to the Admin Interface, the Operator Interface provides an *autodiscover()* function, which sweeps through all *INSTALLED_APPS* directories in search of a *operator.py* module, which shall contain the apps setup of Components, Action Views, Actions and Resources.

Example Component: Coverage Component

This chapter explains an the example component to handle all kinds of interactions concerning coverages, mosaics and dataset series respectively all types of assorted metadata.

Requirements

As described earlier, the interactions shall entail creating/updating/deleting coverages and containers aswell inserting coverages into containers. Additionally users shall also trigger a synchronization process on rectified stitched mosaics and dataset series. As this may well be a time-consuming task, scanning through both the database and the (possibly remote) filesystem, it shall be handled asynchronously and output status messages.

Last but not least, all coverage metadata shall also be handled, including geo-spatial, earth observational and raster specific metadata.

The above requirements can be summarized in the following groups:

- Coverage Handling (also includes geospatial and EO-meta-data as the relation is one-to-one)
- Container Handling (same as above)
- Range Type Handling (as other more tied meta-data is handled in the other sections)

The requirement groups will be implemented as Action Views on the client, using specific widgets to allow interaction.

Server-Side implementation

The identified requirements have several implications on the server side. First off the three Action Views need to be declared to implement the three groups of requirements listed above and suited with the needed resources and actions.

Resources

For simple access to the internally stored data, a list of Resources need to be defined: one for each coverage/container type, one for range types, bands and nil values and also for data sources.

For asynchronous tasks, also the running tasks and their logs need to be exposed as resources.

Actions

The actions derived from the requirements can be summarized in the following list: add coverage to a container, remove a coverage from a container, add a data source to a container, remove a datasource from a container, manually start a synchronization process for a container. The first two actions can likely be handled synchronously as the management overhead is potentially not as high as with the latter three actions. Thus the introduced actions can be split into synchronous and asynchronous actions.

Additionally, for creating/deleting coverages and containers is done by using Actions instead of their Resources, because it involves a higher order of validation and additional tasks to be done which are too complex and unreliable if controlled by the server.

Summary

The following classes with their according hierarchical structure has been identified.

Component	Action Views	Resources	Actions
Coverages	Coverage Handling	Rect. Coverages	Add to Container
		Ref. Coverages	Remove from Container
		Rect. Mosaics	Create Coverage
		Range Types	Delete Coverage
		Bands	
		NilValues	
	Container Handling	Coverages	Add Coverage
		Rect. Mosaics	Remove Coverage
		Dataset Series	Add Datasource
		Logs	Remove Datasource
			Synchronize
			Create Container
		Delete Container	
	RangeType Handling	Range Types	
		Bands	
NilValues			

Client-Side implementation

From the requirements we already have designed three Action Views, which will be implemented as Backbone views. Each offered resource from the server will have a Backbone model/collection counterpart communicating with that interface. Similarly each action will have a proxy class on the client side.

Views

The hierarchy of the client views can be seen in the following figure.

Models/Collection

Each offered resource is encapsulated in a model and collection. The following figure shows the relation of the model/collection layout:

ActionProxies

For each Action on the server, an ActionProxy has to be instantiated on the client which handle the communication with the server. For the three Actions that are running asynchronously, a special ActionProxy subclass is used. The following figure shows which actions are handled synchronously and which follow an asynchronous approach.

Technologies Used

On the server side, the Django framework shall be used to provide the basic functionality of the Operator Interface including specifically the URL setup, HTML templating and request dispatching.

To help publishing RESTful resources, the django extension [Django REST framework](http://django-rest-framework.org/)²³⁴ can be used. It provides a rather simple, yet customizable access to database model. It also supports user authorization as required in the chapter *Access Control* (page 215). The library is available under the BSD license.

²³⁴ <http://django-rest-framework.org/>

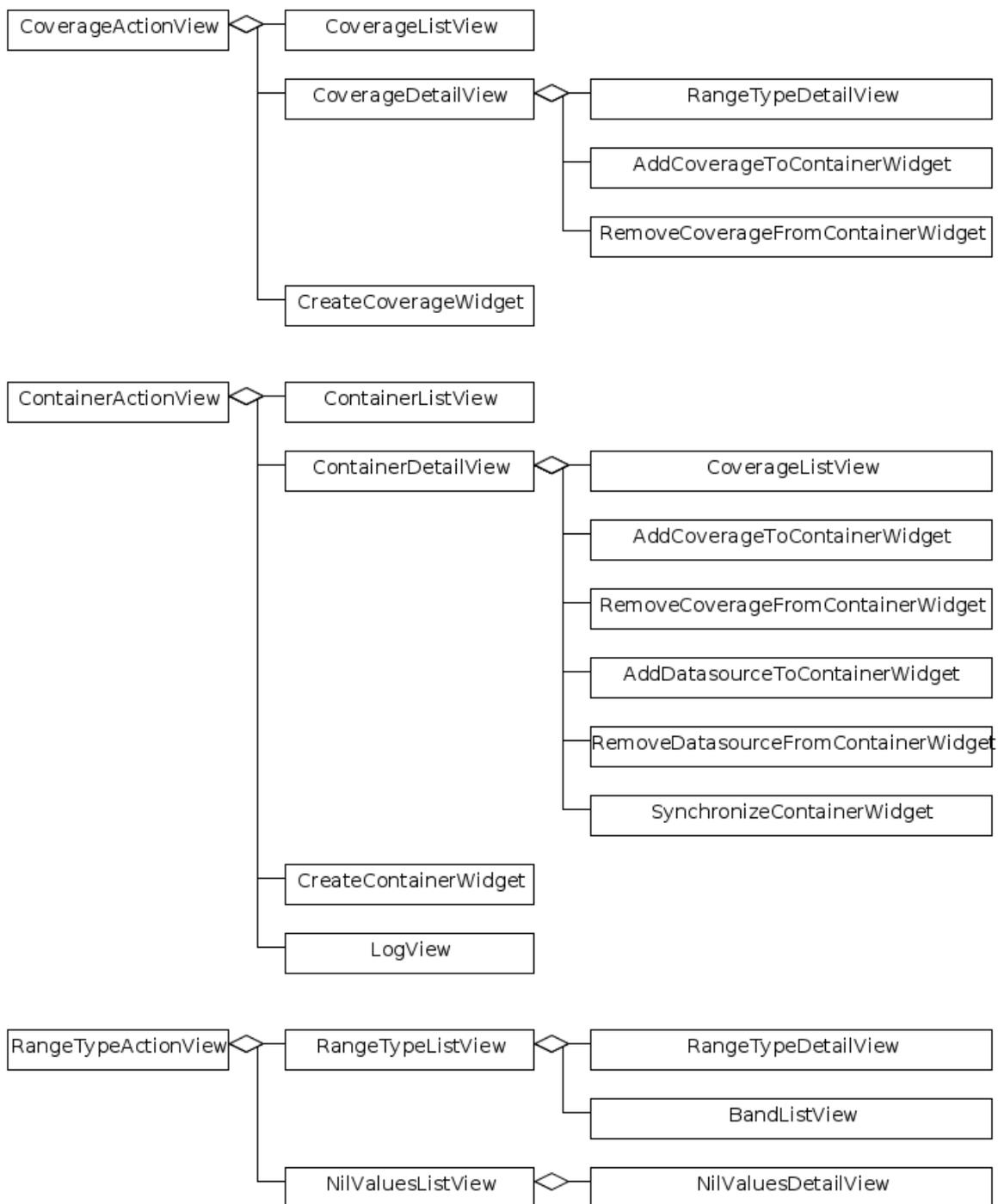


Fig. 3.9: The client views/widget hierarchy.

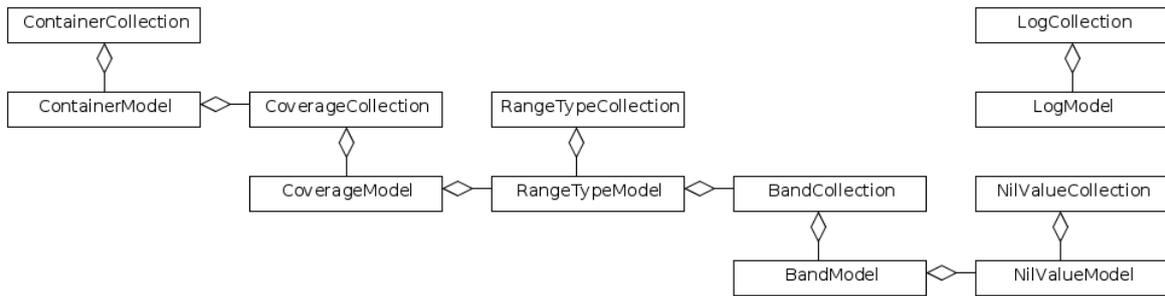


Fig. 3.10: *The models/collection hierarchy on the client.*

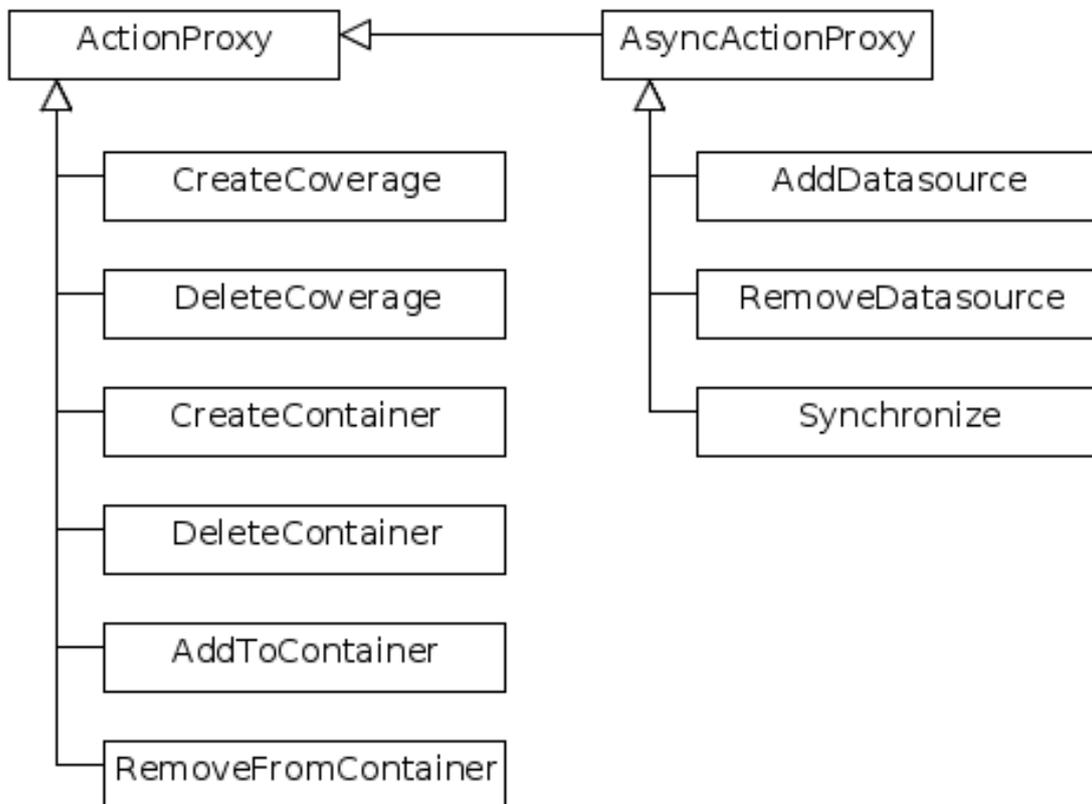


Fig. 3.11: *The action proxies used on the client.*

To provide the RPC interface, there are two possibilities. The first is a wrapped setup of the [SimpleXMLRPC-Server module](#)²³⁵, which would represent an abstraction of the XML to the actual entailed data and the dispatching of registered functions. As the module is already included in the standard library of recent Python versions, this approach would not impose an additional dependency. A drawback is the missing user authorization, which has to be implemented manually. Also, this method is only suitable for XML-RPC, which is more verbose than its JSON counterpart.

The second option would be to use a django extension framework, e.g [rpc4django](#)²³⁶. This framework eases the setup of RPC enabled functions, provides user authorization and is agnostic to the RPC protocol used (either JSON- or XML-RPC). This library also uses the BSD license.

On the client side, several JavaScript libraries are required. For DOM manipulation and several utility functions [jQuery](#)²³⁷ and [jQueryUI](#)²³⁸ are used. The libraries are licensed under the GPL and MIT licenses.

As a general utility library and dependency for other module comes [Underscore](#)²³⁹. To implement a working MVC layout, [Backbone](#)²⁴⁰ is suggested. This library also abstracts the use of REST resources. Both libraries are distributed under the MIT license.

For calling RPC functions and parsing the output, the library [rpc.js](#)²⁴¹ is required. It adheres to either the JSON-RPC or the XML-RPC protocol. The library is dual-licensed under the MIT and the GPL license.

To display larger amounts of objects and to efficiently manipulate them, the [SlickGrid](#)²⁴² and its integration with Backbone, [Slickback](#)²⁴³ are used. The two libraries are both licensed under the MIT license.

For easy management of javascript files in conjunction with other resources the [requirejs](#)²⁴⁴ framework is included. It provides means to modularize javascript code and resolve dependencies. The toolset also includes an optimizer which merges and minimizes all modules into a single javascript file with no changes to the client code. The framework is published under both MIT and BSD license.

To avoid incompatibilities and third party server dependencies, all javascript libraries will be served from the EOxServer static files. This implies that for the operator client-side libraries no additional software needs to be installed as EOxServer ships with all requirements.

On the server-side the two packages [rpc4django](#) and [djangoRESTframework](#) need to be installed for the operator to function. As both libraries can be found on the Python Package Index (PyPI) the installation procedure using *pip* is straightforward when both dependencies are added to the EOxServer *setup.py*.

When EOxServer is installed using another technique than *pip* (like using the RPM or Debian packages), the libraries will likely have to be installed manually. For this reason they have to be listed in the dependencies page in the user manual aswell.

Dependency	Cat.	License	Purpose
Django REST Framework	Server	BSD	Expose server data via REST
RPC 4 Django	Server	BSD	Expose server methods via RPC
jQuery	Client	GPL/MIT	DOM Manipulation / AJAX Client
UnderscoreJS	Client	MIT	General Javascript utilities
BackboneJS	Client	MIT	MVC Framework, REST abstraction
json-xml-rpc	Client	GPL/MIT	RPC client
SlickGrid	Client	MIT	Data Grid widget implementation
Slickback	Client	MIT	SlickGrid to Backbone bridge
requirejs	Client	MIT/BSD	Modularization and optimization

²³⁵ <http://docs.python.org/library/simplexmlrpcserver.html>

²³⁶ <http://davidfischer.name/rpc4django/>

²³⁷ <http://jquery.com/>

²³⁸ <http://jqueryui.com/>

²³⁹ <http://underscorejs.org/>

²⁴⁰ <http://backbonejs.org/>

²⁴¹ <https://github.com/westonruter/json-xml-rpc>

²⁴² <https://github.com/mleibman/SlickGrid>

²⁴³ <https://github.com/teleological/slickback>

²⁴⁴ <http://requirejs.org/>

Voting History

N/A

Traceability

Requirements N/A

Tickets <http://eoxserver.org/ticket/4>

RFC 19: Migrate project repository from svn to git

Author Marko Locher

Created 2013-04-05

Last Edit \$Date\$

Status ACCEPTED

Discussion n/a

Migrating from Subversion to git and in the process also switch from Trac to github.

(Credit: Inspired by MapServer's RFC 84 at: <http://mapserver.org/development/rfc/ms-rfc-84.html>)

Introduction

While svn suits our needs as a collaborative source code version management system, it has shortcomings that make it difficult to work with for developers working on multiple tasks in parallel. Git's easy branching makes it possible to set up branches for individual task, isolating code changes from other branches, thus making the switch from one task to another possible without the risk of losing or erroneously committing work-in-progress code. Three-way merging of different branches means that merging code from one branch to another becomes a rapid task, by only having to deal with actual conflicts in the code. Offline committing and access to entire history make working offline possible.

There is already somewhat of a consensus that the migration from svn to git is a good move. Discussion remains as to how this transition should be performed. This RFC outlines the different options available for hosting the official repository, and the different options available for our ticket tracking.

Current investigation has retained two major options that we could go down with:

- Repository migrated to github, use github provided issue tracking. This option will be referred to as "Github hosting".
- Repository hosted by EOX, current trac instance migrated to hook on the new repository. This option will be referred to as "EOX hosting"

Github hosting

This option consists in moving our entire code+ticket infrastructure to github. The current trac instance becomes nearly read-only, new tickets cannot be created on it. Existing tickets are migrated to github with a script taking a trac postgresql dump (once the migration starts, our trac instance becomes read-only).

Advantages

- Code hosting:
- No need to worry about hosting infrastructure
- Can be up and running with a short delay

- Support for pull requests, allowing external contributions to be rapidly merged into our repository
- Online code editing for quick fixups
- Github visualization tools, for example to check which branches are likely to contain conflicting code sections
- Code and patch commenting make collaboratively working on a given feature very lightweight, i.e. just add your comment on the code line which seems problematic to you
- Documentation contributions highly simplified for one-shot contributions
- Issue tracking:
 - Integration of ticket state with commit messages (e.g: “fix mem allocation in mapDraw(), closes issue #1234
 - Email replies to ticket notifications
 - The free-form label tagging of issues might open up some interesting usages
 - Versioned text-base attachments (gists), with commenting

Inconveniences

- Hosting by a private company, which might become an issue if their TOS evolve or if they go out of business. The source code availability is not an issue as is possible to maintain a mirror on any server, and each developer has a checkout of the full source control history. Ticket migration would be an issue, but there are APIs available to extract existing tickets.
- Issue tracker is in some ways less feature full than trac. The only hard coded attributes are the assignee and the milestone. All the other triaging information goes into free formed labels, a la gmail.
- No way to automatically assign a ticket owner given a component
- No support for image attachments, can be referenced by url but must be hosted elsewhere.
- No support for private security tickets
- Administering committer access will be done through github, old credentials do not apply. Git does not support fine-grained commit permissions per directory, there will be a separate repository for the docs to account for the larger number of committers there.

Git Workflows

Stable Branches

This document outlines a workflow for fixing bugs in our stable branches: http://www.net-snmp.org/wiki/index.php/Git_Workflow I believe it is a very good match for our stable release management:

- pick the oldest branch where the fix should be applied
- commit the fix to this oldest branch
- merge the old branch down to all the more recent ones, including master

Release Management

Instead of freezing development during our beta cycle, a new release branch is created once the feature freeze is decided, and our betas, releases and subsequent bugfix releases are tagged off of this branch. Bug fixes are committed to this new stable branch, and merged into master. New features can continue to be added to master during all the beta phase. <http://nvie.com/posts/a-successful-git-branching-model/> is an interesting read even if it does not fit our stable release branches exactly.

Upgrade path for svn users

For those users who do not wish to change their workflow and continue with svn commands. This is not the recommended way to work with git, as local or remote changes might end up in having conflicts to resolve, like with svn.

Checkout the project

```
git clone git@github.com:EOX-A/eoxserver
```

Update

```
git pull origin master
```

Commit changes

```
git add [list of files]
git commit -m "Commit message"
git push origin master
```

Fix a bug in a branch, and merge the fix into master

```
git checkout feature-branch
git add [list of files]
git commit -m "Commit message"
git push origin feature-branch
git checkout master
git merge feature-branch
git push origin master
```

Tasks

- import svn to git
- assign github users
- split into sub-projects:
 - eoxserver
 - autotest
 - docs
 - soap_proxy
 - document release process
 - migrate website scripts
 - switch trac site to read-only

Voting History

Motion Adopted on 2013-05-15 with +1 from Stephan Meißl, Fabian Schindler, and Martin Paces

Traceability

Requirements N/A

Tickets N/A

Release notes from various versions of EOxServer.

EOxServer 0.3.1

- Migrated to GitHub.
- Added Vagrant configuration
- Fixing several bugs.
- Updated build process by adding support for usage of a custom GDAL transformer needed for ENVISAT data having a big number of GCPs.

EOxServer 0.3.2

- Switched to EOx Maps layers for background and new overlay in WebClient and Admin
- Added documentation as submodule for readthedocs.org
- Adjusting `check_method_and_order()` in reftools
- Improved transformer suggestion for 'vertical-outlines' tie-points' set as used in ngeo-b
- Actually raising `RuntimeErrors` in check of geographic metadata
- Reproject flipped images even if projections are the same in preprocessing

EOxServer 0.4

This major release introduced a lot of new features since the last stable version and included a major restructuring of many of EOxServer internals.

New Data Models

The 0.4 release overhauled the previous data models to provide a more efficient, flexible and performant way to query and insert data.

More important is that the introduction of the new data models made the *Data Integration Layer* obsolete. Only Django's `QuerySet`²⁴⁵ are necessary for all data model related tasks. Especially for large datasets this mechanism improves the overall performance drastically.

The new backends data models provide a more flexible approach for additional data sources and packages that can be realised using the *New Plugin System* (page 226).

New Plugin System

The new plugin system was introduced to make the extension of functionality easier, more efficient and less error prone. For this reason `trac's plugin system`²⁴⁶ was copied and added to the EOxServer source tree.

The configuration of the plugins are not done in the `settings.py` file instead of the database.

Miscellaneous Internal Improvements

Various internal APIs have been revised and improved.

Decoders

A new API for decoding config files, XML files and KVP requests has been established. It has a large spectrum of functionality and allows to parse requests to actual Python types with proper validity checking.

Backends

A new backend data retrieval and cache system was implemented. This goes inline with the new data models and plugin system to easily extend the existing storage possibilities.

XML Encoding

A new XML encoding mechanism on top of `lxml`²⁴⁷ was implemented which is an order of magnitude faster than the previous `dom`²⁴⁸ based solution.

Management Commands

All management commands have been revisited and streamlined to their respective core functionality.

For convenience there now is a bulk ingestion command to allow a fast way to register a large number of datasets with a prepared CSV file.

Service Improvements

Also on the outward side of EOxServers capabilities a lot has been achieved. The service layer makes extensive use of the new Plugin system which makes it easy to add new services, renderers, connectors and whatever else is required.

²⁴⁵ <https://docs.djangoproject.com/en/dev/ref/models/querysets/>

²⁴⁶ <http://trac.edgewall.org/wiki/TracDev/ComponentArchitecture>

²⁴⁷ <http://lxml.de/>

²⁴⁸ <https://docs.python.org/2/library/xml.dom.html>

WCS 2.0

EOxServer now fully supports the following WCS 2.0 service extensions:

- [Scaling Extension](#)²⁴⁹
- [Interpolation Extension](#)²⁵⁰
- [RangeSubsetting Extension](#)²⁵¹
- [CRS Extension](#)²⁵²
- [GeoTIFF Encoding Extension](#)²⁵³

WMS (all versions)

The WMS rendering was rewritten from scratch to allow various additional layer types, input data and storage forms.

WMS mask layers allow the visualization of various mask types (clouds, snow, low quality or the like) either in a colorized manner or as a cutout of the original raster.

WPS 1.0

EOxServer now supports synchronus processes invocation via the WPS 1.0 protocol. Processes are components that are easily written and plugged into any EOxServer instance.

Webclient

The existing webclient was replaced by a custom build of [EOxClient](#)²⁵⁴. It allows the inspection of more than one collection or dataset and features a dynamic timeline to ease the visual inspection of large datasets.

²⁴⁹ <https://portal.opengeospatial.org/files/12-039>

²⁵⁰ <https://portal.opengeospatial.org/files/12-049>

²⁵¹ <https://portal.opengeospatial.org/files/12-040>

²⁵² <https://portal.opengeospatial.org/files/11-053>

²⁵³ https://portal.opengeospatial.org/files/?artifact_id=54813

²⁵⁴ <https://github.com/EOX-A/EOxClient>

Subpackages

`eooserver.backends` package

Subpackages

`eooserver.backends.packages` package

Submodules

`eooserver.backends.packages.tar` module

`eooserver.backends.packages.zip` module

Module contents

`eooserver.backends.storages` package

Submodules

`eooserver.backends.storages.ftp` module

`eooserver.backends.storages.http` module

`eooserver.backends.storages.local` module

`eooserver.backends.storages.rasdaman` module

Module contents

Submodules

eoxserver.backends.access module

eoxserver.backends.cache module

eoxserver.backends.component module

eoxserver.backends.config module

eoxserver.backends.interfaces module

class `eoxserver.backends.interfaces.AbstractStorageInterface`

Bases: `object`²⁵⁵

name

Name of the storage implementation.

validate (*url*)

Validates the given storage locator and raises a `django.core.exceptions.ValidationError`²⁵⁶ if errors occurred.

class `eoxserver.backends.interfaces.ConnectedStorageInterface`

Bases: `eoxserver.backends.interfaces.AbstractStorageInterface` (page 230)

Interface for storages that do not store “files” but provide access to data in a different fashion.

connect (*url*, *location*)

Return a connection string for a remote dataset residing on a storage specified by the given *url* and *location*.

Parameters

- **url** – the URL denoting the storage itself
- **location** – the location of the file to retrieve on the storage

Returns a connection string to open the stream to actually retrieve data

class `eoxserver.backends.interfaces.FileStorageInterface`

Bases: `eoxserver.backends.interfaces.AbstractStorageInterface` (page 230)

Interface for storages that provide access to files and allow the retrieval of those.

list_files (*url*, *location*)

Return a list of retrievable files available on the storage located at the specified URL and given location.

Parameters

- **url** – the URL denoting the storage itself
- **location** – a template to find items on the storage

Returns an iterable of the storage contents under the specified *location*

retrieve (*url*, *location*, *path*)

Retrieve a remote file from the storage specified by the given *url* and location and store it to the given *path*. Storages that don’t need to actually retrieve and store files, just need to return a path to a local file instead of storing it under *path*.

Parameters

- **url** – the URL denoting the storage itself
- **location** – the location of the file to retrieve on the storage

²⁵⁵ <https://docs.python.org/2.7/library/functions.html#object>

²⁵⁶ <https://docs.djangoproject.com/en/1.8/ref/exceptions/#django.core.exceptions.ValidationError>

- **path** – a local path where the file should be saved under; this is used as a *hint*

Returns the actual path where the file was stored; in some cases this can be different than the passed `path`

class `eooserver.backends.interfaces.PackageInterface`

Bases: `object`²⁵⁷

extract (*package_filename, location, path*)

Extract a file specified by the `location` from the package to the given `path` specification.

Parameters

- **package_filename** – the local filename of the package
- **location** – a location *within* the package to be extracted
- **path** – a local path where the file should be saved under; this is used as a *hint*

Returns the actual path where the file was stored; in some cases this can be different than the passed `path`

list_contents (*package_filename, location_regex=None*)

Return a list of item locations under the specified location in the given package.

Parameters

- **package_filename** – the local filename of the package
- **location_regex** – a template to find items within the package

Returns an iterable of the package contents under the specified `location`

name

Name of the package implementation.

eooserver.backends.middleware module

eooserver.backends.models module

eooserver.backends.testbase module

Module contents

eooserver.contrib package

Submodules

eooserver.contrib.gdal module

This module imports and initializes GDAL; i.e enables exceptions and registers all available drivers.

eooserver.contrib.gdal_array module

eooserver.contrib.mapserver module

eooserver.contrib.ogr module

²⁵⁷ <https://docs.python.org/2.7/library/functions.html#object>

eoxserver.contrib.osr module

class `eoxserver.contrib.osr.SpatialReference` (*raw=None, format=None*)

Bases: `object`²⁵⁸

Extension to the original SpatialReference class.

IsSame (*other*)

proj

srid

Convenience function that tries to get the SRID of the projection.

swap_axes

url

wkt

xml

eoxserver.contrib.vrt module

class `eoxserver.contrib.vrt.VRTBuilder` (*size_x, size_y, num_bands=0, data_type=None, vrt_filename=None*)

Bases: `object`²⁵⁹

This class is a helper to easily create VRT datasets from various sources.

Parameters

- **size_x** – the pixel size of the X dimension
- **size_y** – the pixel size of the Y dimension
- **num_bands** – the initial number of bands; bands can be added afterwards
- **data_type** – the GDT data type identifier
- **vrt_filename** – a path the filename shall be stored at; if none is specified the dataset will only be kept in memory

add_band (*data_type=None, options=None, nodata=None*)

Add a band to the VRT Dataset.

Parameters

- **data_type** – the data type of the band to add. if omitted this is determined automatically by GDAL
- **options** – a list of any string options to be supplied to the new band

add_simple_source (*band_index, src, src_band, src_rect=None, dst_rect=None*)

Add a new simple source to the VRT.

Parameters

- **band_index** – the band index the source shall contribute to
- **src** – either a GDAL Dataset or a file path to the source dataset
- **src_band** – specify which band of the source dataset shall contribute to the target VRT band
- **src_rect** – a 4-tuple of integers in the form (offset-x, offset-y, size-x, size-y) or a Rect specifying the source area to contribute

²⁵⁸ <https://docs.python.org/2.7/library/functions.html#object>

²⁵⁹ <https://docs.python.org/2.7/library/functions.html#object>

- **dst_rect** – a 4-tuple of integers in the form (offset-x, offset-y, size-x, size-y) or a `Rect` specifying the target area to contribute

copy_gcps (*ds*, *offset=None*)

Copy the GCPs from the given GDAL `Dataset`, optionally offsetting them

Parameters

- **ds** – a GDAL `Dataset`
- **offset** – a 2-tuple of integers; the pixel offset to be applied to any GCP copied

copy_metadata (*ds*)

Copy the metadata fields and values from the given dataset.

Parameters **ds** – a GDAL `Dataset`

dataset

Returns a handle to the underlying VRT GDAL `Dataset`.

classmethod from_dataset (*ds*, *vrt_filename=None*)

A helper function to create a VRT dataset from a given template dataset.

Parameters **ds** – a GDAL `Dataset`

class `eoxserver.contrib.vrt.VRTBuilder2` (*size_x*, *size_y*, *num_bands=0*, *data_type=None*,
vrt_filename=None)

Bases: `object`²⁶⁰

add_band (*data_type=None*, *options=None*, *nodata=None*)

add_simple_source (*band_index*, *src*, *src_band*, *src_rect=None*, *dst_rect=None*)

Add a new simple source to the VRT.

Parameters

- **band_index** – the band index the source shall contribute to
- **src** – either a GDAL `Dataset` or a file path to the source dataset
- **src_band** – specify which band of the source dataset shall contribute to the target VRT band
- **src_rect** – a 4-tuple of integers in the form (offset-x, offset-y, size-x, size-y) or a `Rect` specifying the source area to contribute
- **dst_rect** – a 4-tuple of integers in the form (offset-x, offset-y, size-x, size-y) or a `Rect` specifying the target area to contribute

build ()

build_sources (*sources*)

set_geotransform (*geotransform*)

warped_gcps (*gcp_dsc*, *resample='NearestNeighbour'*, *order=0*)

`eoxserver.contrib.vrt.get_vrt_driver` ()

Convenience function to get the VRT driver.

eoxserver.contrib.vsi module

This module provides Python file-object like access to VSI files.

class `eoxserver.contrib.vsi.TemporaryVSIFile` (*filename*, *mode='r'*)

Bases: `eoxserver.contrib.vsi.VSIFile` (page 234)

Subclass of `VSIFile`, that automatically deletes the physical file upon deletion.

²⁶⁰ <https://docs.python.org/2.7/library/functions.html#object>

close ()

Close the file. This also deletes it.

classmethod from_buffer (*buf, mode='w', filename=None*)

Creates a *TemporaryVSIFile* (page 233) from a string.

Parameters

- **buf** – the supplied string
- **mode** – the file opening mode
- **filename** – the optional filename the file shall be stored under; by default this is an in-memory location

class `eoxserver.contrib.vsi.VSIFile` (*filename, mode='r'*)

Bases: `object`²⁶¹

File-like object interface for VSI file API.

Parameters

- **filename** – the path to the file; this might also be any VSI special path like “/vsi-curl/...” or “/vsizip/...”. See the [GDAL documentation](#)²⁶² for reference.
- **mode** – the file opening mode

close ()

Close the file.

closed

Return a boolean value to indicate whether or not the file is already closed.

filename

Returns the filename referenced by this file

read (*size=None*)

Read from the file. If no *size* is specified, read until the end of the file.

Parameters **size** – the number of bytes to be read

Returns the bytes read as a string

seek (*offset, whence=0*)

Set the new read/write offset in the file.

Parameters

- **offset** – the new offset
- **whence** – how the offset shall be interpreted; possible options are `os.SEEK_SET`, `os.SEEK_CUR` and `os.SEEK_END`

size

Return the size of the file in bytes

tell ()

Return the current read/write offset of the file.

Returns an integer offset

write (*data*)

Write the buffer *data* to the file.

Parameters **data** – the string buffer to be written

`eoxserver.contrib.vsi.open` (*filename, mode='r'*)

A function mimicking the builtin function `open` but returning a *VSIFile* (page 234) instead.

²⁶¹ <https://docs.python.org/2.7/library/functions.html#object>

²⁶² <http://trac.osgeo.org/gdal/wiki/UserDocs/ReadInZip>

Parameters

- **filename** – the path to the file; this might also be any VSI special path like “/vsi-curl/...” or “/vsizip/...”. See the [GDAL documentation](#)²⁶³ for reference.
- **mode** – the file opening mode

Returns a *VSIFile* (page 234)

Module contents

This package provides a common interface to contributing third party libraries that need some special care when importing or are provided with additional features.

eoxserver.core package**Subpackages**

eoxserver.core.decoders package

Submodules

eoxserver.core.decoders.base module

eoxserver.core.decoders.config module

eoxserver.core.decoders.kvp module

eoxserver.core.decoders.xml module

Module contents

eoxserver.core.util package

Submodules

eoxserver.core.util.functools module

eoxserver.core.util.geotools module

eoxserver.core.util.importtools module

eoxserver.core.util.iteratortools module

eoxserver.core.util.multiparttools module

eoxserver.core.util.perftools module

eoxserver.core.util.rect module

²⁶³ <http://trac.osgeo.org/gdal/wiki/UserDocs/ReadInZip>

`eoxserver.core.util.timetools` module

`eoxserver.core.util.xmltools` module

Module contents

Submodules

`eoxserver.core.component` module

`eoxserver.core.config` module

`eoxserver.core.management` module

`eoxserver.core.models` module

`eoxserver.core.views` module

Module contents

`eoxserver.processing` package

Subpackages

`eoxserver.processing.gdal` package

Submodules

`eoxserver.processing.gdal.reftools` module

`eoxserver.processing.gdal.vrt` module

`eoxserver.processing.gdal.vrt.create_simple_vrt` (*ds*, *vrt_filename*)

Module contents

`eoxserver.processing.preprocessing` package

Submodules

`eoxserver.processing.preprocessing.exceptions` module

`eoxserver.processing.preprocessing.format` module

`eoxserver.processing.preprocessing.georeference` module

`eoxserver.processing.preprocessing.optimization` module

`eoxserver.processing.preprocessing.util` module

Module contents

Submodules

Module contents

eoxserver.resources package

Subpackages

eoxserver.resources.coverages package

Subpackages

eoxserver.resources.coverages.metadata package

Subpackages

eoxserver.resources.coverages.metadata.formats package

Submodules

eoxserver.resources.coverages.metadata.formats.dimap_general module

eoxserver.resources.coverages.metadata.formats.eoom module

eoxserver.resources.coverages.metadata.formats.gdal_dataset module

eoxserver.resources.coverages.metadata.formats.gdal_dataset_envisat module

eoxserver.resources.coverages.metadata.formats.inspire module

eoxserver.resources.coverages.metadata.formats.native module

Module contents

Submodules

eoxserver.resources.coverages.metadata.component module

eoxserver.resources.coverages.metadata.interfaces module

class `eoxserver.resources.coverages.metadata.interfaces.GDALDatasetMetadataReaderInterface`

Bases: `object`²⁶⁴

Interface for GDAL dataset metadata readers.

format (*obj*)

Returns a format specifier for the given object. Can be ignored, when the reader only supports one format.

read_ds (*ds*)

Returns a dict with any of the following keys: - identifier (string) - extent (a four tuple of floats) - size

²⁶⁴ <https://docs.python.org/2.7/library/functions.html#object>

(a two-tuple of ints) - projection (an integer or two-tuple of two strings (definition and format)) - footprint (a `django.contrib.gis.geos.MultiPolygon`) - begin_time (a python `datetime.datetime`) - end_time (a python `datetime.datetime`)

The argument *ds* is a `gdal.Dataset`.

test_ds (*obj*)

Return a boolean value, whether or not metadata can be extracted from the given object.

class `eoxserver.resources.coverages.metadata.interfaces.MetadataReaderInterface`
Bases: `object`²⁶⁵

Interface for metadata readers.

format (*obj*)

Returns a format specifier for the given object. Can be ignored, when the reader only supports one format.

read (*obj*)

Returns a dict with any of the following keys: - identifier (string) - extent (a four tuple of floats) - size (a two-tuple of ints) - projection (an integer or two-tuple of two strings (definition and format)) - footprint (a `django.contrib.gis.geos.MultiPolygon`) - begin_time (a python `datetime.datetime`) - end_time (a python `datetime.datetime`)

The argument *obj* is of an arbitrary type, the reader needs to determine whether or not the type is supported and an exception shall be raised if not.

test (*obj*)

Return a boolean value, whether or not metadata can be extracted from the given object.

class `eoxserver.resources.coverages.metadata.interfaces.MetadataWriterInterface`
Bases: `object`²⁶⁶

Interface for metadata writers.

formats

write (*values*, *file_obj*, *format=None*)

Write the given values (a dict) to the file-like object *file_obj*. The dict contains all of the following entries: - identifier (string) - extent (a four tuple of floats) - size (a two-tuple of ints) - projection (an integer or two-tuple of two strings (definition and format)) - footprint (a `django.contrib.gis.geos.MultiPolygon`) - begin_time (a python `datetime.datetime`) - end_time (a python `datetime.datetime`)

The writer may ignore non-applicable parameters.

Module contents

Submodules

[eoxserver.resources.coverages.crss module](#)

[eoxserver.resources.coverages.dateline module](#)

[eoxserver.resources.coverages.formats module](#)

[eoxserver.resources.coverages.models module](#)

[eoxserver.resources.coverages.rangetype module](#)

²⁶⁵ <https://docs.python.org/2.7/library/functions.html#object>

²⁶⁶ <https://docs.python.org/2.7/library/functions.html#object>

eoxserver.resources.coverages.util module

Module contents

eoxserver.resources.processes package

Submodules

eoxserver.resources.processes.admin module

eoxserver.resources.processes.models module

eoxserver.resources.processes.tracker module

eoxserver.resources.processes.views module

Module contents

Module contents

eoxserver.services package

Subpackages

eoxserver.services.auth package

Submodules

eoxserver.services.auth.base module

eoxserver.services.auth.charonpdp module

eoxserver.services.auth.dummyspdp module

eoxserver.services.auth.exceptions module

exception `eoxserver.services.auth.exceptions.AuthorizationException`

Bases: `exceptions.Exception`²⁶⁷

`code = 'AccessForbidden'`

eoxserver.services.auth.interfaces module

class `eoxserver.services.auth.interfaces.PolicyDecisionPointInterface`

Bases: `object`²⁶⁸

This is the interface for Policy Decision Point (PDP) implementations.

authorize (*request*)

This method takes an `OWSRequest` object as input and returns an `AuthorizationResponse`

²⁶⁷ <https://docs.python.org/2.7/library/exceptions.html#exceptions.Exception>

²⁶⁸ <https://docs.python.org/2.7/library/functions.html#object>

instance. It is expected to check if the authenticated user (if any) is authorized to access the requested resource and set the `authorized` flag of the response accordingly.

In case the user is not authorized, the content and status of the response shall be filled with an error message and the appropriate HTTP Status Code (403).

The method shall not raise any exceptions.

`pdp_type`

The type name of this PDP.

`eoxserver.services.auth.middleware` module

Module contents

`eoxserver.services.gdal` package

Subpackages

`eoxserver.services.gdal.wcs` package

Submodules

`eoxserver.services.gdal.wcs.referenceable_dataset_renderer` module

Module contents

Module contents

`eoxserver.services.gml` package

Subpackages

`eoxserver.services.gml.v32` package

Submodules

`eoxserver.services.gml.v32.encoders` module

Module contents

Module contents

`eoxserver.services.mapserver` package

Subpackages

`eoxserver.services.mapserver.connectors` package

Submodules

`eoxserver.services.mapserver.connectors.multifile_connector` module

`eoxserver.services.mapserver.connectors.polygonmask_connector` module

`eoxserver.services.mapserver.connectors.simple_connector` module

`eoxserver.services.mapserver.connectors.tileindex_connector` module

Module contents

`eoxserver.services.mapserver.wcs` package

Submodules

`eoxserver.services.mapserver.wcs.base_renderer` module

`eoxserver.services.mapserver.wcs.capabilities_renderer` module

`eoxserver.services.mapserver.wcs.coverage_description_renderer` module

`eoxserver.services.mapserver.wcs.coverage_renderer` module

Module contents

`eoxserver.services.mapserver.wms` package

Subpackages

`eoxserver.services.mapserver.wms.layerfactories` package

Submodules

`eoxserver.services.mapserver.wms.layerfactories.base` module

`eoxserver.services.mapserver.wms.layerfactories.colorized_mask_layer_factory` module

`eoxserver.services.mapserver.wms.layerfactories.coverage_bands_layer_factory` module

`eoxserver.services.mapserver.wms.layerfactories.coverage_layer_factory` module

`eoxserver.services.mapserver.wms.layerfactories.coverage_mask_layer_factory` module

`eoxserver.services.mapserver.wms.layerfactories.coverage_masked_outlines_layer_factory` module

`eoxserver.services.mapserver.wms.layerfactories.coverage_outlines_layer_factory` module

Module contents

eoxserver.services.mapserver.wms.styleapplicators package

Submodules

eoxserver.services.mapserver.wms.styleapplicators.sld module

Module contents

Submodules

eoxserver.services.mapserver.wms.capabilities_renderer module

eoxserver.services.mapserver.wms.feature_info_renderer module

eoxserver.services.mapserver.wms.legendgraphic_renderer module

eoxserver.services.mapserver.wms.map_renderer module

eoxserver.services.mapserver.wms.util module

Module contents

Submodules

eoxserver.services.mapserver.interfaces module

class eoxserver.services.mapserver.interfaces.ConnectorInterface

Bases: `object`²⁶⁹

Interface for connectors between *mapscript.layerObj* and associated data.

connect (*coverage, data_items, layer, options*)

Connect a layer (a *mapscript.layerObj*) with the given data items and coverage (a list of two-tuples: location and semantic).

disconnect (*coverage, data_items, layer, options*)

Performs all necessary cleanup operations.

supports (*data_items*)

Returns *True* if the given *data_items* are supported and *False* if not.

class eoxserver.services.mapserver.interfaces.LayerFactoryInterface

Bases: `object`²⁷⁰

Interface for factories that create *mapscript.layerObj* objects for coverages.

generate (*eo_object, group_layer, options*)

Returns an iterable of *mapscript.layerObj* objects preconfigured for the given EO object. This is easily done via the *yield* statement.

generate_group (*name*)

Returns a 'group layer' to be referenced by all other layers generated by this factory.

requires_connection

Return whether or layers generated by this factory require to be connected via a layer connector.

²⁶⁹ <https://docs.python.org/2.7/library/functions.html#object>

²⁷⁰ <https://docs.python.org/2.7/library/functions.html#object>

suffixes

The suffixes associated with layers this factory produces. This is used for “specialized” layers such as “bands” or “outlines” layers. For factories that don’t use this feature, it can be left out.

class `eoxserver.services.mapserver.interfaces.StyleApplicatorInterface`

Bases: `object`²⁷¹

Interface for style applicators.

apply (*coverage, data_items, layer*)

Apply all relevant styles.

Module contents

eoxserver.services.native package

Subpackages

eoxserver.services.native.wcs package

Submodules

eoxserver.services.native.wcs.capabilities_renderer module

eoxserver.services.native.wcs.coverage_description_renderer module

Module contents**Module contents**

eoxserver.services.ows package

Subpackages

eoxserver.services.ows.common package

Subpackages

eoxserver.services.ows.common.v11 package

Submodules

eoxserver.services.ows.common.v11.encoders module

Module contents

eoxserver.services.ows.common.v20 package

Submodules

²⁷¹ <https://docs.python.org/2.7/library/functions.html#object>

`eoxserver.services.ows.common.v20.encoders` module

`eoxserver.services.ows.common.v20.exceptionhandler` module

Module contents

Submodules

`eoxserver.services.ows.common.config` module

Module contents

`eoxserver.services.ows.wcs` package

Subpackages

`eoxserver.services.ows.wcs.v10` package

Submodules

`eoxserver.services.ows.wcs.v10.describecoverage` module

`eoxserver.services.ows.wcs.v10.exceptionhandler` module

`eoxserver.services.ows.wcs.v10.getcapabilities` module

`eoxserver.services.ows.wcs.v10.getcoverage` module

`eoxserver.services.ows.wcs.v10.parameters` module

`eoxserver.services.ows.wcs.v10.util` module

Module contents

`eoxserver.services.ows.wcs.v11` package

Submodules

`eoxserver.services.ows.wcs.v11.describecoverage` module

`eoxserver.services.ows.wcs.v11.exceptionhandler` module

`eoxserver.services.ows.wcs.v11.getcapabilities` module

`eoxserver.services.ows.wcs.v11.getcoverage` module

`eoxserver.services.ows.wcs.v11.parameters` module

`eoxserver.services.ows.wcs.v11.util` module

Module contents

`eoxserver.services.ows.wcs.v20` package

Subpackages

`eoxserver.services.ows.wcs.v20.encodings` package

Submodules

`eoxserver.services.ows.wcs.v20.encodings.geotiff` module

Module contents

`eoxserver.services.ows.wcs.v20.packages` package

Submodules

`eoxserver.services.ows.wcs.v20.packages.tar` module

`eoxserver.services.ows.wcs.v20.packages.zip` module

Module contents

Submodules

`eoxserver.services.ows.wcs.v20.describecoverage` module

`eoxserver.services.ows.wcs.v20.describeecoverageset` module

`eoxserver.services.ows.wcs.v20.encoders` module

`eoxserver.services.ows.wcs.v20.exceptionhandler` module

`eoxserver.services.ows.wcs.v20.getcapabilities` module

`eoxserver.services.ows.wcs.v20.getcoverage` module

`eoxserver.services.ows.wcs.v20.getecoverageset` module

`eoxserver.services.ows.wcs.v20.parameters` module

`eoxserver.services.ows.wcs.v20.util` module

Module contents

Submodules

`eoxserver.services.ows.wcs.basehandlers` module

eoxserver.services.ows.wcs.interfaces module**class** `eoxserver.services.ows.wcs.interfaces.EncodingExtensionInterface`Bases: `object`²⁷²**parse_encoding_params** (*request*)

Return a dict, containing all additional encoding parameters from a given request.

supports (*format, options*)

Return a boolean value, whether or not an encoding extension supports a given format.

class `eoxserver.services.ows.wcs.interfaces.PackageWriterInterface`Bases: `object`²⁷³

Interface for package writers.

add_to_package (*package, file_obj, size, location*)Add the file object to the package, that is returned by the *create_package* method.**cleanup** (*package*)

Perform any necessary cleanups, like closing files, etc.

create_package (*filename, format, params*)Create a package, which the encoder can later add items to with the *cleanup* and *add_to_package* method.**get_file_extension** (*package, format, params*)

Retrieve the file extension for the given package and format specifier.

get_mime_type (*package, format, params*)

Retrieve the output mime type for the given package and/or format specifier.

supports (*format, params*)

Return a boolean value, whether or not a writer supports a given format.

class `eoxserver.services.ows.wcs.interfaces.WCSCapabilitiesRendererInterface`Bases: `object`²⁷⁴

Interface for WCS Capabilities renderers.

render (*params*)

Render the capabilities including information about the given coverages.

supports (*params*)

Returns a boolean value to indicate whether or not the renderer is able to render the capabilities with the given parameters.

class `eoxserver.services.ows.wcs.interfaces.WCSCoverageDescriptionRendererInterface`Bases: `object`²⁷⁵

Interface for coverage description renderers.

render (*params*)

Render the description of the given coverages.

supports (*params*)

Returns a boolean value to indicate whether or not the renderer is able to render the coverage and the given WCS version.

class `eoxserver.services.ows.wcs.interfaces.WCSCoverageRendererInterface`Bases: `object`²⁷⁶

Interface for coverage renderers.

²⁷² <https://docs.python.org/2.7/library/functions.html#object>²⁷³ <https://docs.python.org/2.7/library/functions.html#object>²⁷⁴ <https://docs.python.org/2.7/library/functions.html#object>²⁷⁵ <https://docs.python.org/2.7/library/functions.html#object>²⁷⁶ <https://docs.python.org/2.7/library/functions.html#object>

render (*params*)

Render the coverage with the given parameters.

supports (*params*)

Returns a boolean value to indicate whether or not the renderer is able to render the coverage with the given parameters.

eoxserver.services.ows.wcs.parameters module

class `eoxserver.services.ows.wcs.parameters.CoverageDescriptionRenderParams` (*coverages, version*)
 Bases: `eoxserver.services.ows.wcs.parameters.WCSParamsMixin` (page 247),
`eoxserver.services.parameters.VersionedParams` (page 259)

coverage_ids

coverage_ids_key_name = None

coverages

class `eoxserver.services.ows.wcs.parameters.CoverageRenderParams` (*coverage, version*)
 Bases: `eoxserver.services.ows.wcs.parameters.WCSParamsMixin` (page 247),
`eoxserver.services.parameters.VersionedParams` (page 259)

coverage

coverage_id

coverage_id_key_name = None

class `eoxserver.services.ows.wcs.parameters.WCSCapabilitiesRenderParams` (*coverages, version, sections=None, accept_languages=None, accept_formats=None, update_sequence=None, request=None*)
 Bases: `eoxserver.services.ows.wcs.parameters.WCSParamsMixin` (page 247),
`eoxserver.services.parameters.CapabilitiesRenderParams` (page 259)

class `eoxserver.services.ows.wcs.parameters.WCSParamsMixin`

Bases: `object`²⁷⁷

Module contents

eoxserver.services.ows.wms package

Subpackages

²⁷⁷ <https://docs.python.org/2.7/library/functions.html#object>

`eoxserver.services.ows.wms.v10` package

Submodules

`eoxserver.services.ows.wms.v10.getcapabilities` module

`eoxserver.services.ows.wms.v10.getfeatureinfo` module

`eoxserver.services.ows.wms.v10.getmap` module

Module contents

`eoxserver.services.ows.wms.v11` package

Submodules

`eoxserver.services.ows.wms.v11.getcapabilities` module

`eoxserver.services.ows.wms.v11.getfeatureinfo` module

`eoxserver.services.ows.wms.v11.getmap` module

Module contents

`eoxserver.services.ows.wms.v13` package

Submodules

`eoxserver.services.ows.wms.v13.exceptionhandler` module

`eoxserver.services.ows.wms.v13.getcapabilities` module

`eoxserver.services.ows.wms.v13.getfeatureinfo` module

`eoxserver.services.ows.wms.v13.getlegendgraphic` module

`eoxserver.services.ows.wms.v13.getmap` module

Module contents

Submodules

`eoxserver.services.ows.wms.basehandlers` module

`eoxserver.services.ows.wms.exceptions` module

exception `eoxserver.services.ows.wms.exceptions.InvalidCRS` (*value*,
crs_param_name)

Bases: `exceptions.Exception`²⁷⁸

²⁷⁸ <https://docs.python.org/2.7/library/exceptions.html#exceptions.Exception>

code = 'InvalidCRS'

exception `eoxserver.services.ows.wms.exceptions.InvalidFormat` (*value*)
 Bases: `exceptions.Exception`²⁷⁹

code = 'InvalidFormat'

locator = 'format'

exception `eoxserver.services.ows.wms.exceptions.LayerNotDefined` (*layer*)
 Bases: `exceptions.Exception`²⁸⁰

code = 'LayerNotDefined'

locator = 'layers'

eoxserver.services.ows.wms.interfaces module

class `eoxserver.services.ows.wms.interfaces.WMSCapabilitiesRendererInterface`
 Bases: `object`²⁸¹

Interface for WMS compatible capabilities renderers.

render (*collections, coverages, request_values*)
 Render a capabilities document, containing metadata of the given collections and coverages.

class `eoxserver.services.ows.wms.interfaces.WMSFeatureInfoRendererInterface`
 Bases: `object`²⁸²

Interface for WMS compatible feature info renderers.

render (*layer_groups, request_values, **options*)
 Render the given layer hierarchy with the provided request values and further options.
 options contains relevant options such as specified bands.

suffixes
 Return a list of supported layer suffixes for this renderer.

class `eoxserver.services.ows.wms.interfaces.WMSLegendGraphicRendererInterface`
 Bases: `object`²⁸³

Interface for WMS compatible legend graphic renderers.

render (*collection, eo_object, request_values, **options*)
 Render the given collection and coverage with the provided request values and further options.
 options contains relevant options such as specified bands.

suffixes
 Return a list of supported layer suffixes for this renderer.

class `eoxserver.services.ows.wms.interfaces.WMSMapRendererInterface`
 Bases: `object`²⁸⁴

Interface for WMS compatible map renderers.

render (*layer_groups, request_values, **options*)
 Render the given layer hierarchy with the provided request values and further options.
 options contains relevant options such as specified bands.

²⁷⁹ <https://docs.python.org/2.7/library/exceptions.html#exceptions.Exception>

²⁸⁰ <https://docs.python.org/2.7/library/exceptions.html#exceptions.Exception>

²⁸¹ <https://docs.python.org/2.7/library/functions.html#object>

²⁸² <https://docs.python.org/2.7/library/functions.html#object>

²⁸³ <https://docs.python.org/2.7/library/functions.html#object>

²⁸⁴ <https://docs.python.org/2.7/library/functions.html#object>

suffixes

Return a list of supported layer suffixes for this renderer.

eoxserver.services.ows.wms.util module

Module contents

eoxserver.services.ows.wps package

Subpackages

eoxserver.services.ows.wps.parameters package

Submodules

eoxserver.services.ows.wps.parameters.allowed_values module

eoxserver.services.ows.wps.parameters.base module

eoxserver.services.ows.wps.parameters.bboxdata module

eoxserver.services.ows.wps.parameters.codecs module

eoxserver.services.ows.wps.parameters.complexdata module

eoxserver.services.ows.wps.parameters.crs module

eoxserver.services.ows.wps.parameters.data_types module

eoxserver.services.ows.wps.parameters.formats module

eoxserver.services.ows.wps.parameters.inputs module

eoxserver.services.ows.wps.parameters.literaldata module

eoxserver.services.ows.wps.parameters.response_form module

eoxserver.services.ows.wps.parameters.units module

Module contents

eoxserver.services.ows.wps.processes package

Submodules

eoxserver.services.ows.wps.processes.get_time_data module

Module contents

`eoxserver.services.ows.wps.v10` package

Subpackages

`eoxserver.services.ows.wps.v10.encoders` package

Submodules

`eoxserver.services.ows.wps.v10.encoders.base` module

`eoxserver.services.ows.wps.v10.encoders.capabilities` module

`eoxserver.services.ows.wps.v10.encoders.execute_response` module

`eoxserver.services.ows.wps.v10.encoders.execute_response_raw` module

`eoxserver.services.ows.wps.v10.encoders.parameters` module

`eoxserver.services.ows.wps.v10.encoders.process_description` module

Module contents

Submodules

`eoxserver.services.ows.wps.v10.describeprocess` module

`eoxserver.services.ows.wps.v10.exceptionhandler` module

`eoxserver.services.ows.wps.v10.execute` module

`eoxserver.services.ows.wps.v10.execute_decoder_kv` module

`eoxserver.services.ows.wps.v10.execute_decoder_xml` module

`eoxserver.services.ows.wps.v10.getcapabilities` module

`eoxserver.services.ows.wps.v10.util` module

Module contents

Submodules

`eoxserver.services.ows.wps.exceptions` module

exception `eoxserver.services.ows.wps.exceptions.ExecuteError` (*message*='', *locator*='process.execute()')

Bases: `eoxserver.services.ows.wps.exceptions.NoApplicableCode` (page 252)

exception `eoxserver.services.ows.wps.exceptions.FileSizeExceeded` (*message*, *locator*)

Bases: `eoxserver.services.ows.wps.exceptions.OWS10Exception` (page 252)

exception `eoxserver.services.ows.wps.exceptions.InvalidInputError` (*input_id*)
Bases: `eoxserver.services.ows.wps.exceptions.InvalidParameterValue` (page 252)

exception `eoxserver.services.ows.wps.exceptions.InvalidInputReferenceError` (*input_id*,
message)
Bases: `eoxserver.services.ows.wps.exceptions.InvalidParameterValue` (page 252)

exception `eoxserver.services.ows.wps.exceptions.InvalidInputValueError` (*input_id*,
message)
Bases: `eoxserver.services.ows.wps.exceptions.InvalidParameterValue` (page 252)

exception `eoxserver.services.ows.wps.exceptions.InvalidOutputDefError` (*output_id*,
message)
Bases: `eoxserver.services.ows.wps.exceptions.InvalidParameterValue` (page 252)

exception `eoxserver.services.ows.wps.exceptions.InvalidOutputError` (*output_id*)
Bases: `eoxserver.services.ows.wps.exceptions.InvalidParameterValue` (page 252)

exception `eoxserver.services.ows.wps.exceptions.InvalidOutputValueError` (*output_id*,
message)
Bases: `eoxserver.services.ows.wps.exceptions.NoApplicableCode` (page 252)

exception `eoxserver.services.ows.wps.exceptions.InvalidParameterValue` (*message*,
locator)
Bases: `eoxserver.services.ows.wps.exceptions.OWS10Exception` (page 252)

exception `eoxserver.services.ows.wps.exceptions.MissingParameterValue` (*message*,
locator)
Bases: `eoxserver.services.ows.wps.exceptions.OWS10Exception` (page 252)

exception `eoxserver.services.ows.wps.exceptions.MissingRequiredInputError` (*input_id*)
Bases: `eoxserver.services.ows.wps.exceptions.InvalidParameterValue` (page 252)

exception `eoxserver.services.ows.wps.exceptions.NoApplicableCode` (*message*,
locator)
Bases: `eoxserver.services.ows.wps.exceptions.OWS10Exception` (page 252)

http_status_code = 500

exception `eoxserver.services.ows.wps.exceptions.NoSuchProcessError` (*identifier*)
Bases: `eoxserver.services.ows.wps.exceptions.InvalidParameterValue` (page 252)

exception `eoxserver.services.ows.wps.exceptions.NotEnoughStorage` (*message*)
Bases: `eoxserver.services.ows.wps.exceptions.OWS10Exception` (page 252)

http_status_code = 507

exception `eoxserver.services.ows.wps.exceptions.OWS10Exception` (*code*, *locator*,
message)
Bases: `exceptions.Exception`²⁸⁵
Base OWS 1.0 exception of the WPS 1.0.0 exceptions

http_status_code = 400

exception `eoxserver.services.ows.wps.exceptions.ServerBusy` (*message*)
Bases: `eoxserver.services.ows.wps.exceptions.OWS10Exception` (page 252)

http_status_code = 503

²⁸⁵ <https://docs.python.org/2.7/library/exceptions.html#exceptions.Exception>

exception `eoxserver.services.ows.wps.exceptions.StorageNotSupported` (*message*)

Bases: `eoxserver.services.ows.wps.exceptions.OWS10Exception` (page 252)

exception `eoxserver.services.ows.wps.exceptions.VersionNegotiationFailed` (*message*, *locator*)

Bases: `eoxserver.services.ows.wps.exceptions.OWS10Exception` (page 252)

eoxserver.services.ows.wps.interfaces module

class `eoxserver.services.ows.wps.interfaces.AsyncBackendInterface`

Bases: `object`²⁸⁶

Interface class for an asynchronous WPS back-end. NOTE: Only one asynchronous back-end at time is allowed to be configured.

cancel (*job_id*, ***kwargs*)

Cancel the job execution.

execute (*process*, *raw_inputs*, *resp_form*, *extra_parts=None*, *job_id=None*, *version='1.0.0'*, ***kwargs*)

Execute process asynchronously. The request is defined by the process's identifier `process_id`, `raw_inputs` (before the decoding and resolution of the references), and the `resp_form` (holding the outputs' parameters). The `version` of the WPS standard to be used. Optionally, the user defined `job_id` can be passed. If the `job_id` cannot be used the execute shall fail.

The `extra_parts` should contain a dictionary of named request parts should the request contain multi-part/related CID references.

On success, the method returns the `job_id` assigned to the executed job.

get_response_url (*job_id*)

Get URL of the execute response for the given job id

get_status (*job_id*)

Get status of a job. Allowed responses and their meanings are: ACCEPTED - job scheduled for execution STARTED - job in progress PAUSED - job is stopped and it can be resumed CANCELLED - job was terminated by the user FAILED - job ended with an error SUCCEEDED - job ended successfully

pause (*job_id*, ***kwargs*)

Pause the job execution.

purge (*job_id*, ***kwargs*)

Purge the job from the system by removing all the resources occupied by the job.

resume (*job_id*, ***kwargs*)

Resume the job execution.

supported_versions

A list of versions of the WPS standard supported by the back-end.

class `eoxserver.services.ows.wps.interfaces.ProcessInterface`

Bases: `object`²⁸⁷

Interface class for processes offered, described and executed by the WPS.

asynchronous

Optional boolean flag indicating whether the process can be executed asynchronously. If missing False is assumed.

²⁸⁶ <https://docs.python.org/2.7/library/functions.html#object>

²⁸⁷ <https://docs.python.org/2.7/library/functions.html#object>

description

A human-readable detailed description of the process. Optional. (Content of the the abstract in the WPS process description.)

execute (***kwargs*)

The main execution function for the process. The *kwargs* are the parsed input inputs (using the keys as defined by the *inputs*) and the Complex Data format requests (using the keys as defined by the *outputs*). The method is expected to return a dictionary of the output values (using the keys as defined by the *outputs*). In case of only one output item defined by the *outputs*, one output value is allowed to be returned directly.

identifier

An identifier (URI) of the process. Optional. When omitted it defaults to the process' class-name.

inputs

A dict mapping the inputs' identifiers to their respective types. The type can be either one of the supported native python types (automatically converted to a `LiteralData` object) or an instance of one of the data-specification classes (`LiteralData`, `BoundingBoxData`, or `ComplexData`). Mandatory.

metadata

A dict of title/URL meta-data pairs associated with the process. Optional.

outputs

A dict mapping the outputs' identifiers to their respective types. The type can be either one of the supported native python types (automatically converted to a `LiteralData` object) or an instance of one of the data-specification classes (`LiteralData`, `BoundingBoxData`, or `ComplexData`). Mandatory.

profiles

A iterable of URNs of WPS application profiles this process adheres to. Optional.

retention_period

This optional property (*datetime.timedelta*) indicates the minimum time the process results shall be retained after the completion. If omitted the default server retention policy is applied.

synchronous

Optional boolean flag indicating whether the process can be executed synchronously. If missing True is assumed.

title

A human-readable title of the process. Optional. When omitted it defaults to the process identifier.

version

The version of the process, if applicable. Optional. When omitted it defaults to '1.0.0'.

wSDL

A URL of WSDL document describing this process. Optional.

[eoxserver.services.ows.wps.test_allowed_values module](#)

[eoxserver.services.ows.wps.test_data_types module](#)

Module contents

Submodules

[eoxserver.services.ows.component module](#)

[eoxserver.services.ows.decoders module](#)

eoxserver.services.ows.interfaces module**class** `eoxserver.services.ows.interfaces.ExceptionHandlerInterface`Bases: `object`²⁸⁸

Interface for OWS exception handlers.

handle_exception (*request, exception*)The main exception handling method. Parameters are an object of the `django.http.Request` type and the raised exception.**request**

The supported request method.

serviceThe name of the supported service in uppercase letters. This can also be an iterable, if the handler shall support more than one service specifier. Some service specifications demand that the service parameter can be omitted for certain requests. In this case this property can also be `None` or contain `None`.**versions**

An iterable of all supported versions as strings.

class `eoxserver.services.ows.interfaces.GetServiceHandlerInterface`Bases: `eoxserver.services.ows.interfaces.ServiceHandlerInterface` (page 255)

Interface for service handlers that support HTTP GET requests.

class `eoxserver.services.ows.interfaces.PostServiceHandlerInterface`Bases: `eoxserver.services.ows.interfaces.ServiceHandlerInterface` (page 255)

Interface for service handlers that support HTTP POST requests.

class `eoxserver.services.ows.interfaces.ServiceHandlerInterface`Bases: `object`²⁸⁹

Interface for OWS Service handlers.

constraints

Optional property to return a dict with constraints for default values.

handle (*request*)The main handling method. Takes a `django.http.Request` object as single parameter.**index**

Optional. The index this service handler shall have when being reported in a capabilities document.

request

The supported request method.

serviceThe name of the supported service in uppercase letters. This can also be an iterable, if the handler shall support more than one service specifier. Some service specifications demand that the service parameter can be omitted for certain requests. In this case this property can also be `None` or contain `None`.**versions**

An iterable of all supported versions as strings.

class `eoxserver.services.ows.interfaces.VersionNegotiationInterface`Bases: `eoxserver.services.ows.interfaces.ServiceHandlerInterface` (page 255)

Interface for handlers that contribute to version negotiation.

²⁸⁸ <https://docs.python.org/2.7/library/functions.html#object>²⁸⁹ <https://docs.python.org/2.7/library/functions.html#object>

eoxserver.services.ows.version module

`eoxserver.services.ows.version.parse_version_string(version_string)`
Convenience function to parse a version from a string.

class `eoxserver.services.ows.version.Version` (*major, minor, revision=None*)
Bases: `object`²⁹⁰

Abstraction for OWS versions. Must be in the form 'x.y(.z)', where all components must be positive integers or zero. The last component may be unspecified (None).

Versions can be compared with other versions. Strings and tuples of the correct layout are also comparable.

Versions are compared by the "major" and the "minor" number. Only if both versions provide a "revision" it is taken into account. So Versions "1.0" and "1.0.1" are considered equal!

major

minor

revision

Module contents

Submodules

eoxserver.services.exceptions module

exception `eoxserver.services.exceptions.HTTPMethodNotAllowedError` (*msg, allowed_methods*)
Bases: `exceptions.Exception`²⁹¹

This exception is raised in case of a HTTP requires with unsupported HTTP method. This exception should always lead to the 405 Method not allowed HTTP error.

The constructor takes two arguments, the error message *msg* and the list of the accepted HTTP methods *allowed_methods*.

exception `eoxserver.services.exceptions.InterpolationMethodNotSupportedException`
Bases: `exceptions.Exception`²⁹²

This exception indicates a not supported interpolation method.

code = 'InterpolationMethodNotSupported'

locator = 'interpolation'

exception `eoxserver.services.exceptions.InvalidAxisLabelException` (*axis_label*)
Bases: `exceptions.Exception`²⁹³

This exception indicates that an invalid axis name was chosen in a WCS 2.0 subsetting parameter.

code = 'InvalidAxisLabel'

exception `eoxserver.services.exceptions.InvalidFieldSequenceException` (*msg, locator*)
Bases: `exceptions.Exception`²⁹⁴

Error in RangeSubsetting for illegal intervals.

code = 'InvalidFieldSequence'

²⁹⁰ <https://docs.python.org/2.7/library/functions.html#object>

²⁹¹ <https://docs.python.org/2.7/library/exceptions.html#exceptions.Exception>

²⁹² <https://docs.python.org/2.7/library/exceptions.html#exceptions.Exception>

²⁹³ <https://docs.python.org/2.7/library/exceptions.html#exceptions.Exception>

²⁹⁴ <https://docs.python.org/2.7/library/exceptions.html#exceptions.Exception>

exception `eoxserver.services.exceptions.InvalidOutputCrsException`

Bases: `exceptions.Exception`²⁹⁵

This exception indicates an invalid WCS 2.0 `outputCrs` parameter was submitted.

code = `'OutputCrs-NotSupported'`

locator = `'outputCrs'`

exception `eoxserver.services.exceptions.InvalidRequestException` (*msg*,
code=None,
locator=None)

Bases: `exceptions.Exception`²⁹⁶

This exception indicates that the request was invalid and an exception report shall be returned to the client.

The constructor takes three arguments, namely `msg`, the error message, `code`, the error code, and `locator`, which is needed in OWS exception reports for indicating which part of the request produced the error.

How exactly the exception reports are constructed is not defined by the exception, but by exception handlers.

exception `eoxserver.services.exceptions.InvalidScaleExtentException` (*low*,
high)

Bases: `exceptions.Exception`²⁹⁷

Error in `ScaleExtent` operations

code = `'InvalidExtent'`

exception `eoxserver.services.exceptions.InvalidScaleFactorException` (*scalefactor*)

Bases: `exceptions.Exception`²⁹⁸

Error in `ScaleFactor` and `ScaleAxis` operations

code = `'InvalidScaleFactor'`

exception `eoxserver.services.exceptions.InvalidSubsettingCrsException`

Bases: `exceptions.Exception`²⁹⁹

This exception indicates an invalid WCS 2.0 `subsettingCrs` parameter was submitted.

code = `'SubsettingCrs-NotSupported'`

locator = `'subsettingCrs'`

exception `eoxserver.services.exceptions.InvalidSubsettingException`

Bases: `exceptions.Exception`³⁰⁰

This exception indicates an invalid WCS 2.0 `subsetting` parameter was submitted.

code = `'InvalidSubsetting'`

locator = `'subset'`

exception `eoxserver.services.exceptions.LocatorListException` (*items*)

Bases: `exceptions.Exception`³⁰¹

Base class for exceptions that report that a number of items are missing or invalid

locator

This property provides a list of all missing/invalid items.

²⁹⁵ <https://docs.python.org/2.7/library/exceptions.html#exceptions.Exception>

²⁹⁶ <https://docs.python.org/2.7/library/exceptions.html#exceptions.Exception>

²⁹⁷ <https://docs.python.org/2.7/library/exceptions.html#exceptions.Exception>

²⁹⁸ <https://docs.python.org/2.7/library/exceptions.html#exceptions.Exception>

²⁹⁹ <https://docs.python.org/2.7/library/exceptions.html#exceptions.Exception>

³⁰⁰ <https://docs.python.org/2.7/library/exceptions.html#exceptions.Exception>

³⁰¹ <https://docs.python.org/2.7/library/exceptions.html#exceptions.Exception>

exception `ioxserver.services.exceptions.NoSuchCoverageException` (*items*)
Bases: `ioxserver.services.exceptions.LocatorListException` (page 257)

This exception indicates that the requested coverage(s) do not exist.

code = 'NoSuchCoverage'

exception `ioxserver.services.exceptions.NoSuchDatasetSeriesOrCoverageException` (*items*)
Bases: `ioxserver.services.exceptions.LocatorListException` (page 257)

This exception indicates that the requested coverage(s) or dataset series do not exist.

code = 'NoSuchDatasetSeriesOrCoverage'

exception `ioxserver.services.exceptions.NoSuchFieldException` (*msg, locator*)
Bases: `exceptions.Exception`³⁰²

Error in RangeSubsetting when band does not exist.

code = 'NoSuchField'

exception `ioxserver.services.exceptions.OperationNotSupportedException` (*message, operation=None*)
Bases: `exceptions.Exception`³⁰³

Exception to be thrown when some operations are not supported or disabled.

code = 'OperationNotSupported'

locator

exception `ioxserver.services.exceptions.RenderException` (*message, locator, is_parameter=True*)
Bases: `exceptions.Exception`³⁰⁴

Rendering related exception.

code

exception `ioxserver.services.exceptions.ScaleAxisUndefinedException` (*axis*)
Bases: `exceptions.Exception`³⁰⁵

Error in all scaling operations involving an axis

code = 'ScaleAxisUndefined'

exception `ioxserver.services.exceptions.ServiceNotSupportedException` (*service*)
Bases: `ioxserver.services.exceptions.OperationNotSupportedException` (page 258)

Exception to be thrown when a specific OWS service is not enabled.

exception `ioxserver.services.exceptions.VersionNegotiationException`
Bases: `exceptions.Exception`³⁰⁶

This exception indicates that version negotiation fails. Such errors can happen with OWS 2.0 compliant "new-style" version negotiation.

code = 'VersionNegotiationFailed'

³⁰² <https://docs.python.org/2.7/library/exceptions.html#exceptions.Exception>

³⁰³ <https://docs.python.org/2.7/library/exceptions.html#exceptions.Exception>

³⁰⁴ <https://docs.python.org/2.7/library/exceptions.html#exceptions.Exception>

³⁰⁵ <https://docs.python.org/2.7/library/exceptions.html#exceptions.Exception>

³⁰⁶ <https://docs.python.org/2.7/library/exceptions.html#exceptions.Exception>

exception `eoxserver.services.exceptions.VersionNotSupportedException` (*service*,
ver-
sion)

Bases: `exceptions.Exception`³⁰⁷

Exception to be thrown when a specific OWS service version is not supported.

code = `'InvalidParameterValue'`

eoxserver.services.models module

eoxserver.services.parameters module

class `eoxserver.services.parameters.CapabilitiesRenderParams` (*coverages*,
version, *sec-*
tions=None, *ac-*
cept_languages=None,
ac-
cept_formats=None,
updatese-
quence=None,
request=None)

Bases: `object`³⁰⁸

accept_formats

accept_languages

coverages

request

sections

updatesequence

version

class `eoxserver.services.parameters.RenderParameters`

Bases: `object`³⁰⁹

Abstract base class for render parameters

class `eoxserver.services.parameters.VersionedParams` (*version*)

Bases: `object`³¹⁰

version

eoxserver.services.result module

eoxserver.services.subset module

eoxserver.services.urls module

eoxserver.services.views module

³⁰⁷ <https://docs.python.org/2.7/library/exceptions.html#exceptions.Exception>

³⁰⁸ <https://docs.python.org/2.7/library/functions.html#object>

³⁰⁹ <https://docs.python.org/2.7/library/functions.html#object>

³¹⁰ <https://docs.python.org/2.7/library/functions.html#object>

Module contents

eoxserver.testing package

Submodules

eoxserver.testing.xcomp module

Simple XML documets' comparator.

exception `eoxserver.testing.xcomp.XMLError`

Bases: `exceptions.Exception`³¹¹

XML base error error

exception `eoxserver.testing.xcomp.XMLMismatchError`

Bases: `eoxserver.testing.xcomp.XMLError` (page 260)

XML mismatch error

exception `eoxserver.testing.xcomp.XMLParseError`

Bases: `eoxserver.testing.xcomp.XMLError` (page 260)

XML parse error

`eoxserver.testing.xcomp.xmlCompareDOMs` (*xml0, xml1, verbose=False*)

Compare two XML documents passed as DOM trees (`xml.dom.minidom`).

`eoxserver.testing.xcomp.xmlCompareFiles` (*src0, src1, verbose=False*)

Compare two XML documents passed as filenames, file or file-like objects.

`eoxserver.testing.xcomp.xmlCompareStrings` (*str0, str1, verbose=False*)

Compare two XML documents passed as strings.

Module contents

eoxserver.webclient package

Submodules

eoxserver.webclient.models module

eoxserver.webclient.urls module

eoxserver.webclient.views module

Module contents

Submodules

eoxserver.views module

`eoxserver.views.index` (*request*)

³¹¹ <https://docs.python.org/2.7/library/exceptions.html#exceptions.Exception>

Module contents

`eoxserver.get_version()`

EOxServer Open License

EOxServer Open License
Version 1, 8 June 2011

Copyright (C) 2011 EOX IT Services GmbH

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the “Software”), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies of this Software or works derived from this Software.

THE SOFTWARE IS PROVIDED “AS IS”, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

EOxServer-Soap Proxy Open License

Soap Proxy is Copyright (C) 2011 ANF DATA Spol. s r.o. Prague. The terms of the license are otherwise identical to those of the main EOxServer Open License.

CHAPTER 7

Credits



312

Work on EOxServer has been partly funded by the European Space Agency (ESA)³¹³ in the frame of the HMA-FO³¹⁴ and O3S³¹⁵ projects.

³¹² <http://rssportal.esa.int/tiki-index.php?page=Open%20Software>

³¹³ http://www.esa.int/esaMI/ESRIN_SITE/

³¹⁴ <http://wiki.services.eoportal.org/tiki-index.php?page=HMA-FO>

³¹⁵ <http://wiki.services.eoportal.org/tiki-index.php?page=O3S>

A

- AbstractStorageInterface (class in eoxserver.backends.interfaces), 230
 - accept_formats (eoxserver.services.parameters.CapabilitiesRenderParams attribute), 259
 - accept_languages (eoxserver.services.parameters.CapabilitiesRenderParams attribute), 259
 - add_band() (eoxserver.contrib.vrt.VRTBuilder method), 232
 - add_band() (eoxserver.contrib.vrt.VRTBuilder2 method), 233
 - add_simple_source() (eoxserver.contrib.vrt.VRTBuilder method), 232
 - add_simple_source() (eoxserver.contrib.vrt.VRTBuilder2 method), 233
 - add_to_package() (eoxserver.services.ows.wcs.interfaces.PackageWriterInterface method), 246
 - apply() (eoxserver.services.mapserver.interfaces.StyleApplicatorInterface method), 243
 - AsyncBackendInterface (class in eoxserver.services.ows.wps.interfaces), 253
 - asynchronous (eoxserver.services.ows.wps.interfaces.ProcessInterface attribute), 253
 - AuthorisationException, 239
 - authorize() (eoxserver.services.auth.interfaces.PolicyDecisionPointInterface method), 239
 - Autotest, 129
-
- B**
 - build() (eoxserver.contrib.vrt.VRTBuilder2 method), 233
 - build_sources() (eoxserver.contrib.vrt.VRTBuilder2 method), 233
-
- C**
 - cancel() (eoxserver.services.ows.wps.interfaces.AsyncBackendInterface method), 253
 - CapabilitiesRenderParams (class in eoxserver.services.parameters), 259
 - cleanup() (eoxserver.services.ows.wcs.interfaces.PackageWriterInterface method), 246
 - close() (eoxserver.contrib.vsi.TemporaryVSIFile method), 233
 - close() (eoxserver.contrib.vsi.VSIFile method), 234
 - closed (eoxserver.contrib.vsi.VSIFile attribute), 234
 - code (eoxserver.services.auth.exceptions.AuthorisationException attribute), 239
 - code (eoxserver.services.exceptions.InterpolationMethodNotSupportedException attribute), 256
 - code (eoxserver.services.exceptions.InvalidAxisLabelException attribute), 256
 - code (eoxserver.services.exceptions.InvalidFieldSequenceException attribute), 256
 - code (eoxserver.services.exceptions.InvalidOutputCrtsException attribute), 257
 - code (eoxserver.services.exceptions.InvalidScaleExtentException attribute), 257
 - code (eoxserver.services.exceptions.InvalidScaleFactorException attribute), 257
 - code (eoxserver.services.exceptions.InvalidSubsettingCrtsException attribute), 257
 - code (eoxserver.services.exceptions.InvalidSubsettingException attribute), 257
 - code (eoxserver.services.exceptions.NoSuchCoverageException attribute), 258
 - code (eoxserver.services.exceptions.NoSuchDatasetSeriesOrCoverageException attribute), 258
 - code (eoxserver.services.exceptions.NoSuchFieldException attribute), 258
 - code (eoxserver.services.exceptions.OperationNotSupportedException attribute), 258
 - code (eoxserver.services.exceptions.RenderException attribute), 258
 - code (eoxserver.services.exceptions.ScaleAxisUndefinedException attribute), 258
 - code (eoxserver.services.exceptions.VersionNegotiationException attribute), 258
 - code (eoxserver.services.exceptions.VersionNotSupportedException attribute), 259
 - code (eoxserver.services.ows.wms.exceptions.InvalidCRS attribute), 248
 - code (eoxserver.services.ows.wms.exceptions.InvalidFormat attribute), 249
 - code (eoxserver.services.ows.wms.exceptions.LayerNotDefined attribute), 249
 - Commit Management, 180
 - Configuration, 22

- Configuration Options, 106
- connect() (eoxserver.backends.interfaces.ConnectedStorageInterface attribute), 253
- connect() (eoxserver.backends.interfaces.ConnectedStorageInterface method), 230
- connect() (eoxserver.services.mapserver.interfaces.ConnectorInterface method), 242
- connect() (eoxserver.services.mapserver.interfaces.ConnectorInterface method), 242
- ConnectedStorageInterface (class in eoxserver.backends.interfaces), 230
- ConnectorInterface (class in eoxserver.services.mapserver.interfaces), 242
- constraints (eoxserver.services.ows.interfaces.ServiceHandlerInterface attribute), 255
- copy_gcps() (eoxserver.contrib.vrt.VRTBuilder method), 233
- copy_metadata() (eoxserver.contrib.vrt.VRTBuilder method), 233
- coverage (eoxserver.services.ows.wcs.parameters.CoverageRenderParams attribute), 247
- coverage_id (eoxserver.services.ows.wcs.parameters.CoverageRenderParams attribute), 247
- coverage_id_key_name (eoxserver.services.ows.wcs.parameters.CoverageRenderParams attribute), 247
- coverage_ids (eoxserver.services.ows.wcs.parameters.CoverageRenderParams attribute), 247
- coverage_ids_key_name (eoxserver.services.ows.wcs.parameters.CoverageRenderParams attribute), 247
- CoverageDescriptionRenderParams (class in eoxserver.services.ows.wcs.parameters), 247
- CoverageRenderParams (class in eoxserver.services.ows.wcs.parameters), 247
- coverages (eoxserver.services.ows.wcs.parameters.CoverageDescriptionRenderParams attribute), 247
- coverages (eoxserver.services.parameters.CapabilitiesRenderParams attribute), 259
- create_package() (eoxserver.services.ows.wcs.interfaces.PackageWriterInterface method), 246
- create_simple_vrt() (in module eoxserver.processing.gdal.vrt), 236
- Credits, 261, 263
- ## D
- Data Access Layer, 161, 177
- Data Integration Layer, 161, 176
- dataset (eoxserver.contrib.vrt.VRTBuilder attribute), 233
- Demonstration, 37
- Dependencies, 16
- Deployment, 23
- DescribeCoverage (Demonstration), 38
- DescribeCoverage (EO-WCS Request Parameters), 46
- DescribeEOCoverageSet (Demonstration), 39
- DescribeEOCoverageSet (EO-WCS Request Parameters), 46
- description (eoxserver.services.ows.wps.interfaces.ProcessInterface attribute), 253
- disconnect() (eoxserver.services.mapserver.interfaces.ConnectorInterface method), 242
- Distribution Core, 174
- dynamic binding, 161
- ## E
- EncodingExtensionInterface (class in eoxserver.services.ows.wcs.interfaces), 246
- EO-WCS Request Parameters, 45
- eoxserver (module), 261
- EOxServer Configuration, 22
- EOxServer Dependencies, 16
- EOxServer Deployment, 23
- EOxServer Installation, 15
- EOxServer Instance Creation, 18
- EOxServer Instances, 177
- EOxServer Migration, 33
- EOxServer Open License, 261
- EOxServer Service Instance Creation, 21
- EOxServer Upgrade, 33
- EOxServer Proxy Open License, 261
- eoxserver.backends (module), 231
- eoxserver.backends.interfaces (module), 230
- eoxserver.backends.packages (module), 229
- eoxserver.backends.storages (module), 229
- eoxserver.contrib (module), 235
- eoxserver.contrib.gdal (module), 231
- eoxserver.contrib.gdal_array (module), 231
- eoxserver.contrib.ogr (module), 231
- eoxserver.contrib.osr (module), 232
- eoxserver.contrib.vrt (module), 232
- eoxserver.contrib.vrt.vrt (module), 233
- eoxserver.processing (module), 237
- eoxserver.processing.gdal (module), 236
- eoxserver.processing.gdal.vrt (module), 236
- eoxserver.resources (module), 239
- eoxserver.resources.coverages (module), 239
- eoxserver.resources.coverages.metadata (module), 238
- eoxserver.resources.coverages.metadata.formats (module), 237
- eoxserver.resources.coverages.metadata.interfaces (module), 237
- eoxserver.resources.processes (module), 239
- eoxserver.services (module), 260
- eoxserver.services.auth (module), 240
- eoxserver.services.auth.exceptions (module), 239
- eoxserver.services.auth.interfaces (module), 239
- eoxserver.services.exceptions (module), 256
- eoxserver.services.gdal (module), 240
- eoxserver.services.gdal.wcs (module), 240
- eoxserver.services.gml (module), 240
- eoxserver.services.gml.v32 (module), 240
- eoxserver.services.mapserver (module), 243
- eoxserver.services.mapserver.connectors (module), 241
- eoxserver.services.mapserver.interfaces (module), 242

eooserver.services.mapserv.wcs (module), 241
 eooserver.services.mapserv.wms (module), 242
 eooserver.services.mapserv.wms.layerfactories (module), 241
 eooserver.services.mapserv.wms.styleapplicators (module), 242
 eooserver.services.native (module), 243
 eooserver.services.native.wcs (module), 243
 eooserver.services.ows (module), 256
 eooserver.services.ows.common (module), 244
 eooserver.services.ows.common.v11 (module), 243
 eooserver.services.ows.common.v20 (module), 244
 eooserver.services.ows.interfaces (module), 255
 eooserver.services.ows.version (module), 256
 eooserver.services.ows.wcs (module), 247
 eooserver.services.ows.wcs.interfaces (module), 246
 eooserver.services.ows.wcs.parameters (module), 247
 eooserver.services.ows.wcs.v10 (module), 244
 eooserver.services.ows.wcs.v11 (module), 245
 eooserver.services.ows.wcs.v20 (module), 245
 eooserver.services.ows.wcs.v20.encodings (module), 245
 eooserver.services.ows.wcs.v20.packages (module), 245
 eooserver.services.ows.wms (module), 250
 eooserver.services.ows.wms.exceptions (module), 248
 eooserver.services.ows.wms.interfaces (module), 249
 eooserver.services.ows.wms.v10 (module), 248
 eooserver.services.ows.wms.v11 (module), 248
 eooserver.services.ows.wms.v13 (module), 248
 eooserver.services.ows.wps (module), 254
 eooserver.services.ows.wps.exceptions (module), 251
 eooserver.services.ows.wps.interfaces (module), 253
 eooserver.services.ows.wps.processes (module), 250
 eooserver.services.ows.wps.v10 (module), 251
 eooserver.services.parameters (module), 259
 eooserver.testing (module), 260
 eooserver.testing.xcomp (module), 260
 eooserver.views (module), 260
 eooserver.webclient (module), 260
 ExceptionHandlerInterface (class in eooserver.services.ows.interfaces), 255
 execute() (eooserver.services.ows.wps.interfaces.AsyncBackendInterface method), 253
 execute() (eooserver.services.ows.wps.interfaces.ProcessInterface method), 254
 ExecuteError, 251
 extract() (eooserver.backends.interfaces.PackageInterface method), 231

F

filename (eooserver.contrib.vsi.VSIFile attribute), 234
 FileSizeExceeded, 251
 FileStorageInterface (class in eooserver.backends.interfaces), 230
 format() (eooserver.resources.coverages.metadata.interfaces.MetadataReaderInterface method), 237

format() (eooserver.resources.coverages.metadata.interfaces.MetadataReaderInterface method), 238
 formats (eooserver.resources.coverages.metadata.interfaces.MetadataWriterInterface attribute), 238
 from_buffer() (eooserver.contrib.vsi.TemporaryVSIFile class method), 234
 from_dataset() (eooserver.contrib.vrt.VRTBuilder class method), 233

G

GDALDatasetMetadataReaderInterface (class in eooserver.resources.coverages.metadata.interfaces), 237
 generate() (eooserver.services.mapserv.interfaces.LayerFactoryInterface method), 242
 generate_group() (eooserver.services.mapserv.interfaces.LayerFactoryInterface method), 242
 get_file_extension() (eooserver.services.ows.wcs.interfaces.PackageWriterInterface method), 246
 get_mime_type() (eooserver.services.ows.wcs.interfaces.PackageWriterInterface method), 246
 get_response_url() (eooserver.services.ows.wps.interfaces.AsyncBackendInterface method), 253
 get_status() (eooserver.services.ows.wps.interfaces.AsyncBackendInterface method), 253
 get_version() (in module eooserver), 261
 get_vrt_driver() (in module eooserver.contrib.vrt), 233
 GetCapabilities (Demonstration), 37
 GetCapabilities (EO-WCS Request Parameters), 45
 GetCoverage (Demonstration), 41
 GetCoverage (EO-WCS Request Parameters), 47
 GetServiceHandlerInterface (class in eooserver.services.ows.interfaces), 255
 Global Use Case, 5

H

handle() (eooserver.services.ows.interfaces.ServiceHandlerInterface method), 255
 handle_exception() (eooserver.services.ows.interfaces.ExceptionHandlerInterface method), 255
 http_status_code (eooserver.services.ows.wps.exceptions.NoApplicableCoverageException attribute), 252
 http_status_code (eooserver.services.ows.wps.exceptions.NotEnoughStorageException attribute), 252
 http_status_code (eooserver.services.ows.wps.exceptions.OWS10Exception attribute), 252
 http_status_code (eooserver.services.ows.wps.exceptions.ServerBusyException attribute), 252
 HTTPMethodNotAllowedError, 256

I

identifier (eooserver.services.ows.wps.interfaces.ProcessInterface attribute), 254
 index (eooserver.services.ows.interfaces.ServiceHandlerInterface attribute), 255
 index() (in module eooserver.views), 260
 inputs (eooserver.services.ows.wps.interfaces.ProcessInterface attribute), 254

Installation, 15
 Installation on CentOS, 18
 Instance Creation, 18, 21
 InterpolationMethodNotSupportedException, 256
 InvalidAxisLabelException, 256
 InvalidCRS, 248
 InvalidFieldSequenceException, 256
 InvalidFormat, 249
 InvalidInputError, 251
 InvalidInputReferenceError, 252
 InvalidInputValueError, 252
 InvalidOutputCrSException, 257
 InvalidOutputDefError, 252
 InvalidOutputError, 252
 InvalidOutputValueError, 252
 InvalidParameterValue, 252
 InvalidRequestException, 257
 InvalidScaleExtentException, 257
 InvalidScaleFactorException, 257
 InvalidSubsettingCrSException, 257
 InvalidSubsettingException, 257
 IsSame() (eoxserver.contrib.osr.SpatialReference method), 232

L

LayerFactoryInterface (class in eoxserver.services.mapserver.interfaces), 242
 LayerNotDefined, 249
 License, 261
 list_contents() (eoxserver.backends.interfaces.PackageInterface method), 231
 list_files() (eoxserver.backends.interfaces.FileStorageInterface method), 230
 locator (eoxserver.services.exceptions.InterpolationMethodNotSupportedException attribute), 256
 locator (eoxserver.services.exceptions.InvalidOutputCrSException attribute), 257
 locator (eoxserver.services.exceptions.InvalidSubsettingCrSException attribute), 257
 locator (eoxserver.services.exceptions.InvalidSubsettingException attribute), 257
 locator (eoxserver.services.exceptions.LocatorListException attribute), 257
 locator (eoxserver.services.exceptions.OperationNotSupportedException attribute), 258
 locator (eoxserver.services.ows.wms.exceptions.InvalidFormat attribute), 249
 locator (eoxserver.services.ows.wms.exceptions.LayerNotDefined attribute), 249
 LocatorListException, 257

M

Mailing List, 36
 major (eoxserver.services.ows.version.Version attribute), 256
 metadata (eoxserver.services.ows.wps.interfaces.ProcessInterface attribute), 254

MetadataReaderInterface (class in eoxserver.resources.coverages.metadata.interfaces), 238
 MetadataWriterInterface (class in eoxserver.resources.coverages.metadata.interfaces), 238
 Migration, 33
 minor (eoxserver.services.ows.version.Version attribute), 256
 MissingParameterValue, 252
 MissingRequiredInputError, 252

N

name (eoxserver.backends.interfaces.AbstractStorageInterface attribute), 230
 name (eoxserver.backends.interfaces.PackageInterface attribute), 231
 NoApplicableCode, 252
 NoSuchCoverageException, 257
 NoSuchDatasetSeriesOrCoverageException, 258
 NoSuchFieldException, 258
 NoSuchProcessError, 252
 NotEnoughStorage, 252

O

open() (in module eoxserver.contrib.vsi), 234
 OperationNotSupportedException, 258
 outputs (eoxserver.services.ows.wps.interfaces.ProcessInterface attribute), 254
 OWS10Exception, 252

P

PackageInterface (class in eoxserver.backends.interfaces), 231
 PackageWriterInterface (class in eoxserver.services.ows.wcs.interfaces), 246
 parse_encoding_params() (eoxserver.services.ows.wcs.interfaces.EncodingExtensionInterface method), 246
 parse_version_string() (in module eoxserver.services.ows.version), 256
 pause() (eoxserver.services.ows.wps.interfaces.AsyncBackendInterface method), 253
 pdp_type (eoxserver.services.auth.interfaces.PolicyDecisionPointInterface attribute), 240
 PolicyDecisionPointInterface (class in eoxserver.services.auth.interfaces), 239
 PostServiceHandlerInterface (class in eoxserver.services.ows.interfaces), 255
 Processing Chains, 173
 Processing Layer, 161, 175
 ProcessInterface (class in eoxserver.services.ows.wps.interfaces), 253
 profiles (eoxserver.services.ows.wps.interfaces.ProcessInterface attribute), 254

- proj (eoxserver.contrib.osr.SpatialReference attribute), 232
- Project Steering Committee (PSC) Guidelines, 153
- purge() (eoxserver.services.ows.wps.interfaces.AsyncBackendInterface method), 253
- Python Enhancement Proposals
PEP 333, 178
- ## R
- read() (eoxserver.contrib.vsi.VSIFile method), 234
- read() (eoxserver.resources.coverages.metadata.interfaces.MetadataReaderInterface method), 238
- read_ds() (eoxserver.resources.coverages.metadata.interfaces.GDALDatasetMetadataReaderInterface method), 237
- Recommendations for Operational Installation, 25
- Release Guidelines, 178
- render() (eoxserver.services.ows.wcs.interfaces.WCSCapabilitiesRendererInterface method), 246
- render() (eoxserver.services.ows.wcs.interfaces.WCSCoverageDescriptionRendererInterface method), 246
- render() (eoxserver.services.ows.wcs.interfaces.WCSCoverageRendererInterface method), 246
- render() (eoxserver.services.ows.wms.interfaces.WMSCapabilitiesRendererInterface method), 249
- render() (eoxserver.services.ows.wms.interfaces.WMSFeatureInfoRendererInterface method), 249
- render() (eoxserver.services.ows.wms.interfaces.WMSLegendGraphicRendererInterface method), 249
- render() (eoxserver.services.ows.wms.interfaces.WMSMapRendererInterface method), 249
- RenderException, 258
- RenderParameters (class in eoxserver.services.parameters), 259
- request (eoxserver.services.ows.interfaces.ExceptionHandlerInterface attribute), 255
- request (eoxserver.services.ows.interfaces.ServiceHandlerInterface attribute), 255
- request (eoxserver.services.parameters.CapabilitiesRenderParams attribute), 259
- requires_connection (eoxserver.services.mapserver.interfaces.LayerFactoryInterface attribute), 242
- resume() (eoxserver.services.ows.wps.interfaces.AsyncBackendInterface method), 253
- retention_period (eoxserver.services.ows.wps.interfaces.ProcessInterface attribute), 254
- retrieve() (eoxserver.backends.interfaces.FileStorageInterface method), 230
- revision (eoxserver.services.ows.version.Version attribute), 256
- ## RFC
- RFC 0, 153
- RFC 1, 156
- RFC 7, 178
- RFC 8, 180
- RFC Guidelines, 150
- RFC Policies, 148
- ## S
- ScaleAxisUndefinedException, 258
- sections (eoxserver.services.parameters.CapabilitiesRenderParams attribute), 259
- seek() (eoxserver.contrib.vsi.VSIFile method), 234
- ServerBusy, 252
- service (eoxserver.services.ows.interfaces.ExceptionHandlerInterface attribute), 255
- service (eoxserver.services.ows.interfaces.ServiceHandlerInterface attribute), 255
- ServiceHandlerInterface (class in eoxserver.services.ows.interfaces), 255
- ServiceNotSupportedException, 258
- set_geotransform() (eoxserver.contrib.vrt.VRTBuilder2 method), 233
- size (eoxserver.contrib.vsi.VSIFile attribute), 234
- software architecture
description, 160
layers, 161
overview, 158
release 0.1.1, 161
requirements, 159
- SpatialReference (class in eoxserver.contrib.osr), 232
- SRID (eoxserver.contrib.osr.SpatialReference attribute), 232
- StorageNotSupportedException, 252
- StyleApplicatorInterface (class in eoxserver.services.mapserver.interfaces), 243
- suffixes (eoxserver.services.mapserver.interfaces.LayerFactoryInterface attribute), 242
- suffixes (eoxserver.services.ows.wms.interfaces.WMSFeatureInfoRendererInterface attribute), 249
- suffixes (eoxserver.services.ows.wms.interfaces.WMSLegendGraphicRendererInterface attribute), 249
- suffixes (eoxserver.services.ows.wms.interfaces.WMSMapRendererInterface attribute), 249
- Supported CRSs and Their Configuration, 110
- Supported Raster File Formats and Their Configuration, 111
- supported_versions (eoxserver.services.ows.wps.interfaces.AsyncBackendInterface attribute), 253
- supports() (eoxserver.services.mapserver.interfaces.ConnectorInterface method), 242
- supports() (eoxserver.services.ows.wcs.interfaces.EncodingExtensionInterface method), 246
- supports() (eoxserver.services.ows.wcs.interfaces.PackageWriterInterface method), 246
- supports() (eoxserver.services.ows.wcs.interfaces.WCSCapabilitiesRendererInterface method), 246
- supports() (eoxserver.services.ows.wcs.interfaces.WCSCoverageDescriptionRendererInterface method), 246
- supports() (eoxserver.services.ows.wcs.interfaces.WCSCoverageRendererInterface method), 247
- swap_axes (eoxserver.contrib.osr.SpatialReference attribute), 232

- synchronous (eoxserver.services.ows.wps.interfaces.ProcessInterface attribute), 254
- T**
- tell() (eoxserver.contrib.vsi.VSIFile method), 234
- TemporaryVSIFile (class in eoxserver.contrib.vsi), 233
- test() (eoxserver.resources.coverages.metadata.interfaces.MetadataReaderInterface method), 238
- test_ds() (eoxserver.resources.coverages.metadata.interfaces.MetadataReaderInterface method), 238
- title (eoxserver.services.ows.wps.interfaces.ProcessInterface attribute), 254
- U**
- updatesequence (eoxserver.services.parameters.CapabilitiesRenderParams attribute), 259
- Upgrade, 33
- url (eoxserver.contrib.osr.SpatialReference attribute), 232
- Use Case, 5
- V**
- validate() (eoxserver.backends.interfaces.AbstractStorageInterface method), 230
- Version (class in eoxserver.services.ows.version), 256
- version (eoxserver.services.ows.wps.interfaces.ProcessInterface attribute), 254
- version (eoxserver.services.parameters.CapabilitiesRenderParams attribute), 259
- version (eoxserver.services.parameters.VersionedParams attribute), 259
- VersionedParams (class in eoxserver.services.parameters), 259
- VersionNegotiationException, 258
- VersionNegotiationFailed, 253
- VersionNegotiationInterface (class in eoxserver.services.ows.interfaces), 255
- VersionNotSupportedException, 258
- versions (eoxserver.services.ows.interfaces.ExceptionHandlerInterface attribute), 255
- versions (eoxserver.services.ows.interfaces.ServiceHandlerInterface attribute), 255
- VRTBuilder (class in eoxserver.contrib.vrt), 232
- VRTBuilder2 (class in eoxserver.contrib.vrt), 233
- VSIFile (class in eoxserver.contrib.vsi), 234
- W**
- warped_gcps() (eoxserver.contrib.vrt.VRTBuilder2 method), 233
- WCSCapabilitiesRendererInterface (class in eoxserver.services.ows.wcs.interfaces), 246
- WCSCapabilitiesRenderParams (class in eoxserver.services.ows.wcs.parameters), 247
- WCSCoverageDescriptionRendererInterface (class in eoxserver.services.ows.wcs.interfaces), 246
- WCSFeatureInfoRendererInterface (class in eoxserver.services.ows.wms.interfaces), 249
- WCSLegendGraphicRendererInterface (class in eoxserver.services.ows.wms.interfaces), 249
- WCSMapRendererInterface (class in eoxserver.services.ows.wms.interfaces), 249
- write() (eoxserver.contrib.vsi.VSIFile method), 234
- write() (eoxserver.resources.coverages.metadata.interfaces.MetadataWriterInterface method), 238
- wSDL (eoxserver.services.ows.wps.interfaces.ProcessInterface attribute), 254
- X**
- xml (eoxserver.contrib.osr.SpatialReference attribute), 232
- xmlCompareDOMs() (in module eoxserver.testing.xcomp), 260
- xmlCompareFiles() (in module eoxserver.testing.xcomp), 260
- xmlCompareStrings() (in module eoxserver.testing.xcomp), 260
- XMLError, 260
- XMLMismatchError, 260
- XMLParseError, 260